
PROGRAMMING FOR...

engineering students

... *or secondary school pupils*

beginners

... *or geeks*

Lectures of Ecole des Ponts ParisTech - 2023/2024

Renaud Keriven and Pascal Monasse

IMAGINE - École des Ponts ParisTech

pascal.monasse@enpc.fr

Electronic version and programs:

<http://imagine.enpc.fr/~monasse/Info/>

*“Do not consider your computers as living entities...
They don't like it”*

-
- *“This d*d computer does not do what I want!”*
 - *“Indeed... It does what you asked it to!”*
-

Contents

1	Preamble	7
1.1	Why learn programming?	9
1.2	How to learn?	10
1.2.1	Language choice	10
1.2.2	Choice of environment	10
1.2.3	Principles and advice	11
2	Hello, World!	13
2.1	The Computer	15
2.1.1	The microprocessor	15
2.1.2	The memory	17
2.1.3	Other Components	18
2.2	Operating System	20
2.3	The Compilation	21
2.4	The Programming Environment	22
2.4.1	File Names	22
2.4.2	Debugger	23
2.5	The bare minimum	23
2.5.1	To understand the practical session	23
2.5.2	A bit more...	25
2.5.3	The debugger	25
2.5.4	Practical Session	25
3	First programs	27
3.1	Everything in <code>main()</code> !	27
3.1.1	Variables	27
3.1.2	Tests	31
3.1.3	Loops	33
3.1.4	Recreations	34
3.2	Functions	36
3.2.1	Return	39
3.2.2	Parameters	40
3.2.3	Reference passing	41
3.2.4	Scope, Declaration, Definition	43
3.2.5	Local and global variables	44
3.2.6	Overload	45
3.3	Practical	46
3.4	Reference Card	47

4	Arrays	49
4.1	First arrays	49
4.2	Initialization	51
4.3	Specifics of arrays	52
4.3.1	Arrays and Functions	52
4.3.2	Assignment	54
4.4	Recreation	55
4.4.1	Multi-balls	55
4.4.2	With shocks!	57
4.4.3	Shuffling letters	59
4.5	Practical	61
4.6	Reference card	61
5	Structures	65
5.1	Reminders	65
5.1.1	Classic mistakes	65
5.1.2	Original mistakes	66
5.1.3	Advice	67
5.2	Structures	67
5.2.1	Definition	67
5.2.2	Usage	68
5.3	Recreation: Practical	70
5.4	Reference card	70
6	Several Files!	73
6.1	Separate Files	74
6.1.1	Principle	74
6.1.2	Advantages	75
6.1.3	Usage in another project	76
6.1.4	Header files	76
6.1.5	Don'ts...	79
6.1.6	Implementation	79
6.1.7	Mutual inclusions	79
6.1.8	Inclusion path	80
6.2	Operators	81
6.3	Fun: Practical continued and finished	82
6.4	Reference card	82
7	The memory	85
7.1	Call of function	85
7.1.1	Example	85
7.1.2	Call stack and debugger	87
7.2	Local variables	89
7.2.1	Parameters	89
7.2.2	The stack	89
7.3	Recursive functions	90
7.3.1	Why does it work?	90
7.3.2	Efficiency	91

7.4	The heap	92
7.4.1	Limits	92
7.4.2	Variable size arrays	93
7.4.3	Explanation (or trial of)	94
7.5	The optimizer	94
7.6	Assertions	95
7.7	Bidimensional arrays	95
7.8	Reference card	96
8	Dynamic Allocation	99
8.1	Bidimensional arrays	99
8.1.1	Principle	99
8.1.2	Limitations	100
8.1.3	Solution	101
8.2	Dynamic allocation	102
8.2.1	Why does it work?	102
8.2.2	Classical errors	103
8.2.3	Consequences	104
8.3	Structures and dynamic allocation	105
8.4	Loops and <code>continue</code>	108
8.5	Practical	109
8.6	Reference card	109
9	First objects	113
9.1	Philosophy	113
9.2	Simple example	114
9.3	Visibility	115
9.4	Example with matrices	116
9.5	Case of operators	118
9.6	Interface	120
9.7	Protection	121
9.7.1	Principle	121
9.7.2	Structures vs Classes	123
9.7.3	Accessors	123
9.8	Practical	123
9.9	Reference card	125
10	Constructors	129
10.1	The problem	129
10.2	The solution	130
10.3	General case	131
10.3.1	Empty constructor	131
10.3.2	Several constructors	132
10.3.3	Array of objects	133
10.3.4	Field of object type	134
10.4	Temporary objects	135
10.5	Practical	136
10.6	Constant References	136

10.6.1	Principle	136
10.6.2	Constant methods	137
10.7	Reference card	140
11	Destructor	145
11.1	Destructor	145
11.2	Destructors and arrays	147
11.3	Copy constructors	147
11.4	Assignment	148
11.5	Objects with dynamic allocation	150
11.5.1	Construction and destruction	150
11.5.2	Problems!	151
11.5.3	Solution!	152
11.6	Reference card	154
12	Strings, files	159
12.1	Strings	159
12.2	Files	161
12.2.1	Principle	161
12.2.2	String and file	162
12.2.3	Objects and files	163
12.3	Default values for parameters	164
12.3.1	Principle	164
12.3.2	Usefulness	164
12.3.3	Frequent errors	165
12.4	Accessors	165
12.4.1	Reference as return type	165
12.4.2	Usage	166
12.4.3	<code>operator()</code>	166
12.4.4	Overload and constant method	167
12.4.5	Inline functions	168
12.5	Assertions	169
12.6	Enumerated types	170
12.7	Reference card	170
13	Parameterized functions and classes (templates)	175
13.1	<code>template</code>	175
13.1.1	Principle	175
13.1.2	<code>template</code> and files	176
13.1.3	Classes	177
13.1.4	STL	179
13.2	Bitwise operators	181
13.3	Conditional values	183
13.4	Loops and <code>break</code>	183
13.5	Static variables	184
13.6	<code>const</code> and arrays	185
13.7	Reference card	186

A	Practicals	193
A.1	Programming environment	193
A.1.1	Hello, World!	193
A.1.2	First errors	195
A.1.3	Debugger	196
A.1.4	If there is time left	198
A.1.5	Install Imagine++ at home	198
A.1.6	Launching the program from the command line	198
A.2	Variables, loops, conditions, functions	199
A.2.1	First program with functions	199
A.2.2	First program with Imagine++	200
A.2.3	Tennis	202
A.3	Arrays	204
A.3.1	Text Mastermind	204
A.3.2	Mastermind in graphics mode	206
A.4	Structures, Separate Files	208
A.4.1	Part 1: Single file	208
A.4.2	Reorganization, suns galore	209
A.4.3	A third program: Duel	211
A.4.4	Help	212
A.4.5	Physics	213
A.5	Images	216
A.5.1	Allocation	216
A.5.2	Static arrays	216
A.5.3	Dynamic arrays	217
A.5.4	Load a file	217
A.5.5	Functions	218
A.5.6	Structure	218
A.5.7	Final clean-up	219
A.6	First objects and fractals	220
A.6.1	Sierpinski triangle	220
A.6.2	A class rather than a structure	221
A.6.3	Change the implementation	221
A.6.4	The snowflake	222
A.7	Tron	223
A.7.1	Snake	223
A.7.2	Tron	224
A.7.3	Graphics	224
A.7.4	Make the code great	225
B	Imagine++	227
B.1	Common	227
B.2	Graphics	228
B.3	Images	229
B.4	LinAlg	230
B.5	Installation	230

C	Compiler, CMake, Linker, Qt... Help!	233
C.1	Compilation	233
C.1.1	Compiler and linker	233
C.2	With command line	235
C.3	Make and CMake	235
C.4	Usage of an IDE	237
C.5	Configuration of Qt Creator	238
C.5.1	CMake and Qt	238
C.5.2	Kit	238
C.5.3	Project	239
D	Final reference card	241

Chapter 1

Preamble

Note: This clumsy first chapter corresponds to the environment when this course started in 2003, a time where computer science had a bad reputation at Ecole des Ponts. We keep it as testimony of what was required to entice students not to neglect computer science. If we go beyond the simplicity of this redaction (fortunately, the success of Star Wars endures through the years), the analysis and advice that follow are still current.

—

(This first chapter tries mostly to motivate engineering students in their learning of programming. Younger people who would happen to read this to learn programming are probably already motivated and can skip directly to next chapter!)

—

- *The Master Programmer:*¹ “Do not worry! Computers are stupid! Programming is thus easy.”
- *The Apprentice Programmer:*² “Master, computers are indeed only machines and mastering them should be possible for me. Still... Their lack of intelligence makes it difficult for me to have them do what I want. Programming requires precision and the least error is punished by an undecipherable message, a *bug*³ or even a *crash* of the machine. Why be so... precise?” *Programming makes someone maniac! Besides, programmers are all maniac. And I don’t want to turn like that...*
- *M.P.:* “Precision is mandatory to communicate with a machine. It is the human’s task to adapt. You must persevere. In return, you will become its master. Rejoice. Soon, you will create these obedient creatures programs are.”
- *A.P.:* “Fine, Master...” *What a lunatic! He takes himself as God almost. The truth is he talks to machines because he doesn’t know how to talk to people. He compensates with his computers his lack of human contact. The typical programmer... Only big spectacles and greasy hair are missing.*⁴ “Master, I am not sure I wish this. I am not able to

¹Please allow this term openly Lucasian (George Lucas was the inventor of Star Wars, before it was sold to Disney). It seems more appropriate that the usual *Guru*. We are really talking about a know-how to transmit from *Master* to *Apprentice* and not a sect...

²The young *Padawan*, thus, for those in the know...

³Our first term in the programmer’s vocabulary. You will meet others, it is important to get familiar with them.

⁴Any resemblance to real persons, etc.

do it. Do not take it badly, but I am much better in mathematics! Moreover, what will programming bring me?"

- *M.P.*: "The real problems you will face, you will not be able to solve always with mathematics only. Programming you will need!"
- *A.P.*: "I will try..." *I wonder if he is right! Surely, he is a failure in math. That must be the explanation!*
- ...

—

Let's leave here this dialog, as much caricatural as clumsy. It exposes still clearly the situation. Let's sum up:

- For the one who knows, programming:
 - is easy.
 - is necessary.
 - is a creative and fun activity.
- For the one learning, programming:
 - is difficult.
 - is useless.
 - is an ungrateful activity that promotes self-containment.⁵

In the case the student is an engineer, we can complete:

- For the instructor, learning to program:
 - should be simple and quick for engineering students.
 - is more useful than learning more math.
- For the student, programming:
 - is a task for "technicians"⁶ that others will take care of.
 - is not as noble as mathematics, in short, it is not worthy of the student.

Actually, both are wrong:

- The instructor:
 - does not realize the students have an advanced level in math because they have practiced them for more than ten years, and it will take time to learn even the basics of programming. Time... and practice, since, if programming is actually easy compared to what the students know in math, it requires a different way of thinking and much personal work in front of the machine.

⁵Using a computer to program has the same bad press as playing video games. Still, programming is often a team work.

⁶With all the subtext this term means for the student

- forgets he learned often alone, programming simple and playful things.⁷ The students must therefore be drawn toward programming by the playful side and not the same old examples.⁸

- The student:

- does not realize that to know programming will be useful. It is however a basis that comes again in all languages and even modern software.⁹ Later, considered as “the young one” hence the less allergic to computers, the student will be tasked on top of the first job with building a few little programs.
- is easily inclined to despise programming. It is simpler to learn a new branch of mathematics than to make the effort to acquire a new mindset.

As should be clear, it is at the same time easy and difficult to learn programming. For the *engineer*, it will take some effort and motivation: mostly to keep the math skills on the side and retrieve the taste for basic things. For the *secondary school pupil*, motivation and taste for effort will be present, but there will be a few bases of arithmetic to learn also. As announced in the title page of this course, secondary school pupils and engineers are at the same level to learn programming. Moreover, it is the same for the *beginner* and the *geek*. Let us explain: the geek has today so many things to do with the computer, knowing games, internet, graphics and music software, installation or system configuration, etc., but has no advantage relatively to programming. Not so long ago, apart from office work, there was little else to do with the personal computer other than programming, in order to complete the missing functionalities of the computer. Today, mastering the full capabilities of the computer is a full time job! At the end of the day, **all students are at equality**. The engineer should not be ashamed to learn programming at the same time as the neighbor’s child.

1.1 Why learn programming?

We already gave a few reasons. Let’s sum them up and complete:

1. It is the basics. Learning some specific language is not a waste of time since the same concepts appear in most languages. Moreover, event current software can be programmed.
2. An internship or a first job often involves some programming, even, or maybe even more, in environments where few people program.
3. Knowing programming, it is also a better understanding of hardware and software, what is technically possible or not. Even at a non technical position, it is important in order to take the right decisions.

⁷It is an error to believe the students will be interested with programs centered on mathematics or scientific computing. Such programs will maybe be useful to the students later but may not be so motivating. Linear algebra or numerical analysis are exciting to study... but certainly not to program. One has to admit without complex that programming a pinball, a mastermind of a 3D maze is as formative and more motivating than inverting a sparse matrix.

⁸The list is long but quite true: which programming course does not repeat the famous “factorial”, “Fibonacci sequence”, “Quick Sort”, etc?

⁹Knowing how to program is not only useful for C++, Java or Python, but also for Scilab, Matlab or Maple: advanced usage of Excel may require some programming!

1.2 How to learn?

1.2.1 Language choice

First, a programming language must be chosen. An engineer would be tempted to learn Maple, Matlab, Scilab or other. However, they are specialized tools for mathematicians and engineers, that indeed will prove useful and can also be programmed, but are not full fledged generalist programming languages. Without emphasizing the defects of these languages, it is obvious that they are not a good choice to learn programming.

In practice, the current choice is often between C++, Java, and Python. Even though Java was meant as a simplification of C++, among other motivations,¹⁰ and Python makes many complex things simple, we prefer C++ for pedagogical reasons:

1. C++ is more complex as a whole, but knowing its basics is already largely sufficient. We will see thus in this course only a subset of C++, sufficient in practice.
2. More complete, C++ allows high level as well as basic programming.¹¹ C++ allows programming close to the machine, which is important to the specialist but also for the beginner, because only a good understanding of the machine leads to a decent and efficient programming.¹²
3. C++ is the *lingua franca* in certain domains, such as finance.
4. At last, some practical and useful features of C++ have disappeared in Java and Python.¹³

In the last few years, the language that has risen more and more is Python. The reason is that it is portable, powerful and with a low cost of entry. It has still some drawbacks: it is constantly evolving, is not standardized, and compatibility between versions is not guaranteed.¹⁴ The data structures of Python are quite convenient, but hide the complexity of what is involved, especially concerning memory management, and an engineer has to understand how things work. Once again, let us repeat that the choice of language is not paramount, the essential thing is to learn programming.

1.2.2 Choice of environment

We could discuss at length the respective merits of the three main platforms, Windows, Mac, and Linux, and the best choice to learn programming. Actually, we do not have to prescribe some choice, as some tools are common to all. We think that for pedagogical reasons, an *integrated development environment* is more appropriate than multiple

¹⁰We do not reduce Java to a subset of C++, on certain things it is even better, but it has a more reduced expressivity.

¹¹Java forces to learn object oriented programming, which may be complex for the beginner.

¹²Many of these “details” are hidden from the Python programmer for ease of use. Not understanding what is required from the machine to execute a program leads to programs too greedy in time or memory.

¹³Overload of many operators, for example.

¹⁴The operation $3/4$ gave 0 in Python 2 (Euclidean division) and 0.75 in Java 3.

software tools (editor, compiler, debugger, etc.).¹⁵ It is crucial for the beginner. A development environment consists of the following features:

- All the steps of programming are grouped in a single tool in a coherent fashion.
- Editing files, transforming them into a program, finding errors, detecting bugs, browsing the documentation, etc. All that can be done with a single ergonomic tool.

Different development environments can be chosen. Both the most widespread platforms promote their own product: Visual Studio for Microsoft (Windows) and XCode for Apple (MacOS). They work on their own platform and cannot be used on another. Linux, being all about choice, proposes several, such as KDevelop. Anyway, using a portable environment is easier for a class, so that each can continue using their preferred platform. We advise Qt Creator, which, on top of being portable, has other advantages: keyboard shortcuts are the same as Visual Studio, and it understands natively CMake (which we will use in practical sessions).

This course is oriented toward Qt Creator. As the name indicates, it relies on Qt libraries for its portability and graphics. We think using directly Qt for graphics is too difficult for a beginner, and we use a less powerful but simpler layer on top of Qt, `Imagine++`. Moreover, under Windows, there is no C++ compiler by default,¹⁶ but installers of Qt and Qt Creator propose the MinGW compiler.¹⁷

1.2.3 Principles and advice

At the level we aim at teaching it, programming does not require a big theory or encyclopedic knowledge. The concepts are elementary but it is their application that can prove delicate. If we had a single piece of advice to give, it would be the *rule of the three "P"*:

1. **Program**
2. **Program**
3. **Program**

Practice is indeed essential. Let us still add a few more:

1. **Have fun.** It is obvious for pedagogy, but it is so easy in the case of programming that it would be a pity not to use it! At worst, if programming is not a pleasure for some, at least the obtained program, reached through suffering, is interesting!
2. **Tinker.** What we mean is you should not hesitate to test, sift, do, undo, break, etc. The computer is an experimental tool, and programming is also an experimental activity at its basis. Even if the experienced programmer will find the right solution at once, it is important for the beginner to learn to know the language and the programming tool by playing with them.

¹⁵Though many hard-liners learned with these individual tools and would not be lured to using an IDE.

¹⁶Under Linux, the usual and reference compiler is GCC (GNU Compiler Collection). Under Mac, it is Clang, whose interface mimics GCC, and is promoted by Apple.

¹⁷MinGW is a port of GCC to Windows.

3. **Do mistakes voluntarily.** Provoking errors during learning to know them better is the best way to understand many things and also to spot these errors when they are no longer made on purpose.
4. **Stay the master**¹⁸ (of the machine and of the program). That programming is experimental does not mean that we have to do anything until it works more or less. The advance should be progressive, methodical, testing along the way, without leaving the least error or imprecision.
5. **Debug.** Often, the knowledge of the debugger (the tool to find bugs) is neglected and its knowledge is reserved for advance usage. However, this tool is very convenient to understand what happens in a program, even in the absence of bugs. It should thus be considered as essential and an integral part of programming. Here again, a good development environment makes its usage easier.

—

Keep in mind these few principles since it now time to . . .

jump to our first program!

¹⁸The vocabulary is not random: a program is a sequence of *orders*, *commands* or *instructions*. We see who the boss is!

Chapter 2

Hello, World!

(If some secondary school pupils reached this point, they are brave! When I said that they could easily learn programming, I really meant it. Though it was with a bit too much optimism that I pretended they could do it while reading lecture notes written for engineers. Well, I shot myself in the foot! So, I will try to explain along the road the mathematics that could be obscure to them.)

—

If you believe many programming manuals, the first program should look like this:

```
#include <iostream>
using namespace std;

int main() {
    cout << "Hello ,_World!" << endl;
    return 0;
}
```

Very well, let's go! Let's analyze it! In this program, that outputs on screen¹ the text "Hello, World!", lines 1 and 2 are *magical* instructions² that allow using later `cout` and `endl`. Line 4 `int main()` defines a *function* called `main()`, that *returns*³ an integer number.

This function is special since it is the main function of the C++ program, that is called automatically⁴ when the program is *launched*⁵. Delimited by curly brackets ({

¹That expression, remains of the time where computers had screen capable of displaying characters only and not *graphics* (curves, images, etc.), means today that the display will occur in a window simulating the screen of that time. This window is called a terminal, **console**, command window, DOS window, xterm, etc. according to cases. Let's remember with utmost respect that it was already a progress compared to previous generation, without screen and that used a printer to communicate with humans... which was relatively poor interactivity!

²Let's say instructions that we won't explain right now. There is (un)fortunately no magic in programming.

³To whom? To the one that called it, stupid!

⁴Hear, now you know who called `main()`. In a program, functions call each other, but `main()` is called by none since it is the first of them all. (At least apparently because actually the program has plenty to do before arriving in `main()` and it begins by several other functions that the programmer needs not know and that end up calling `main()`. Besides, if nobody called it, to whom would `main()` return an integer?)

⁵I knew that intending to explain all barbarisms specific to computer scientists would impede me

line 5 and } line 8), the function `main()` terminates line 7 with `return 0`; that mandates to return the integer value 0. Note along the way that all instructions end with a semi-colon `;`. Finally, at line 6, the only “interesting” one, `cout << "Hello,_World!" << endl;` displays, thanks to the *variable*⁶ `cout` corresponding to the *console*⁷, data separated by `<<`. The first piece of data is the *character string*⁸ `"Hello,_World!"`. The second one, `endl`, is a *line end*⁹.

Phew! So many terms in italics! So many concepts to explain! All for such a simple program! But this is not the problem. Beginning to explain such a program, it is still being in the void, in magics, in the abstract, in the approximate. We do not really master the machine and seeing what happens without **understanding what happens** is not sensible. In fact, it is even damaging for what follows. We do not order someone without understanding how it works and what the given order involves as work. Likewise,

we do not program suitably without understanding what the computer will need to do to execute the program.

It is this whole approach that is neglected when we begin like we did. Hence,...

Stop! Wrong start! Let's start again the:

much. Whatever. Therefore, a program *starts* or *is launched*. After which, it *executes* or *runs*. At last, it *terminates* or *dies*.

⁶The *data* is stored in *variables* that memorize *values*. These *variables* are not always variable in the usual sense, since some may be constant!

⁷What did I say! We display in a console window!

⁸To be clear, some text.

⁹Which means that next displays, if any, will happen on a new line.

Chapter 2 (second trial)

How does it work?

The problem with the preceding program is that it is very far from what a computer is able to do naturally. Actually, a computer doesn't talk C++. It knows only numbers,¹⁰ transforming numbers into other numbers. Though hardly understandable for the beginner, a C++ program intends to be as close as possible to the human, whereas being accessible to the machine.¹¹ C++ is a very complete language, maybe too much. It can be relatively close to the machine when necessary and on the contrary be "high level" when needed. The width of its spectrum is one of the reasons of its success. Which is also what makes its complete learning a long work and we will study only a restricted part of C++!

2.1 The Computer

To know what a computer can really do, we need to begin with its principal organ: the microprocessor.

2.1.1 The microprocessor

Whatever its architecture¹² and whatever its speed,¹³ a microprocessor can only do somewhat basic things. Without being exhaustive, let's just remember this:

- It knows how to execute a sequential list of instructions.
- It owns a small number of internal memory units called registers.
- It communicates with outside world through memory¹⁴ in much larger quantity than its registers.
- This memory holds, in the form of numbers, the instructions to execute as well as the data on which to work.
- The instructions are typically:

¹⁰A *computer*, indeed!

¹¹This notion is obviously dependent on our know-how for the time being. First programming languages were further away from humans since closer to the machine, which was rather rudimentary, and we can expect future languages to be closer to humans.

¹²Intel Pentium derivative or other

¹³More precisely the frequency at which it executes its instructions. Today, the clock runs at roughly 3 GHz. (But beware: instructions may take more than one clock cycle!)

¹⁴Today, at least 4 GB (gigabytes), that is, $4 \times 1024 \times 1024 \times 1024$ memory units of 8 bits (each unit able to hold a number between 0 and 255). Notice that computer scientists count prefix of bytes with basis 1024 and not 1000, as everybody else! Why? Because $1024 = 2^{10}$ is the closest power of 2 to $1000 = 10^3$, and it is more logical to have as basis 2 than 10 for a computer.

- Read or write a number in a register or in memory.
- Do elementary operations: addition, multiplication, etc.
- Test of compare values and decide possibly to jump to another part of the instruction list.

Here is for example what the microprocessor must do when we require it to execute "c=3*a+2*b;" in C++, where *a*, *b*, *c* are three integer variables:

```
00415A61  mov  eax,dword ptr [a] // put in register eax
                                // the contents of the address where
                                // variable a is memorized
00415A64  imul eax,eax,3          // do eax=eax*3
00415A67  mov  ecx,dword ptr [b] // idem put b in ecx
00415A6A  lea  edx,[eax+ecx*2]    // do edx=eax+ecx*2
00415A6D  mov  dword ptr [c],edx // put the contents of register edx
                                // to the address where variable c
                                // is stored
```

This program is called *Machine Code*. The first number of each line is an address. We will come back to this later. Apart from it, the rest is quite readable for a human (attention, that is me who added the remarks on the right!). This is because it is a program in *assembler* language, a language where each instruction is really a microprocessor instruction, but where the name of the instructions and their arguments are explicit. In reality, the microprocessor does not understand assembler. Understanding "mov eax,dword ptr [a]" would require not only decoding the symbols but also where the variable *a* is stored. The real language of the microprocessor is the *machine language*, in which the instructions are numbers. Here is what it yields for our "c=3*a+2*b;":

```
00415A61 8B 45 F8
00415A64 6B C0 03
00415A67 8B 4D EC
00415A6A 8D 14 48
00415A6D 89 55 E0
```

Apart once again from the left column, each number list¹⁵ corresponds obviously to a single precise instruction. It is quite less readable!¹⁶ Let us note that each microprocessor model has its own *instruction set*¹⁷ which means that the translation of c=3*a+2*b; in the sequence of numbers 8B45F86BC0038B4DEC8D14488955E0 is specific to the Pentium architecture (known as x86) that we used for our example:

Once translated in machine language, a C++ program makes sense only for this microprocessor.

Let us notice also that the architects of Pentium decided to create a specific instruction to compute $edx = eax + ecx * 2$ in one shot since it is very common. If we asked $c = 3 * a + 3 * b$;, our program would have become:

¹⁵Numbers a bit weird, all right, since they contain letters. Be patient, young *Padawan!* We will come back to it right away!

¹⁶Nevertheless, computer scientists programmed like that not so long ago. That was already very good compared to the previous time when it was required to program in base 2... and a lot less good than when they could at last program in assembler!

¹⁷The standard acronym is ISA, *Instruction Set Architecture*.

```

00415A61 8B 45 F8    mov    eax,dword ptr [a]
00415A64 6B C0 03    imul  eax,eax,3
00415A67 8B 4D EC    mov    ecx,dword ptr [b]
00415A6A 6B C9 03    imul  ecx,ecx,3
00415A6D 03 C1      add    eax,ecx
00415A6F 89 45 E0    mov    dword ptr [c],eax

```

as "lea edx, [eax+ecx*3]" does not exist! But let's come back to our numbers...

2.1.2 The memory

The internal memory of our microprocessor is handled by registers, comparable to variables of C++, but their number is predefined. To *store* the sequence of instructions to execute, one has to use memory in much higher quantity, known as *the main memory of the computer*. It is the famous "*memory modules*"¹⁸ of memory that we can buy to increase the capacity of our machine and whose price fluctuates more strongly than other components of a computer. This memory is split into bytes. One byte¹⁹ corresponds to a binary number of 8 bits,²⁰ hence $2^8 = 256$ possible numbers. To locate oneself in memory, it is out of question to give names to each byte. They are simply numbered and we get *memory addresses*. The numbers 00415A61, etc. from above were addresses! At the beginning, these numbers were written in binary, which matched exactly what the computer understands. It became unwise when the size of memory reached above a few hundreds of bytes. The content of a byte in memory being itself stored in the form of a number, a counting system was adopted, which is convenient for 8 bits: rather than writing numbers in binary, the choice of base 16 allows representing the content of a byte with two digits (0,1,...,9,A,B,C,D,E,F). The *hexadecimal system*²¹ was adopted... Conversions from binary to hexadecimal are simple, each hexadecimal digit being worth a packet of 4 bits, whereas between binary and decimal, it is more complex. This system is still used when we designate the content of a byte or an address.²² In that manner, our famous $c=3*a+2*b$; becomes in memory:²³

¹⁸Today, typically modules of 2 to 8GB. Let's have a thought for the first PC that had 640KB (kilobytes, that is, 1024 bytes), even for the older ones among us about the first personal computers with 4KB, or even the first programmable cards with 256 bytes!

¹⁹Beware to distinguish byte and bit, overall in abbreviations like 100Mb/s for internet access debit... b=bit, B=byte=8 bits

²⁰**For secondary school pupils:** in binary, or base 2, we count with two digits instead of the usual ten (i.e., in decimal or base 10). It gives: 0, 1, 10, 11, 100, 101, 110, 111, ... So that 111 in binary is 7 in decimal. Each digit is called a bit. We see easily that with one digit we count from 0 to 1, two possible numbers; with two digits, from 0 to 3, hence $4 = 2 \times 2$ numbers; with 3 digits, from 0 to 7, hence $8 = 2 \times 2 \times 2$ numbers. In short with n bits, we can code 2^n (2 multiplied by itself n times) numbers. I can remember having learned base 2 in kindergarten with plastic cubes! Strange school program. And I do not say that to find an excuse for turning up computer scientist. Though...

²¹**For secondary school pupils (again):** in base 16, or hexadecimal, we count with 16 digits. This needs inventing digits beyond 9 and we take A,B,C,D,E,F. When we count, we get: 0, 1, 2, ..., 9, A, B, C, D, E, F, 10, 11, 12, 13, ..., 19, 1A, 1B, 1C, ... Hence 1F in hexadecimal is worth 31. With 1 digit, we count from 0 to 15, that is, 16 possible numbers; with 2 digits, from 0 to 255, hence $256 = 16 \times 16$ possible numbers, etc. A byte can be written with 8 bits in binary, or 2 digits in hexadecimal, and goes from 0 to 255, or 11111111 in binary, or FF in hexadecimal.

²²In the latter case case, with more than 2 digits: 8 for processors 32 bits, 16 for processors 64 bits.

²³You may notice that instructions can have different lengths.

memory address	content	represents
00415A61	8B	mov eax, dword ptr [a]
00415A62	45	
00415A63	F8	
00415A64	6B	imul eax, eax, 3
00415A65	C0	
...	...	

The memory is not only useful to store the sequence of instructions to execute but also all variables and data of the program, the registers of the microprocessor are not enough. Therefore our variables a , b , c are stored somewhere in memory with a sufficient number of bytes²⁴ to represent integers (here 4 octets) and at a location decided by the C++, so that the instruction `8B45F8` fetches the variable a ! It is hard work, that the C++ does for us and that programmers had to do by hand in the past.²⁵ In short, we have furthermore:²⁶

memory address	content	represents
...	...	
00500000	a_1	a
00500001	a_2	
00500002	a_3	
00500003	a_4	
00500004	b_1	b
00500005	b_2	
00500006	b_3	
00500007	b_4	
...	...	

where bytes a_1, \dots, a_4 , when combined, give the integer a on 32 bits. Some processors (called *big-endian*)²⁷ decide $a = a_1a_2a_3a_4$, others (*little-endian*)²⁸ $a = a_4a_3a_2a_1$. It means that

So as instructions, a number written by one microprocessor in a file can be unreadable by another microprocessor that reads the file!

2.1.3 Other Components

Microprocessor and memory: we have seen the two main blocks. Let us complete the table with other important components of the computer.

²⁴Variables having more than 256 possible values need to be stored on several bytes. With 4 bytes we can count in binary on $4 \times 8 = 32$ bits, hence 2^{32} possible values (more than 4 billions).

²⁵The hardest part was not to decide where to store the variables, but to adjust the instructions accordingly. If we made a mistake, we could write at the wrong location in memory. At best, it erased another variable—this behavior is still possible nowadays—, at worst, it erased instructions and the program could do “big mistakes”—that is now impossible under Windows or Linux, and only still a danger on certain low-end systems.

²⁶We lie horribly here for a simplification purpose. In our case, the variables were local variables to the function `main()` hence stored on the *stack*. They are not at a predefined memory location but at an address relative to where the function will store its local variables, depending on what the program will have done before. This explains the simplicity of the instruction `mov eax, dword ptr [a]` in our case. We will see all that later.

²⁷Such as the PowerPC of older time Macs

²⁸The majority, such as processors Intel and AMD

Memory types

The memory we talked about is known as RAM (Random Access Memory). It is fast²⁹ but has the bad idea to be erased when we switch off the computer. We also need ROM (Read Only Memory), that is memory holding its data when the computer is switched off but that cannot be modified³⁰. This memory contains in general the minimum the computer needs to start and executes a predefined task. Initially, it stored instructions so that the programmer could fill later the RAM with the instructions of the program. It was required to input the program each time!³¹ Quite soon, *recording media* were used to save programs and data when the computer was switched off. It was enough then to put in ROM the necessary capability to handle these storage devices.

Recording media

Some allow to read data, others also to write. Some deliver data only in order, sequentially, others in the order we choose, randomly. They are in general much slower than main memory and it should be remembered! We copy in RAM the part of the recording media on which we work.

Make the microprocessor work with the hard drive is MUCH slower than with main memory (1000 times slower in access time, 100 times in throughput)^a

^aAdd a factor 50 between main memory and memory cache of the processor!

At the beginning, storage media were mechanical: cards or bands. Then they became magnetic: mini-tapes³², disks³³ or hard drives³⁴. Today, we can have CD, DVD, memory cards, "USB drives", etc.

Peripherals

We call peripherals different devices connected to the computer: keyboard, mouse, screen, printer, modem, scanner, etc. They were initially present to serve as interface with the user, as input and output between the microprocessor and the real world. Nowadays, it becomes more difficult to see things that way. For instance, graphics cards, which were a peripheral coming with the screen, have become themselves essential parts of the computer, true workpowers, to the point that an increasing number of programs use them to perform computations without ever displaying anything on the screen. Even more, it is the computer itself that can be considered as a link between

²⁹Though less than registers, or even than the memory cache of the microprocessor, which we won't talk about.

³⁰It is unfortunate that a ROM cannot be modified. Hence, once, we used memory that still could be modified, but with specialized hardware (EPROMS). Nowadays, we use often memory that can be modified by software ("flash" memory) or, for very small amounts of data, energy efficient memory (CMOS) completed with a small battery. In a PC, the memory that is used to start the computer is the BIOS. It can be flashed and its parameters are in CMOS. Beware of the usage of the battery!

³¹Each time the computer was switched on but also each time the program crashed and erased itself, which was most of the time!

³²Very slow and low reliability, but the everyday of PC at the time.

³³Some luxury. A reader of 40KB cost 5000F or 1000 euros!

³⁴First ones were true motor vehicles, reserved for big computer centers.

different devices. Who would call peripheral a camera that is linked to a computer to send the video on the internet or transfer them on DVD? The computer would almost be a peripheral of the camera!

2.2 Operating System

Our vision at this point is the following:

1. The processor starts with instructions present in ROM.
2. These instructions allow it to read other instructions written on hard drive, that are copied into RAM.
3. It executes these instructions pour read input data present on the disk and generate new data (output). Unless input or output were exchanged through peripherals.

Quickly, this principle evolved:

1. The contents of the hard drive was organized in files. Some files represented data³⁵, others programs³⁶, still others contain themselves files³⁷.
2. Processors becoming faster and hard drive capacity increasing, we wanted to handle several programs and execute several: one after the other then simultaneously (multi-tasking) and even for several users simultaneously (multi-user)³⁸, finally with several processors per machine.

To handle all this complexity, the concept of *Operating System* (OS) emerged. Windows, Unix (include notably Linux) and MacOS are the most widespread.³⁹ The OS is responsible today of handling files, interfacing with peripherals or users,⁴⁰ but its most delicate responsibility is to handle the programs (or task or *process*) executing at once. It has to face two problems:⁴¹

1. Make the processor work successively by small time intervals on different programs. It has to give the hand smartly and fairly, but also to resume an interrupted process in the same state as when the interruption occurred.
2. Handle the memory of each process. In practice, an adjustable part of the memory is reserved for each process. The memory of a process becomes *virtual memory*: if a process is moved to another location of the *physical memory* (RAM), it does not notice. The OS can even put temporarily outside the RAM (hence on hard drive) an idle process. With that principle, the hard drive can even be used as memory for a process requiring more than the physical memory: but beware, the hard drive is very slow, the process can become hardly responsive.

³⁵Most were text, where each byte represented a character. That was the famous ASCII (65 for A, 66 for B, etc.). At the multimedia era, formats are numerous, concurrent, and more or less normalized.

³⁶We talk about *executable* files...

³⁷The folders.

³⁸Today, it is even worse. A program can be several parts executing simultaneously (*the threads*). Concerning the processor, it executes all the time several instructions at once (it is said *super-scalar*)!

³⁹Android, based on Linux kernel, in smartphones and iOS in iPhones

⁴⁰Let's hope the users won't become themselves peripherals!

⁴¹The processors obviously evolved to help the OS to do that efficiently.

When a process needs too much memory, the OS can decide without warning to use hard drive space as additional logical memory and become quite slow. We say it *swaps*. Only the slowness (and possibly the sound of hard drive working at full speed!) allows to realize that (this can be checked with the task manager of the system).

Another progress: the virtual memory is separated between the different processes, and inside a given process, the memory containing instructions from the memory for data. It is strictly impossible that a faulty process could modify its instructions or the memory of another process by writing at a wrong memory location.⁴²

With the arrival of operating systems, executable files should adapt for numerous reasons of management and sharing of memory. In practice, a Linux executable cannot run under Windows and reciprocally, even if they contain instructions for the same processor.

An executable file is specific not only to a processor type, but also to a given OS.

At best, successive versions of a processor family try to continue to understand instructions of its predecessors, so as successive versions of a software try to be able to read data produced by earlier versions, different versions of an OS try to be able to execute programs written for earlier versions. This is called *ascending compatibility*, that may cost more complexity and slowness.

2.3 The Compilation

While we tried to understand what happens below the surface to draw some useful info like the memory handling, we have glimpsed that the transformation of a C++ program into an executable file is a difficult task. Some software dispose of their own language, such as Maple, Scilab or Matlab, the Python interpreter, and do not translate their programs into machine language. The translation is done when the program is executed and analyzed along the way:⁴³ we call it an *interpreted language*. Execution is then quite slow. Other languages, like Java, decide to deal with *portability* problems, that is, the dependency on the processor and on the OS, by inserting an intermediate layer between the processor and the program: the *virtual machine*. This machine, obviously written for a given processor and system, can execute programs in a virtual machine language,⁴⁴ the “byte code”. A Java program is then translated into its equivalent in that machine language. The result can be executed on any virtual Java machine. The drawback of such an approach is obviously a loss of efficiency.

The translation of native code or byte code of a program is called *compilation*.⁴⁵ In the case of C++ and of most compiled languages (Fortran, C, etc), the compilation

⁴²It can only modify savagely its own data, which is already sufficient havoc!

⁴³even if it is sometimes preprocessed to accelerate execution

⁴⁴By contrast, the “true” machine language of the processor is then called *native code*.

⁴⁵The principles of compilation are one fundamental topic of computer science, traditional and very instructive. When one knows how to write a compiler, one can program everything. (Obviously, a compiler is a program. It is compiled with the previous compiler! Same thing for operating systems...) It requires a full course by itself and is quite out of the scope of an introduction to programming!

happens to native code. It transforms a *source* file, the C++ program, into an *object* file, sequence of instructions in native language.

Nevertheless, the object file by itself is not self-sufficient. Some additional instructions are necessary to build a full executable file:

- something to launch the `main()`! More precisely, everything the process needs to run before and after the execution of `main()`.
- functions or variables that belong to the language and that the programmer uses with programming them, such as the object `cout`, the function `min()`, etc. The set of these instructions constitutes a so-named *library*.⁴⁶
- functions or variables written by the programmer in other source files, then compiled into object files, but that the programmer wants to use in the program.

The synthesis of these files to an executable file is called the **link edition**. The program dealing with that is called *linker*.

To sum up, the production of an executable file happens in two steps:

1. **Compilation: source file → object file (for each source file).**
2. **Link: object files + standard library and other libraries → executable file.**

2.4 The Programming Environment

The programming environment⁴⁷ is the software allowing to program directly. In our case it is *Qt Creator*. In other cases, it can simply be a set of programs. An environment includes at minimum an *editor* to write source files (the knowledge of the language by the editor can help the programmer write syntactically correct code), a *compiler/linker* to write executables, a *debugger* to track programming errors, and a *project manager* to manage the different source and executable files.

The reader is here invited to read the text of first practical session and the Appendix C. On top of a few rudimentary C++ notions we will see in next chapter, some supplementary information is useful to understand it.

2.4.1 File Names

The *extension* (the suffix) is useful to differentiate the different file types:

- A C++ source file (your code!) ends with `.cpp`.⁴⁸
- An object file ends with `.obj` (Windows) or `.o` (Linux/Mac).
- An executable file ends with `.exe` (Windows) or with no extension (Linux/Mac).

⁴⁶A library is actually a set of object files (hence already compiled) in a single file. It can be a library of functions furnished by C++, then called standard library, but also an additional library furnished by someone else.

⁴⁷often called IDE, Integrated Development Environment.

⁴⁸a file in `.c` is considered as C code.

We will also meet later:

- The “header” C++ files that are included in other source files: files in `.h`
- The libraries: files `.lib` or `.dll` (Windows), `.a` or `.so` (Linux/Mac)

2.4.2 Debugger

When a program misbehaves, one can try to understand what goes wrong by stuffing the source code with instructions to output the value of some variables or simply to follow its run. Obviously, it is not very practical. It is better to be able to follow instruction by instruction and display variable values on demand. That is the role of the debugger.

For an interpreted language (such as Python), it is fairly easy to run the program step by step since it is the language itself that runs the program. In the case of a compiled language, it is the microprocessor that executes the program and one cannot stop it at each instruction! One needs then to put *breakpoints* by modifying temporarily the machine code of the program so that the microprocessor stops when it reaches the instruction corresponding to the source line to debug. Whereas it is complex to set up, it is very easy to use, especially in a graphical programming environment.

We will see along the practical sessions how the debugger can also check function calls, spy on variable changes, etc.

2.5 The bare minimum

2.5.1 To understand the practical session

Here is a strictly minimal C++ program, written in a file `main.cpp`:

```
int main() {}
```

The reserved word `main` indicates the entry point of the program. It is a function (a block of code). Like a mathematical function, it takes one or several input and gives an output. For this function, there is nothing between the parentheses, meaning there is no input argument. On the contrary, it returns an integer value (type `int` like integer). This function does nothing, the list of instructions is empty (block between curly brackets). But where does it specify the value returned? This function returns value 0, which means by convention that everything went normally. To be more explicit, we would have written:

```
int main() {  
    return 0;  
}
```

which has exactly the same effect, since function `main` returns 0 by default. But the case of this function is special, it is better to be more explicit and to indicate that the function returns 0. Let us note that we wrote the `return` on a new line, that every instruction ends with a mandatory semicolon, and that we shifted the instruction to the right (we say “indented”) to show that we are inside the block. Actually, the compiler is indifferent to line returns and to indentations, they are present uniquely to facilitate

the readability of the code for us, the programmer. However, do not consider them as optional, it is very important to write a clean and readable code. Those who know Python remember that it does not use curly brackets for blocks but uses indentation, that becomes mandatory. The fact it is not in C++ does not mean that everything goes equally. Let us insist that indenting properly is important.

This program does nothing, but we still need to check it is correct, meaning it follows the syntactic rules and that it is enough to create an executable. For that, we will use `CMake`, which has the advantage to be multi-platform and compatible with all widespread development environments. Let us first create a file `CMakeLists.txt` containing:

```
add_executable(EssaiQtCreator main.cpp)
```

It indicates we have a source code, `main.cpp`, and that our executable will be called `EssaiQtCreator`. We create this file in the same folder as source `main.cpp` (source folder). We launch Qt Creator and choose the option “Open file or project” in menu `File`. We indicate as project the file `CMakeLists.txt`. In the configuration window that pops up, we normally just have to proceed. The first time we do that on a new project, it can take up a few tens of seconds, checking compiler availability and various things, until the message ends with something like:

```
Configuring done
Generating done
CMake Project was parsed successfully.
```

Behind the scene, Qt Creator launched the program `CMake` and we see the information messages of this program (this appears in window number 6 “General Messages”). We can then compile the program by clicking the hammer tool on the bottom left, which is a shortcut for option “Build All” of menu “Build”. We can see in Figure 2.1 the contents of window number 4 “Compile Output” what it did:

- create the file `main.cpp.o` by compiling `main.cpp`;
- linking;
- creating the executable (“target”) `EssaiQtCreator`.

Clicking on the “Projects” button, it indicates the “Build directory”, where all output files were created. Going into that folder, we can see a file `CMakeCache.txt`, from `CMake`. This file is simply a list of variable/value pairs, we won’t modify it by hand. Among other files, we can also see our executable program. Then, clicking the green triangle on the left of the window runs the program. The result can be checked in window number 3 “Application Output”:

```
Starting /home/pascal/TEMP/Build/EssaiQtCreator...
/home/pascal/TEMP/Build/EssaiQtCreator exited with code 0
```

The program launched and exited with value 0.

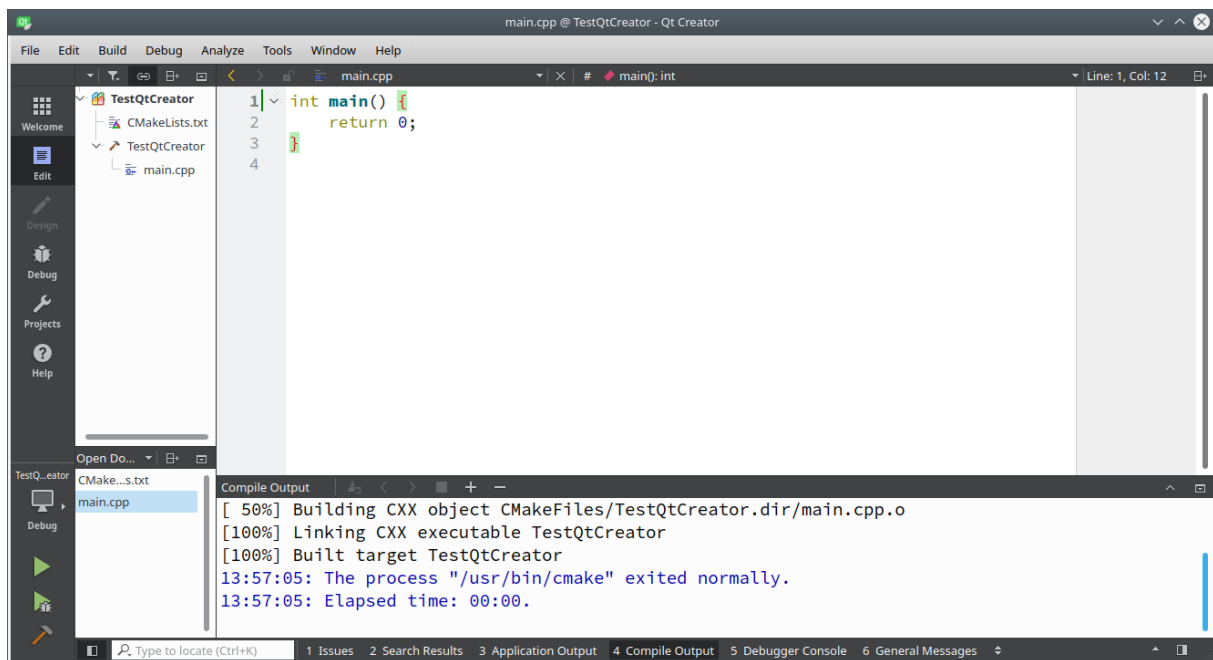


Figure 2.1: Qt Creator after compilation of the test program

2.5.2 A bit more...

In the practical session, we also use `cout` meaning “character output” and allowing writing in the terminal (`endl` means end of line). We could also test `cin`, which reads what the user inputs:

```
int i=2, j;
cout << "i=" << i << endl;
cout << "Input_an_integer:_";
cin >> j;
cout << "The_double_of_" << j << "_is_" << 2*j << endl;
```

It is to be noticed that the integrated terminal of Qt Creator (window number 3) does not handle input. Therefore, we need an external terminal, which can be selected by going into “Projects”, section “Run” and clicking radio button “Run in terminal”.

At last, our first practical session contains a condition command `if - else`.

2.5.3 The debugger

The green triangle with a small bug on it allows following the program with the debugger. It offers buttons to set breakpoints, execute up to next instruction, etc.

2.5.4 Practical Session

You are now ready to practice with the session of Appendix A.1. If practice is essential, remembering something of it is mandatory! You will also find how to install the needed tools on your computer (link <http://imagine.enpc.fr/~monasse/Imagine++> at the end).







Useful keys:			
F5	=		= Debug/Continue
F10	=		= Step over
F11	=		= Step inside
Ctrl+A, Ctrl+I (Qt Creator)	=		Indent the whole file

Figure 2.2: To be remembered from Practical Session 1

We know enough to learn a bit of C++...

We begin now to build our “reference card” that can help us while programming. We will complete it after each chapter with the new notions, which are highlighted in red. The final card is in Appendix D.

Reference Card (1/1)		
Input/Output <ul style="list-style-type: none"> • <code>#include <iostream></code> • <code>using namespace std;</code> • ... • <code>cout <<"I="<<i<<endl;</code> • <code>cin >> i >> j;</code> 	<ul style="list-style-type: none"> • Debug: F5  • Step over: F10  • Step inside: F11  • Indent: Ctrl+A, Ctrl+I 	<ul style="list-style-type: none"> • Errors/warnings: click. • Indent! • Fix warnings. • Use the debugger.
Keyboard	Advice	

Chapter 3

First programs

Ready to experiment along the way with our programming environment, it is time to learn the basics of C++. We begin programming without method... then we will add a minimum of organization by using functions.

—

Classically a programming manual is organized in a logical way with respect to the language, with different successive points: expressions, functions, variables, instructions, etc. The result is difficult to digest because we need exhaustivity on each point. We will rather try to see things as they appear when one learns: progressively and everything at once!¹ For instance, it is not in the current chapter we will see how functions memorize their variables on the “stack”.

3.1 Everything in `main()`!

Without further ado, we can put everything in `main()`. That is how a beginner programs.²

It is already an important step to program *on the road*, putting everything in function `main()`. The important point is to have a working program!

3.1.1 Variables

Types

Variables are *memories* where values (or data) are stored. A piece of data cannot be stored anyway, there needs to decide each time the *space in memory* (number of bytes) and the *format*, that is, the way the bytes are used to represent the value stored by the variable. We already met `int` that are stored with 4 bytes (32 bits) and so can take $2^{32} = 4\,294\,967\,296$ possible values.³ It is understood that `int` stores relative integers,⁴

¹The drawback of this presentation is that this manual, if made to be read from beginning to end, is less adapted to serve as reference.

²So as many students when the instructor does not check!

³We have already seen that this simple idea can give two ways to use the bytes: *big-endian* or *little-endian*.

⁴**Dear secondary school pupils:** that is, 0, 1, 2, ... but also -1, -2, -3, ...

with as many negative as positive numbers⁵ therefore, in the case of 32 bits,⁶ from $-2\,147\,483\,648$ to $2\,147\,483\,647$.⁷

Saying that a variable is `int`, it is precisizing its **type**. Some languages do not have types or guess the types of variables. For example in Python, the type is inferred from the values put in the variable, and can even change during the program execution. In C++, no such liberty, variables have a type from the start and cannot change it. We will see the compiler is rather strict in the usage of types and checks everything.

Precising a type fixes exactly the memory space and the format of a variable. The compiler, if it can make good usage of the information to detect errors, needs that before all to translate the code into machine language.

Definition, Assignment, Initialization, Constants

Before seeing other types, let us look at the syntax on an example:

```

1   int i; // Definition
2   i=2;  // Assignment
3   cout << i << " ";
4   int j;
5   j=i;
6   i=1;  // Modify i, not j!
7   cout << i << " " << j << " ";
8   int k,l,m; // Multiple definitions
9   k=l=3;    // Multiple assignments
10  m=4;
11  cout << k << " " << l << " " << m << " ";
12  int n=5,o=n,p=INT_MAX; // Initializations
13  cout << n << " " << o << " " << p << endl;
14  int q=r=4; // Wrong!
15  const int s=12;
16  s=13; // Wrong!
```

In this piece of program:

- Lines 1 and 2 **define** a variable called `i`⁸ of type `int` then **assigns** value 2 to this variable. The binary representation of 2 is thus stored in memory where the compiler decided to place `i`. What follows the “double slash” (`//`) is a **remark**: the compiler ignores all the rest of the line, it is only intended to help readability for the programmer.
- Line 3 displays the value of `i` then a blank space (without going to next line)

⁵up to one!

⁶Actually, the exact size of `int` is not prescribed by the C++ norm, it is a compiler choice, though most would use 32 bits. Anyway, constants `INT_MIN` and `INT_MAX` allow to retrieve the extreme values of the type.

⁷Though the norm does not even mandates that `INT_MIN=-INT_MAX-1`, but only that taking twice the negative of a positive value yields the original value.

⁸The *name* of a variable is also called *identifier*. Error messages from the compiler with rather use this vocabulary!

- Lines 4, 5 and 6 define an `int` named `j`, copy its value of `i`, which is 2, in `j`, then put 1 in `i`. Note that `i` and `j` are two distinct variables: `i` becomes 1 but `j` stays at 2!
- Line 8 shows how to define simultaneously several variables of same type.
- Line 9 teaches us that we can assign the same value to several variables at once.
- At line 12, variables are defined and assigned at once. Actually, we call these variables **initialized**: they take an initial value at the same time they are defined. Note that, for reasons of efficiency, **variables are not initialized by default**: as long as no value was put in them, **they are worth whatever**, an arbitrary value!⁹
- Beware still, it is useless and not permitted to try simultaneous initialization. Line 14 is a mistake that the compiler will not allow.
- At last, one can add `const` in front of the variable type: it becomes then a constant and we cannot modify its value later. Line 15 defines such a constant and line 16 is a compiler error.

In summary, after removal of lines 14 and 16, this (amazing!) piece of code displays:¹⁰

```
2 1 2 3 3 4 5 5 2147483647
```

Variable names (actually, all identifiers, including function names) are composed only of characters a to z (and capitals), digits and underscore `_`, but cannot begin with a digit. Do not use accented characters, it is a portability issue.

Scope

In the previous example, variables were defined when needed. This is not such an evidence. For instance, in C all variables needed to be defined before the first instruction of the function. In C++, we have more leeway, but beware:

the variables do exist (and cannot be used before) only from the line they are defined. They have a limited lifespan and die as soon as we exit the block limited by curly brackets they belong to.^a. This is what is called the scope of a variable.

^aIt is a bit more complex for *global variables*. We will see that also...

Therefore, taking a bit of advance on test syntax, the following program presents scope errors at lines 2 and 8:

```
1   int i;
2   i=j; // Error: j not yet defined!
3   int j=2;
4   if (j>1) {
```

⁹It is an error to use the value of a variable before its first assignment. The compiler will not warn about that in general.

¹⁰Generally, cf. preceding remark about `INT_MAX`

```

5     int k=3;
6     j=k;
7     }
8     i=k; // Error: k does not exist anymore

```

Other types

We will see the different types along the way. Here are the most usual:

```

int i=3;           // Relative integer
double x=12.3;    // Real number (double precision)
char c='A';       // Character
string s="hop";   // String of characters
bool t=true;     // Boolean (true or false)

```

Real numbers are generally approached by variables of type `double` (“double precision”, 8 bytes usually). Characters are represented by a single byte, the correspondence character/value being defined by the ASCII table (65 pour A, 66 pour B, etc.), that we don’t need to learn since the syntax `'A'` with single quotes is translated to value 65 by the compiler, etc. Double quotes are reserved for “strings” of characters.¹¹ At last, booleans are variables that can take only two values, `true` and `false`.

Without being exhaustive, here are some additional types:

```

float y=1.2f;     // Real number (single precision)
unsigned int j=4; // Natural integer
signed char d=-128; // One-byte integer
unsigned char d=254; // One-byte natural integer
complex<double> z(2,3); // Complex number

```

where one finds:

- the `float`, real numbers less precise than `double` but faster, on 4 bytes (curious ones can explore the language documentation and see that a `float` has a maximum value `FLT_MAX` (about $3.4e+38$ ¹²) and that their smallest strictly positive value is `FLT_MIN` (about $1.2e-38$), in the same manner as `double` have constants `DBL_MAX` and `DBL_MIN` of value roughly $1.8e+308$ and $2.2e-308$);
- the `unsigned int`, positive integers used to go further than `int` in positive values (from 0 to `UINT_MAX`, generally 4294967295);
- the `unsigned char`, from 0 to 255;
- the `signed char`, from -128 to 127;
- and at last complex numbers.¹³

¹¹Beware, usage of string requires `#include<string>` at the beginning of the program.

¹²Read 3.4×10^{38} . **Secondary school pupils:** 10^{38} is 1 followed by 38 zeros, 10^{-38} or $1e-38$ is 0.000...01 with 37 zeros before the 1. $3.4e+38$ is 34 followed by 37 zeros (38 digits after 3) and $1.2e-38$ is 0.00...012 still with 37 zeros between the decimal separator and 1 (1 is at place 38).

¹³It is too early to learn the syntax of this definition but it is good to know that complex are readily available in C++. **Secondary school pupils:** do not panic! You will learn complex numbers when you grow up, we won’t use them in this manual any more.

3.1.2 Tests

Simple Tests

Tests are used to executed one instruction or another based on the value of variables. The condition is between parentheses and must be a `bool` value. Comparison operators are equality `==` and difference `!=`, and inequalities `>`, `>=`, `<` and `<=`. Several conditions can be combined by "and" `&&`, "or" `||`. Negation of a boolean is `!` (before the boolean). Depending on the value `true` or `false` of the condition, one instruction or another is executed. Many times, we need several instructions, they can be grouped in a block by curly braces. Look at the example:

```

if(i==0) // i is null?
    cout << "i_is_null" << endl;
...
if(i>2) // i greater than 2?
    j=3;
else
    j=5; // If we are here, then i<=2
...
// More complex!
if(i!=3 || (j==2 && k!=3) || !(i>j && i>k)) {
    // Here, i is not 3 or
    // j is 2 and k not 3 or
    // i is not greater than both j and k
    cout << "First_instruction" << endl;
    cout << "Second_instruction" << endl;
}

```

Notice that the condition of `if` can be a `bool` variable, but most of the time it is directly the test. But if we want to expand it, no problem:

```

bool t= ((i==3)|| (j==4));
if(t)
    k=5;

```

Finally, some very important thing: the test operator `==` should be distinguished from assignment `=`.¹⁴ It could be the most frequent error among beginners. Fortunately, the compiler should emit a warning.¹⁵

Warning: Use `if (i==3)` and *not* `if (i=3)`!

The "switch"

We may have to do one or two things depending on the values of a variable. For clarity purposes, we then use the `switch` statement that looks more elegant than a series of `if/else`. Each possible case for the values of the variable is introduced with `case`

¹⁴Writing `if (i=3)` puts 3 in `i` then gives 3, which is not a `bool`, but is considered as true because by convention any value except 0 is considered `true`.

¹⁵The compiler is your friend, even though it may be annoying sometimes (friends can also be)!

and **must end with `break`**.¹⁶ Several `case` can be next to each other for multiple cases. At last, the keyword `default`, which must appear last, gets used for all other cases. The following program¹⁷ reacts to pressed keys on the keyboard and uses a `switch` statement to display amazing comments!

```

1 #include <iostream>
2 using namespace std;
3 #include <conio.h> // Avoid: non standard , curses.h under Linux
4
5 int main()
6 {
7     bool end=false;
8     do {
9         char c=_getch(); //Avoid: non standard , getch() under Linux
10        switch (c) {
11            case 'a':
12                cout << "You_pressed_'a '! " << endl;
13                break;
14            case 'f':
15                cout << "You_pressed_'f'._Bye!" << endl;
16                end=true;
17                break;
18            case 'e':
19            case 'i':
20            case 'o':
21            case 'u':
22            case 'y':
23                cout << "You_pressed_another_vowel!" << endl;
24                break;
25            default:
26                cout << " Still_some_other_key!" << endl;
27                break;
28        }
29    } while (!end);
30    return 0;
31 }

```

Except the `do/while` loop, which we will soon see, you should have understood everything and you should realize it is equivalent to:

```

if (c=='a')
    cout << "You_pressed_'a '! " << endl;
else if (c=='f') {
    cout << "You_pressed_'f'._Bye!" << endl;
}

```

¹⁶It is a frequent and damaging error to forget the `break`. Without it, the program executes also the instructions of the next case

¹⁷Beware, a `cin >> c`, instruction we will see later, reads the keyboard but does not terminates at each pressed key: it waits the key `Enter`! Reading just a single keystroke is not standard and is not so used in our world of graphical interfaces. Under Windows, we would use the function `_getch()` after `#include <conio.h>` (cf. lines 3 and 9) and under Linux `getch()` after `#include <curses.h>`.

```

    end=true ;
} else if (c=='e' || c=='i' || c=='o' || c=='u' || c=='y')
    cout << "You_pressed_another_vowel!" << endl;
else
    cout << "Still_some_other_key!" << endl;

```

Before all, remember the main error source with `switch`:

In a `switch`, do not forget the `break`!

You could have noticed a line 2 somewhat cryptic. A `namespace` is like a prefix for certain things. The prefix of standard objects is `std`. Hence `cout` and `endl`, provided by `#include <iostream>`, have full name `std::cout` and `std::endl`. Line 2 allows to omit the prefix in the rest of the file.

3.1.3 Loops

It is difficult to do anything interesting without the possibility to execute several times the same instruction (with different data!). That is the role of loops. The most powerful is `for()`, but it is not the simplest to understand. Let's begin with `do ... while`, that loops until a test is valid. The next program waits from the user an integer between 1 and 10 and asks again until the input is correct:

```

1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     int i;
7     do { // Loop begins
8         cout << "A_number_between_1_and_10,_please:_";
9         cin >> i;
10    } while (i<1 || i>10); // Go back to beginning if
11                        // the test is true
12    cout << "Thanks!_Your_number_is_" << i << endl;
13    return 0;
14 }

```

Note line 9 that puts in `i` a number input by the user. The variable `cin` is the counterpart ("console in") of `cout`.

Next comes the `while` loop (without `do`) that performs a check before entering the loop. The next program displays digits from 1 to 100:

```

int i=1;
while(i<=100) {
    cout << i << endl;
    i=i+1;
}

```

Finally, we have a special loop, the most frequent: `for()` that executes an instruction before starting, does a test at each iteration so as `while`, and executes an instruction at

the end of each iteration. Initial instruction, test and final instruction are separated by a semicolon `;`, which yields the following program, equivalent to the preceding one:

```
int i;
for (i=1; i<=100; i=i+1) {
    cout << i << endl;
}
```

Most frequently, the `for()` is used as in the previous example to increase an index and span an interval. We will find often:

```
for (int i=1; i<=100; i++)
    cout << i << endl;
```

when we know that:

- We can define a variable in the left part of `for()`. Beware, this variable admits as `for()` the scope limit: the variable does not exist after the `for()`.¹⁸
- `i++` is an abbreviation of `i=i+1`. The `++` can also be written as prefix `++i`. The same is valid for the decrement operator `--`.
- Since there is a single instruction in the loop, curly braces are not mandatory.

We can also use the comma `,` to include several instructions¹⁹ in the right part of `for`. The following program starts with $i = 1$ and $j = 100$, and increases i by 2 while decreasing j by 3 at each iteration until their values cross.²⁰

```
for (int i=1, j=100; j>i; i=i+2, j=j-3)
    cout << i << " " << j << endl;
```

Note also we could abbreviate `i=i+2` in `i+=2` and `j=j-3` in `j-=3`.

3.1.4 Recreations

We are now able to write many programs. For example, play the game of the secret price. The program chooses a price and the user has to guess it:

```
1 #include <iostream>
2 #include <cstdlib>
3 using namespace std;
4
5 int main()
6 {
7     int n=rand()%100; // Number between 0 and 99
8     int i;
9     do {
10        cout << "Your price: ";
```

¹⁸Some very old C++ compilers did not allow defining a variable in `for()`. Some more recent ones (but still old!) allowed it but the variable survived the `for()`! Forget that, the standard says that it should be as we explained it.

¹⁹For the curious ones: it is not out of the ordinary, as several instructions separated by a comma count in C++ as a single instruction.

²⁰The last printed line will be `39 43`, meaning that the loop exits when $i = 41$ and $j = 40$.

```

11     cin >> i;
12     if(i>n)
13         cout << "Try_less" << endl;
14     else if(i<n)
15         cout << "Try_more" << endl;
16     else
17         cout << "Exact!" << endl;
18 } while(i!=n);
19 return 0;
20 }

```

Only line 7 requires more explanation:

- The function `rand()` returns a random number between 0 and a compiler-specific constant `RAND_MAX`. We need to add `#include <cstdlib>` to use it.
- `%` is the modulo function.²¹

It is obviously more interesting when the program has to guess. It will work by *dichotomy* to find in a minimum of trials:

```

1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     cout << "Choose_a_number_between_1_and_100" << endl;
7     cout << "Answer_my_trials_by_+,_-_or_=" << endl;
8     int a=1,b=100; // Extreme values
9     bool found=false;
10    do {
11        int c=(a+b)/2; // Propose middle
12        cout << "Is_it_" << c << "?:_";
13        char r;
14        do
15            cin >> r;
16        while(r!='=' && r!='+' && r!='-');
17        if(r=='=')
18            found=true;
19        else if(r=='-')
20            b=c-1; // Too high, try between a and c-1
21        else
22            a=c+1; // Too low, try between c+1 and b
23    } while(!found && (a<=b));
24    if(found)
25        cout << "What_talent_do_I_have!" << endl;
26    else

```

²¹**Secondary school pupils:** counting “modulo N”, it is going back to 0 when we reach N. Modulo 4, it gives: 0,1,2,3,0,1,2,3,0,... For example 12%10 is 2 and 11%3 also! Here, modulo 100 ensures we have a result between 0 and 99.

```

27     cout << "You_cheated!" << endl;
28     return 0;
29 }

```

One can also complete the supplementary program of the practical of Appendix A.1. It is a ball bounding in a square. (See Appendix B for graphics instructions...)

```

1 #include <Imagine/Graphics.h>
2 using namespace Imagine;
3
4 int main()
5 {
6     int w=300, h=210;
7     openWindow(w,h); // Graphic window
8     int i=0,j=0;     // Position
9     int di=2,dj=3;  // Speed
10    while(true) {
11        fillRect(i,j,4,4,RED); // Draw ball
12        milliSleep(10); // Wait a little bit...
13        if(i+di>w || i+di<0) {
14            di=-di; // Horizontal rebound if we exit
15        }
16        int ni=i+di; // New position
17        if(j+dj>h || j+dj<0) {
18            dj=-dj; // Vertical rebound if we exit
19        }
20        int nj=j+dj;
21        fillRect(i,j,4,4,WHITE); // Erase
22        i=ni; // Move to new position
23        j=nj;
24    }
25    endGraphics();
26    return 0;
27 }

```

Notice the `endGraphics()` whose role is to wait a mouse click from the user before terminating the program, so that the window remains visible as long as necessary. This function is not standard and is in the namespace `Imagine`. Line 2 allows calling it without providing its full name `Imagine::endGraphics()`. Other functions called in this little program (`openWindow`, `fillRect` and `milliSleep`) come also from `Imagine`.

3.2 Functions

When everything is in `main()` one realizes fast that one has to copy/paste often pieces of the program. If lines of code begin to look alike, it is likely an opportunity to build functions. It is done for clarity, but also to preserve from typing too much on the keyboard!

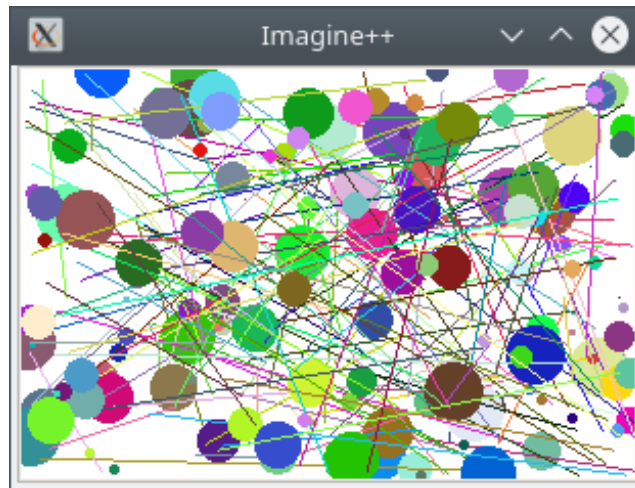


Figure 3.1: Random lines and circles...

One has to group identical pieces of code in functions:

- to get a clearer code...
- and to avoid getting tired!

It is important to understand when to make a function and not to decorate a program in little pieces with any logic.^a

^aor just to please the instructor. Cutting randomly the program code is the best way to dissuade doing it again.

Actually, the possibility of reusing previous work is the thread of good programming. For the moment, we use functions only to reuse what we typed a few lines before. Later, we will want to reuse what was written in other programs, maybe long before, or in other people's software, and we will see how to do it.

Let's consider the following program that draws random lines and circles as shown in Figure 3.1:

```

1 #include <Imagine/Graphics.h>
2 using namespace Imagine;
3 #include <cstdlib>
4 using namespace std;
5
6 int main() {
7     openWindow(300,200);
8     for (int i=0;i<150;i++) {
9         int x1=rand()%300; // Initial point
10        int y1=rand()%200;
11        int x2=rand()%300; // Final point
12        int y2=rand()%200;
13        Color c=Color(rand()%256,rand()%256,rand()%256); //RGB
14        drawLine(x1,y1,x2,y2,c); // Line drawing
15        int xc=rand()%300; // Circle center

```

```

16     int yc=rand()%200;
17     int rc=rand()%10; // Radius
18     Color cc=Color(rand()%256,rand()%256,rand()%256); //RGB
19     fillCircle(xc,yc,rc,cc); // Circle
20 }
21 endGraphics();
22 return 0;
23 }

```

The first striking thing²² is the repeated call to rand() and to modulo to draw a random number. We will often have to draw a number in a given interval and it is natural to do it with a function. By the way, we fix another annoying thing: integers 300 and 200 come often. If we wanted to change the dimensions, we would have to replace 300 and 200 in numerous places.

From the beginning, one has to spot the constant parameters of the program used several times and place them in constants. We save a lot of time^a when one wishes to change them later.

^aAgain the rule of least effort... When we begin to do too much copy/paste or find/replace with the editor, it's a bad sign!

In short, our program becomes:

```

// Number between 0 and n-1
int chance(int n) {
    return rand()%n;
}

int main() {
    const int w=300,h=200;
    openWindow(w,h);
    for(int i=0;i<150;i++) {
        int x1=chance(w),y1=chance(h); // Initial point
        int x2=chance(w),y2=chance(h); // Final point
        Color c=Color(chance(256),chance(256),chance(256));
        drawLine(x1,y1,x2,y2,c); // Line drawing
        int xc=chance(w),yc=chance(h); // Circle center
        int rc=chance(w/20); // Radius
        Color cc=Color(chance(256),chance(256),chance(256));
        fillCircle(xc,yc,rc,cc); // Circle drawing
    }
    endGraphics();
    return 0;
}

```

One could think that typing chance(w) is as long as typing rand()%w and that our function is useless. This is not false, but in practice one could then forget the existence of rand() and how to do a modulo. It is even better than that: we become independent

²²besides obviously the syntax “object” of variables of type Color for which we indulge one Color(r,g,b) well in advance on what we will learn...

of these functions and if you want to draw random numbers with another function,²³ we would only have to change once function chance(). It is again an important rule.

We should also make a function when we want to separate and factorize work. It is then easier^a to modify the function than all the lines it replaced!

^aLeast effort, always!

3.2.1 Return

We defined without explanation a function chance() that takes a parameter n of type `int` and that returns a result, of type `int` also. There is not much more to know besides that:

1. A function may not return anything. Its return “type” is then `void` and there is no `return` at the end (though writing `return;` is admitted). For instance:

```
void say_hello_to_the_lady(string name_of_lady) {
    cout << "Hello ,_Mrs_" << name_of_lady << "!" << endl;
}
...
say_hello_to_the_lady("Germaine");
say_hello_to_the_lady("Fitzgerald");
```

2. A function may have several `return` instructions. This allows exiting the function when we want, what is much clearer and closer to our way of thinking:

```
int sign_with_single_return(double x) {
    int s;
    if(x==0)
        s=0;
    else if(x<0)
        s=-1;
    else
        s=1;
    return s;
}

int sign_simpler(double x) {
    if(x<0)
        return -1;
    if(x>0) // Note no else here, useless
        return 1;
    return 0;
}
```

3. For a function `void`, one can use `return` without anything:

²³What for? In our case the function rand() is enough for standard applications but may not be “random enough” for math applications. At last, we forgot to initialize the random seed. If you please, we will see another time what this means and how to do it.

```

void phone_with_single_return(string name) {
    if(I_have_the_phone) {
        if(my_phone_works) {
            if(in_directory(name)) {
                int number=phone_number(name);
                dial(number);
                if(answering) {
                    talk();
                    hang_up();
                }
            }
        }
    }
}

void phone_simpler(string name) {
    if(!I_have_the_phone ||
        !my_phone_works ||
        !in_directory(name))
        return;
    int number=phone_number(name);
    dial(number);
    if(!answering)
        return;
    talk();
    hang_up();
}

```

3.2.2 Parameters

We have only seen functions with a single parameter. Here is how to input several parameters or none:

```

// Number between a and b
int chance2(int a, int b) {
    return a+(rand()%(b-a+1));
}

// Number between 0 and 1
double chance3() {
    return rand()/double(RAND_MAX);
}
...
int a=chance2(1,10);
double x=chance3();
...

```

Beware to use `x=chance3()` and not simply `x=chance3` to call a parameter-less function. This simple program is also an opportunity to talk about a frequent error: the division of two integers yields an integer! Writing `double x=1/3;` is a mistake since C++ first

computes $1/3$ with integers, which yields 0, then converts 0 to `double` to put it in `x`.²⁴ It does not know when computing $1/3$ that the result will be put in a `double`! One has to ensure that 1 and 3 are in `double` and write `double x=1.0/3.0`;; `double x=1.0/3`; or `double x=1/3.0`; If, as in our case, we deal with variables of type `int`, one has to convert one into `double` with syntax `double (...)` that we will see later.

1. Function without parameter: `x=hop()`; not `x=hop`;

2. Integer division:

- `double x=1.0/3`; not `double x=1/3`;
- `double x=double(i)/j`,^a not `double x=i/j`, not `double x=double(i/j)`^b

^a`double x=double(i)/double(j)`; is fine also though more verbose.

^bThis conversion into `double` arrives too late, division of integers was already performed!

3.2.3 Reference passing

When a function modifies the value of one of its parameters, and if that parameter was a variable in the calling function, then this variable is *not* modified. The following program fails its goal:

```
void triple(int x) {
    x=x*3;
}
...
int a=2;
triple(a);
cout << a << endl;
```

It display 2 and not 6. Actually, the parameter `x` of function `triple` is first 2, then 6. But its turn to 6 does not modify `a`. We will see later that `x` is memorized somewhere else than `a`, which explains everything! It is the value of `a` that is passed to function `triple()` and not the variable `a` itself! We talk about **passing by value**. One can still make sure that the function modifies the parameter. We talk about **passage by reference** (or *by variable*). We just need to append `&` after the parameter type:

```
void triple(int& x) {
    x=x*3;
}
```

Generally, the following example is used to illustrate the need of references:

```
void swap1(int x, int y) {
    int t=x;
    x=y;
    y=t;
}
```

²⁴Version 2 of Python used to do the same. It was such a frequent error (and Python wants to be friendly to beginners), that it was changed in Python 3. Future standards of C++ will never change that for two reasons: (1) it is more logical not to change type between the operands and the result; (2) for backward compatibility.

```

void swap2(int& x, int& y) {
    int t=x;
    x=y;
    y=t;
}
...
int a=2,b=3;
swap1(a,b);
cout << a << " " << b << " ";
swap2(a,b);
cout << a << " " << b << endl;
...

```

It displays 2 3 3 2, swap1() doing nothing. Note that at the call site, nothing distinguishes the call by value or by variable, though your code editor may be smart enough to put in slanted font the variables passed by reference and not by value.

A good way to understand passing by reference is to consider variables x and y of swap1 as variables truly independent of a and b of the caller, whereas in the call swap2, x and y of swap2 become “links” to a and b . Each time we use x in swap2, it is actually a that is used. To understand even better, go see the first exercise of Practical 2 (A.2.1).

In practice,

one uses also reference passing for a function that needs to return several things at once,

in the following manner:

```

void a_point(int& x, int& y) {
    x=...;
    y=...;
}
...
int a,b;
a_point(a,b);

```

Our random drawing program can become:

```

1 #include <Imagine/Graphics.h>
2 using namespace Imagine;
3 #include <cstdlib>
4 using namespace std;
5
6 // Number between 0 and n-1
7 int chance(int n) {
8     return rand()%n;
9 }
10
11 Color a_color() {
12     Color col=Color(chance(256), chance(256), chance(256));
13     return col;
14 }

```

```

15
16 void a_point(int w,int h,int& x,int& y){
17     x=chance(w);
18     y=chance(h);
19 }
20
21 int main() {
22     const int w=300,h=200;
23     openWindow(w,h);
24     for(int i=0; i<150; i++) {
25         int x1,y1; // Initial point
26         a_point(w,h,x1,y1);
27         int x2,y2; // Final point
28         a_point(w,h,x2,y2);
29         Color c=a_color();
30         drawLine(x1,y1,x2,y2,c); // Line drawing
31         int xc,yc; // Circle center
32         a_point(w,h,xc,yc);
33         int rc=chance(w/20); // Radius
34         Color cc=a_color();
35         fillCircle(xc,yc,rc,cc); // Circle drawing
36     }
37     endGraphics();
38     return 0;
39 }

```

With the following advice:

one can use directly the result of a function without an intermediary variable.

it even becomes

```

13     returnColor(chance(256),chance(256),chance(256));
35     fillCircle(xc,yc,rc,a_color()); // Circle

```

and variables `col` (line 12) and `cc` (line 34) can be dropped.

3.2.4 Scope, Declaration, Definition

From the beginning, we created functions by **defining** them. It is sometimes useful to know only the return type and the parameters of a function without having to know how it is programmed, that is, without having the core of the function. One reason of this need is:

So as variables, functions have a scope and are known only at lines that follow it.

Therefore the following program does not compile:

```

1 int main() {
2     f();
3     return 0;
4 }
5 void f() {
6 }

```

since at line 2, `f()` is not yet known. It is enough here to move lines 5 and 6 before `main()` so that it compiles. On the contrary, it is more complex to make compile:

```

void f() {
    g(); // Error: g() unknown
}
void g() {
    f();
}

```

since both functions need each other, and no order would solve that. The following rule saves us:

- Replacing the core of a function by a single `;` is called *declaration of the function*.
- Declaration of a function is enough for the compiler, it can “wait”^a until its *definition*.

^aActually, the compiler needs *only* declaration. The linker will have to find somewhere the function definition, or more exactly the result of the compiled code built from its definition!

The preceding program can compile with a single extra line:²⁵

```

void g(); // Declaration of g
void f() {
    g(); // OK: declared function
}
void g() { // Definition of g
    f();
}

```

3.2.5 Local and global variables

We have seen Section 3.1.1 the scope of a variable. The rule of curly braces is obviously also valid to the curly braces of the core of a function.

The variables of a function are unknown outside the function.

We talk about **local variables**. The following program is malformed:

```

void f() {
    int x;
}

```

²⁵It compiles and runs, but when either one of the functions is called, the program would be caught in an infinite loop. What will concretely happen is left as an experiment for the reader.


```

    x=3;
}
void g() {
    int y;
    y=x; // Error: x unknown
}

```

If really two functions use common variables, one can “exit” them from functions. They become then **global variables**, here is an example:

```

1 int z; // global
2 void f() {
3     int x; // local
4     ...
5     if(x<z)
6         ...
7 }
8 void g() {
9     int y; // local
10    ...
11    z=y;
12    ...
13 }

```

The use of global variables is tolerated and sometimes justified.²⁶ But it is a lazy solution beginners abuse of and we have to fight this tendency from the start:

Global variables must be avoided because:

- they allow abusive communication between functions, source of bugs^a.
- the functions that use them are often harder to reuse in other contexts.^b

In general, they are a clue of a badly organized solution.

^aThat is why non-constant variables are not tolerated from beginners. See the previous program where g() talked to f() through z.

^bIn particular, they interfere with parallelism.

3.2.6 Overload

It is sometimes useful to have a function accepting different types of arguments passed. We can use *overload* for this:

Two functions that have different parameter numbers or types can have the same name.^a Warning: two functions with just different return types cannot have the same name.^b

^aSince then the way they are called will allow the compiler to know which is meant.

^bSince the compiler will not be able to differentiate their call.

²⁶In that case, the usage of the variable in the functions may be written `::z` to emphasize that `z` comes “from outside”.

Our functions “chance” from above could be written:

```

1 // Number between 0 and n-1
2 int chance(int n) {
3     return rand()%n;
4 }
5 // Number between a and b
6 int chance(int a,int b) {
7     return a+(rand()%(b-a+1));
8 }
9 // Number between 0 and 1
10 double chance() {
11     return rand()/double(RAND_MAX);
12 }
13 ...
14 int i=chance(3); // in 0...2
15 int j=chance(2,4) // in 2...4
16 double k=chance(); // in 0...1

```

The functions can do completely different and unrelated things, they do not share any code. Obviously, it is a bad idea to use overload in that case.

3.3 Practical

You know already plenty after this (dense) lesson and are ready (after a well deserved break) to try Practical 2 in Appendix A.2 in order to better understand the functions and to play a game of mini tennis (Figure 3.2).

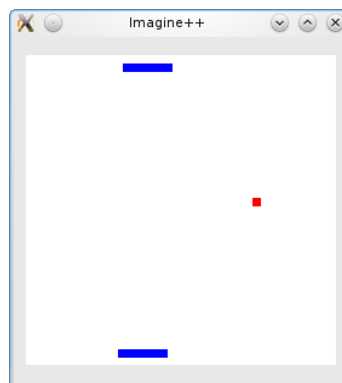





Figure 3.2: Mini tennis...

3.4 Reference Card

Reference Card (1/2)		
<p>Variables</p> <ul style="list-style-type: none"> • Definition: <pre>int i; int k,l,m;</pre> • Assignment: <pre>i=2; j=i; k=l=3;</pre> • Initialization: <pre>int n=5,o=n;</pre> • Constants: <pre>const int s=12;</pre> • Scope: <pre>int i; // i=j; forbidden! int j=2; i=j; // OK! if (j>1) { int k=3; j=k; // OK! } //i=k; forbidden!</pre> • Types: <pre>int i=3; double x=12.3; char c='A'; string s="hop"; bool t=true; float y=1.2f; unsigned int j=4; signed char d=-128; unsigned char d=25; complex<double> z(2,3);</pre> • Global variables: <pre>int n; const int m=12; void f() { n=10; // OK int i=m; // OK ...</pre> • Conversion: <pre>int i=int(x),j; float x=float(i)/j;</pre> <p>Tests</p> <ul style="list-style-type: none"> • Comparison: <pre>== != < > <= >=</pre> 	<ul style="list-style-type: none"> • Negation: ! • Combinations: && • <pre>if (i==0) j=1;</pre> • <pre>if (i==0) j=1; else j=2;</pre> • <pre>if (i==0) { j=1; k=2; }</pre> • <pre>bool t=(i==0); if (t) j=1;</pre> • <pre>switch (i) { case 1: ...; ...; break; case 2: case 3: ...; break; default: ...; }</pre> <hr/> <p>Loops</p> <ul style="list-style-type: none"> • <pre>do { ... } while(!ok);</pre> • <pre>int i=1; while(i<=100) { ... i=i+1; }</pre> • <pre>for(int i=1;i<=10;i++) ...</pre> • <pre>for(int i=1,j=10;j>i; i=i+2,j=j-3) ...</pre> <hr/> <p>Functions</p> <ul style="list-style-type: none"> • Definition: <pre>int plus(int a,int b){ int c=a+b; return c; } void display(int a) { cout << a << endl; }</pre> 	<ul style="list-style-type: none"> • Declaration: <pre>int plus(int a,int b);</pre> • Return: <pre>int sign(double x) { if (x<0) return -1; if (x>0) return 1; return 0; } void display(int x, int y) { if (x<0 y<0) return; if (x>=w y>=h) return; DrawPoint(x,y,RED); }</pre> • Call: <pre>int f(int a) { ... } int g() { ... } ... int i=f(2),j=g();</pre> • References: <pre>void swap(int& a, int& b){ int tmp=a; a=b;b=tmp; } ... int x=3,y=2; swap(x,y);</pre> • Overload: <pre>int chance(int n); int chance(int a, int b); double chance();</pre> <hr/> <p>Miscellaneous</p> <ul style="list-style-type: none"> • <pre>i++; i--; i-=2; j+=3;</pre> • <pre>j=i%n; // Modulo</pre> • <pre>#include <cstdlib> ... i=rand()%n; x=rand()/ double(RAND_MAX);</pre>

Reference Card (2/2)		
<p>Input/Output</p> <ul style="list-style-type: none"> • <code>#include <iostream></code> <code>using namespace std;</code> <code>...</code> <code>cout <<"I="<<i<<endl;</code> <code>cin >> i >> j;</code> <p>Frequent errors</p> <ul style="list-style-type: none"> • No definition of function inside a function! • <code>int q=r=4; // NO!</code> • <code>if (i=2) // NO!</code> <code>if i==2 // NO!</code> <code>if (i==2) then // NO!</code> 	<ul style="list-style-type: none"> • <code>for(int i=0,i<100,i++)</code> <code>// NO!</code> • <code>int f() {...}</code> <code>int i=f; // NO!</code> • <code>double x=1/3; // NO!</code> <code>int i,j;</code> <code>x=i/j; // NO!</code> <code>x=double(i/j); //NO!</code> <hr/> <p>Imagine++</p> <ul style="list-style-type: none"> • See documentation... <hr/> <p>Keyboard</p> <ul style="list-style-type: none"> • Debug: F5  	<ul style="list-style-type: none"> • Step over: F10  • Step inside: F11  • Indent: Ctrl+A, Ctrl+I <hr/> <p>Advice</p> <ul style="list-style-type: none"> • Errors/warnings: click. • Indent! • Fix warnings. • Use the debugger. • Write functions.

Chapter 4

Arrays

While we continue using functions to assimilate them, we are going to add the **arrays** that we would otherwise miss quickly. We will proceed slowly and will see only arrays of one dimension with fixed size for a start. We will see in another chapter variable length arrays and questions of memory (“stack” and “heap”).

—

4.1 First arrays

So as we needed at once loops to do similar things several times, it is useful to have the possibility to do similar things on different variables. Hence arrays... The following program

```
int x1,y1,u1,v1; // Ball 1
int x2,y2,u2,v2; // Ball 2
int x3,y3,u3,v3; // Ball 3
int x4,y4,u4,v4; // Ball 4
```

...

```
moveBall(x1,y1,u1,v1);
moveBall(x2,y2,u2,v2);
moveBall(x3,y3,u3,v3);
moveBall(x4,y4,u4,v4);
```

will be replaced with

```
int x[4],y[4],u[4],v[4]; // Balls
```

...

```
for(int i=0; i<4; i++)
    moveBall(x[i],y[i],u[i],v[i]);
```

in which `int x[4]` defines an *array* of 4 variables of type `int`: `x[0]`, `x[1]`, `x[2]` and `x[3]`. In practice, the compiler reserves in memory a place to store the 4 variables and handle things to that the request `x[i]` designates the right variable.

Another example to understand better, that adds some `double` variables by pairs while memorizing the results:

```
double x[100],y[100],z[100];
```

...

```

... // Here, x[i] and y[i] take values
...
for (int i=0; i<100; i++)
    z[i]=x[i]+y[i];
...
... // Here, we use z[i]
...

```

It would have been nice to be able to write `z=x+y`; but it would not compile.¹ As we will see, with arrays, we have to do everything by hand (with loops)!

There are two essential things to memorize

1. First:

Indices of an array `t` of size `n` go from 0 to `n-1`. Any access to `t[n]` (reading or writing) can provoke a serious error when running the program. THAT IS ONE OF THE MOST COMMON ERRORS IN C++. Either it would read or write in a place reserved for another variable,^a or we would access illegal memory and the program “crashes”.^b

^aIn the example above, if we replaced the loop so that `i` went from 1 to 100, `x[100]` would certainly fetch `y[0]` instead. In the same manner, `z[100]` may fetch the variable `i` of the loop, which would risk producing strange things later, `i` taking an unexpected value!

^bAbove, `z[i]` with a wild `i` would write outside the memory reserved to data, which would stop the program more or less delicately!

In the last example, we use `x[0]` to `x[99]`. It is customary to have the loop with `i<100` as test, rather than `i<=99`. But beware not to mix things and write `i<=100`!

2. Then:

an array must have a fixed size known at compilation time. This size can be directly a number or a constant variable, but not a variable.

Even when we think the compiler should know the size, it plays dumb and accepts only constants:²

```

1 double x[10],y[4],z[5]; // OK
2 const int n=5;
3 int i[n],j[2*n],k[n+1]; // OK
4 int n1; // n1 has no value
5 int t1[n1]; // hence ERROR
6 int n2;
7 cin >> n2; // n2 takes a value, but known
8 // only at runtime
9 int t2[n2]; // hence ERROR
10 int n3;

```

¹But we will find an elegant solution to this problem in a later chapter.

²Actually, your compiler *may* accept the code. Doing so, it does not respect the standard and you take risks. Do not! The solution will come in a later chapter.

```

11 n3=5;           // n3 takes a value, known,
12                // but... not constant
13 int t3[n3];    // hence ERROR (YES!)

```

Keeping in mind both points, we can easily use arrays. Beware though:

do not use an array where it is a waste of memory, notably when translating a mathematical formula.

Let me explain. If you want to compute $s = \sum_{i=1}^{100} f(i)$ with given f ,³ such as $f(i) = 3i + 4$, do not write, as we see sometimes:

```

1 double f[100];
2 for(int i=1; i<=100; i++)
3     f[i]=3*i+4;
4 double s;
5 for(int i=1; i<=100; i++)
6     s=s+f[i];

```

neither, after having fixed bugs:

```

1 double f[100];           // Store f(i) in f[i-1]
2 for(int i=1; i<=100; i++)
3     f[i-1]=3*i+4;       // Beware indices
4 double s=0;             // Better like that
5 for(int i=1; i<=100; i++)
6     s=s+f[i-1];

```

but rather directly without array:

```

1 double s=0;
2 for (int i=1;i<=100;i++)
3     s=s+(3*i+4); // Or better: s+=3*i+4

```

which saves for the machine an array (that is, memory), and for you bugs (hence your nerves!). Note that we used the recurrence relation:

$$s_k := \sum_{i=1}^k f(i) = s_{k-1} + f(k)$$

to compute s_{100} . Since to compute s_k we need only to remember s_{k-1} , we can use a single variable s that we update.

4.2 Initialization

As for a variable, an array can be initialized when defined:

```

int t[4]={1,2,3,4};
string s[2]={"hip", "hop"};

```

³For secondary school pupils: meaning $s = f(1) + f(2) + \dots + f(100)$.

Beware, the syntax for initialization does not work for assignment:⁴

```
int t[2];
t={1,2}; // Error!
```

4.3 Specifics of arrays

Arrays are special variables. They do not behave as other variables...⁵

4.3.1 Arrays and Functions

So as variables, we need to pass arrays as arguments to functions. The syntax is simple:

```
void display(int s[4]) {
    for (int i=0;i<4;i++)
        cout << s[i] << ' '; // Print all on same line
}
...
int t[4]={1,2,3,4};
display(t); // replaces cout<<t which does not work
```

but two things must be remembered:

- An array is always passed as *reference* even though not using the '&'.^a
- A function cannot return an array.^b

^aA void f(int& t[4]) or any other syntax is an error.

^bWe will understand later why, for efficiency matters, the designers of C++ wanted an array not be passed by value and not returned.

hence:

```
1 // Reminder: this does not work
2 void set1(int x,int val) {
3     x=val;
4 }
5 // Reminder: that works!
6 void set2(int& x,int val) {
7     x=val;
8 }
9 // A function working without '&'
10 void fill(int s[4],int val) {
11     for (int i=0;i<4;i++)
```

⁴We will see below that assignment does not even work between two arrays! All that nonsense will be fixed with objects...

⁵It is frequent for programmers to use directly variables of type `std::vector` that are objects that implement all functionalities of arrays, while behaving like standard variables. We prefer not mentioning more `vector` since understanding them requires knowledge of objects and of "template". We believe that knowledge of arrays, even if requiring some effort, is mandatory and helps understanding memory management.


```

12     s[i]=val;
13 }
14 ...
15 int a=1;
16 set1(a,0);      // a not set to 0
17 cout << a << endl; // check
18 set2(a,0);      // a is set to 0
19 cout << a << endl; // check
20 int t[4];
21 fill(t,0);      // Put t[i] to 0
22 display(t);     // check all t[i] are 0

```

and also:

```

1 // Add two arrays , not even compiling
2 // To return an array
3 int sum1(int x[4],int y[4])[4] { // we could think adding
4 // [4] here or elsewhere:
5 // no way!
6     int z[4];
7     for (int i=0;i<4;i++)
8         z[i]=x[i]+y[i];
9     return z;
10 }
11 // In practice , we proceed like this
12 // Sum of two arrays that works.
13 void sum2(int x[4],int y[4],int z[4])
14     for (int i=0;i<4;i++)
15         z[i]=x[i]+y[i]; // OK: 'z' passed as reference!
16 }
17
18 int a[4],b[4];
19 ... // fill a and b
20 int c[4];
21 c=sum1(a,b); // ERROR
22 sum2(a,b,c); // OK

```

At last, and used every time,

A function is not restricted to work on arrays of a unique size...but it is impossible to ask an array its size!

We use the syntax `int t[]` as parameter to a function, without precisising its size. As we need to go along the array in the function and we cannot recover its size, we ask an additional argument for the function:

```

1 // Not working
2 void display1(int t[]) {
3     for (int i=0;i<SIZE(t);i++) // SIZE(t) does not exist!
4         cout << t[i] << ' ';
5 }

```

```

6 // What we do in practice
7 void display2(int t[],int n) {
8     for (int i=0;i<n;i++)
9         cout << t[i] << ' ';
10 }
11 ...
12 int t1[2]={1,2};
13 int t2[3]={3,4,5};
14 display2(t1,2); // OK
15 display2(t2,3); // OK

```

Note that even if a function `sizeof` does exist and we could substitute `SIZE` by it, it would compile but still not work.

4.3.2 Assignment

It's simple:

Assigning an array does not work! We need to handle elements of an array one by one...

The code

```

int s[4]={1,2,3,4},t[4];
t=s; // ERROR at compilation time

```

does not work and we need to write:

```

int s[4]={1,2,3,4},t[4];
for (int i=0;i<4;i++)
    t[i]=s[i]; // OK

```

The problem is that:

Assigning an array never works but does not always provoke a compilation error, nor even a warning. It is the case between two array arguments of functions. We will understand why later and what such an assignment really means...

```

1 // Function not working
2 // but compiling perfectly, thanks!
3 void set1(int s[4],int t[4]) {
4     t=s; // Does not do what we mean
5         // but compiles without warning!
6 }
7 // Working function, and compiling! :-)
8 void set2(int s[4],int t[4]) {
9     for (int i=0;i<4;i++)
10        t[i]=s[i]; // OK
11 }
12 ...

```

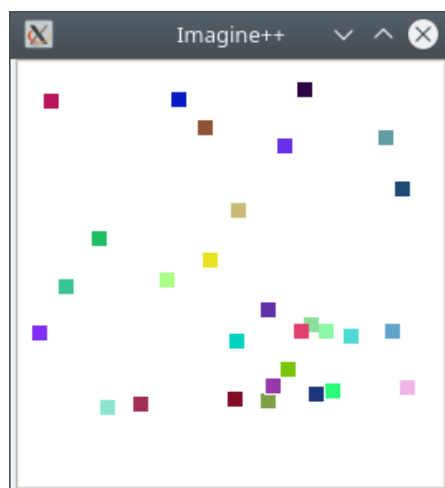


Figure 4.1: Bouncing balls... (still fixed! Go to web page for an animated program!)

```

13 int s[4]={1,2,3,4},t[4];
14 set1(s,t); // No effect
15 set2(s,t); // OK

```

4.4 Recreation

4.4.1 Multi-balls

We will take again the program with the bouncing ball, given in Section 3.1.4, but improved with functions and constants, of the practical of Appendix A.2. Thanks to arrays, it is easy to have several moving balls at once. We draw also randomly the color, initial position and initial speed of balls. Several functions may be unknown to you:

- The initialization of random generator with `srand((unsigned int)time(0))`, which is explained in Practical 3 (Appendix A.3)
- The functions `noRefreshBegin` and `noRefreshEnd` that are used to accelerate the display of all balls (see documentation of Imagine++ at Appendix B).

Here is the listing of the program (display example (unfortunately static!) Figure 4.1):

```

1 #include <Imagine/Graphics.h>
2 using namespace Imagine;
3 #include <cstdlib>
4 #include <ctime>
5 using namespace std;
6 ///////////////////////////////////////////////////////////////////
7 // Constants of program
8 const int width=256; // Window width
9 const int height=256; // Window height
10 const int ballRad=4; // Radius of ball

```

```

11 const int nb_balls=30; // Number of balls
12 ///////////////////////////////////////////////////////////////////
13 // Random generator
14 // Call it only once , before Random()
15 void initRand () {
16     srand((unsigned int)time (0));
17 }
18 // Between a and b
19 int random(int a,int b) {
20     return a+(rand()%(b-a+1));
21 }
22 ///////////////////////////////////////////////////////////////////
23 // Random position and speed
24 void initBall(int &x,int &y,int &u,int &v,Color &c) {
25     x=random(ballRad ,width-ballRad );
26     y=random(ballRad ,height-ballRad );
27     u=random(0 ,4);
28     v=random(0 ,4);
29     c=Color(byte(random(0 ,255)),
30             byte(random(0 ,255)),
31             byte(random(0 ,255)));
32 }
33 ///////////////////////////////////////////////////////////////////
34 // Display of ball
35 void displayBall(int x,int y,Color col) {
36     fillRect(x-ballRad ,y-ballRad ,2*ballRad+1,2*ballRad+1,col );
37 }
38 ///////////////////////////////////////////////////////////////////
39 // Move ball
40 void moveBall(int &x,int &y,int &u,int &v) {
41     // Left and right bouncing
42     if(x+u>width-ballRad || x+u<ballRad)
43         u=-u;
44     // Top and bottom bouncing
45     if(y+v<ballRad || y+v>height-ballRad)
46         v=-v;
47     // Update position
48     x+=u;
49     y+=v;
50 }
51 ///////////////////////////////////////////////////////////////////
52 // Main function
53 int main() {
54     // Open the window
55     openWindow(width ,height);
56     // Position and speed of balls
57     int xb[nb_balls],yb[nb_balls],ub[nb_balls],vb[nb_balls];
58     Color cb[nb_balls]; // Colors of balls

```

```

59     initRand ();
60     for (int i=0; i<nb_balls; i++) {
61         initBall (xb [ i ], yb [ i ], ub [ i ], vb [ i ], cb [ i ]);
62         moveBall (xb [ i ], yb [ i ], ub [ i ], vb [ i ]);
63     }
64     // Main loop
65     while (true) {
66         milliSleep (25);
67         noRefreshBegin ();
68         for (int i=0; i<nb_balls; i++) {
69             displayBall (xb [ i ], yb [ i ], WHITE);
70             moveBall (xb [ i ], yb [ i ], ub [ i ], vb [ i ]);
71             displayBall (xb [ i ], yb [ i ], cb [ i ]);
72         }
73         noRefreshEnd ();
74     }
75     endGraphics ();
76     return 0;
77 }

```

4.4.2 With shocks!

It is then not too complicated to modify the program so that balls bounce on each other. The following listing was built as follows:

1. When a ball moves, we check also if it meets another ball. Therefore, `moveBall` needs to know the location of other balls. We thus modify `moveBall` by passing complete arrays of positions and speeds, and just specifying the index of ball to move (lines 70 and 108). The loop of line 77 checks then via the test line 80 if another ball is shocked by the current one. In which case, we call `ShockBalls` that modifies the speeds of the balls. Note lines 78 and 79 that avoid considering the shock of a ball with itself (we will see the instruction `continue` another time).
2. The formulas for shocks can be found easily on the web. The function `shockBalls` implements the formulas. (Note the inclusion of file `<cmath>` to access the square root function `sqrt()`, the sine and cosine functions `cos()` and `sin()`, and arc-cosine function `acos()`).
3. We then realize that integer variables that store position and speed have rounding errors that accumulate and that speeds become null! We switch all concerned variables to `double`, thinking of converting them to `int` for display (line 36).⁶

All that gives a more animated program. We can obviously not observe the difference on a static figure. Download the program on the web page and try it!

⁶Forgetting the conversion is not a mistake, but the compiler may signal warnings to make the programmer aware that decimals may be lost in the implicit conversions. To appease it and let it know that the programmer is aware, an explicit conversion is advised.

```

22 ///////////////////////////////////////////////////////////////////
23 // Random position and speed
24 void initBall(double &x, double &y, double &u, double &v, Color &c){
25     x=random(ballRad ,width-ballRad );
26     y=random(ballRad ,height-ballRad );
27     u=random(0 ,4);
28     v=random(0 ,4);
29     c=Color (byte (random(0 ,255)),
30             byte (random(0 ,255)),
31             byte (random(0 ,255)));
32 }
33 ///////////////////////////////////////////////////////////////////
34 // Display of ball
35 void displayBall(double x, double y, Color col) {
36     fillRect (int(x)-ballRad , int(y)-ballRad ,
37             2*ballRad+1,2*ballRad+1,col );
38 }
39 ///////////////////////////////////////////////////////////////////
40 // Elastic shock of two spherical balls
41 // cf labo.ntic.org
42 #include <cmath>
43 void shockBalls(double&x1, double&y1, double&u1, double&v1,
44                double&x2, double&y2, double&u2, double&v2)
45 {
46     // Distance
47     double o2o1x=x1-x2, o2o1y=y1-y2;
48     double d=sqrt(o2o1x*o2o1x+o2o1y*o2o1y);
49     if(d==0) return; // Same center?
50     // Frame (o2, x, y)
51     double Vx=u1-u2, Vy=v1-v2;
52     double V=sqrt(Vx*Vx+Vy*Vy);
53     if(V==0) return; // Same speed?
54     // Next frame V (o2, i, j)
55     double ix=Vx/V, iy=Vy/V, jx=-iy, jy=ix;
56     // Height
57     double H=o2o1x*jx+o2o1y*jy;
58     // Angle
59     double th=acos(H/d), c=cos(th), s=sin(th);
60     // Speed after shock in (o2, i, j)
61     double v1i=V*c*c, v1j=V*c*s, v2i=V*s*s, v2j=-v1j;
62     // In original frame (O, x, y)
63     u1=v1i*ix+v1j*jx+u2;
64     v1=v1i*iy+v1j*jy+v2;
65     u2+=v2i*ix+v2j*jx;
66     v2+=v2i*iy+v2j*jy;
67 }
68 ///////////////////////////////////////////////////////////////////
69 // Motion of ball
70 void moveBall(double x[], double y[], double u[], double v[], int i)
71 { // Bounce on left and right borders
72     if(x[i]+u[i]>width-ballRad || x[i]+u[i]<ballRad)

```

```

73     u[i]=-u[i];
74     // Bound on top and bottom borders
75     if(y[i]+v[i]<ballRad || y[i]+v[i]>height-ballRad)
76         v[i]=-v[i];
77     for(int j=0; j<nb_balls; j++) {
78         if(j==i)
79             continue;
80         if(abs(x[i]+u[i]-x[j])<2*ballRad &&
81             abs(y[i]+v[i]-y[j])<2*ballRad) {
82             shockBalls(x[i],y[i],u[i],v[i],x[j],y[j],u[j],v[j]);
83         }
84     }
85     // Update position
86     x[i]+=u[i];
87     y[i]+=v[i];
88 }
89 ///////////////////////////////////////////////////////////////////
90 // Main function
91 int main() {
92     // Opening window
93     openWindow(width,height);
94     // Position and speed of balls
95     double xb[nb_balls],yb[nb_balls],ub[nb_balls],vb[nb_balls];
96     Color cb[nb_balls]; // Colors of balls
97     initRand();
98     for(int i=0; i<nb_balls; i++) {
99         initBall(xb[i],yb[i],ub[i],vb[i],cb[i]);
100        displayBall(xb[i],yb[i],cb[i]);
101    }
102    // Main loop
103    while(true) {
104        milliSleep(25);
105        noRefreshBegin();
106        for(int i=0; i<nb_balls; i++) {
107            displayBall(xb[i],yb[i],WHITE);
108            moveBall(xb,yb,ub,vb,i);
109            displayBall(xb[i],yb[i],cb[i]);
110        }
111        noRefreshEnd();
112    }
113    endGraphics();
114    return 0;
115 }

```

4.4.3 Shuffling letters

The following program considers a sentence and shuffles randomly the internal letters of each word (that is, without touching extremities). It uses the type `string`, string of characters, for which `s[i]` returns the i th character of string `s`, and `s.size()` the number of characters of `s` (we will explain later the notation “object” of this function). The

considered sentence may become:

Tihl lltle sennctee sohuld sltl be raedable for yuor poor biarn

Have you got it? Doesn't matter! It is the listing you will understand:

```

1 #include <iostream>
2 #include <string>
3 #include <cstdlib>
4 #include <ctime>
5 using namespace std;
6
7 ////////////////////////////////////////////////////
8 // Random generator
9 // Call only once, before Random()
10 void initRand() {
11     srand((unsigned int)time(0));
12 }
13 // Between a and b
14 int random(int a,int b) {
15     return a+(rand()%(b-a+1));
16 }
17
18 ////////////////////////////////////////////////////
19 // Shuffle internal letters n times
20 string shuffle(string s,int n) {
21     int l=int(s.size());
22     if (l<=3)
23         return s;
24     string t=s;
25     for(int i=0; i<n; i++) {
26         int a=random(1,l-2);
27         int b;
28         do
29             b=random(1,l-2);
30         while(a==b);
31         char c=t[a];
32         t[a]=t[b]; t[b]=c;
33     }
34     return t;
35 }
36
37 int main() {
38     const int n=11;
39     string phrase[n]={"This", "little", "sentence", "should",
40                     "still", "be", "readable", "for", "your",
41                     "poor", "brain"};
42
43     initRand();

```

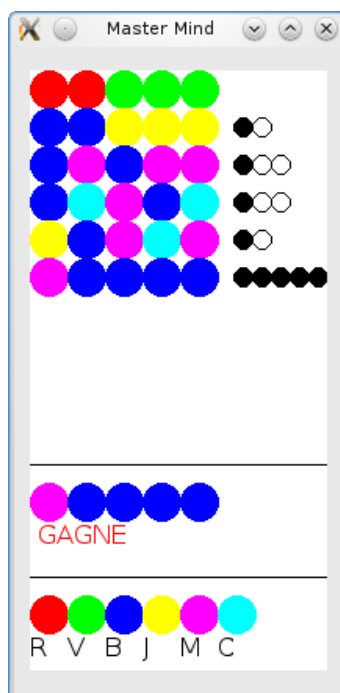



Figure 4.2: Mastermind...

```

44     for(int i=0; i<n; i++)
45         cout << shuffle(phrase[i],3) << " ";
46     cout << endl;
47
48     return 0;
49 }

```




4.5 Practical

We can now proceed to the third Practical of Appendix A.3 so as to understand better arrays and also to get a game of Mastermind (see Figure 4.2 for an example of an interesting game!).

4.6 Reference card

As promised, we complete, in **color**, the “reference card” with what we saw in the chapter and the associated Practical.

Reference Card (1/2)		
<p>Variables</p> <ul style="list-style-type: none"> • Definition: <pre>int i; int k,l,m;</pre> • Assignment: <pre>i=2; j=i; k=l=3;</pre> • Initialization: <pre>int n=5,o=n;</pre> • Constants: <pre>const int s=12;</pre> • Scope: <pre>int i; // i=j; forbidden! int j=2; i=j; // OK! if (j>1) { int k=3; j=k; // OK! } //i=k; forbidden!</pre> • Types: <pre>int i=3; double x=12.3; char c='A'; string s="hop"; bool t=true; float y=1.2f; unsigned int j=4; signed char d=-128; unsigned char d=25; complex<double> z(2,3);</pre> • Global variables: <pre>int n; const int m=12; void f() {</pre> 	<pre>n=10; // OK int i=m; // OK ...</pre> <ul style="list-style-type: none"> • Conversion: <pre>int i=int(x),j; float x=float(i)/j;</pre> <hr/> <p>Tests</p> <ul style="list-style-type: none"> • Comparison: <pre>== != < > <= >=</pre> • Negation: ! • Combinations: && • <pre>if (i==0) j=1;</pre> • <pre>if (i==0) j=1; else j=2;</pre> • <pre>if (i==0) { j=1; k=2; }</pre> • <pre>bool t=(i==0); if (t) j=1;</pre> • <pre>switch (i) { case 1: ...; ...; break; case 2: case 3: ...; break; default: ...; }</pre> <hr/> <p>Arrays</p> <ul style="list-style-type: none"> • Definition: 	<pre>- double x[5],y[5]; for(int i=0;i<5;i++) y[i]=2*x[i];</pre> <pre>- const int n=5; int i[n],j[2*n];</pre> <ul style="list-style-type: none"> • Initialization: <pre>int t[4]={1,2,3,4}; string s[2]={"ab","c"};</pre> • Assignment: <pre>int s[3]={1,2,3},t[3]; for (int i=0;i<3;i++) t[i]=s[i];</pre> • As parameter: <pre>- void init(int t[4]){ for(int i=0;i<4;i++) t[i]=0; } - void init(int t[], int n) { for(int i=0;i<n;i++) t[i]=0; }</pre> <hr/> <p>Loops</p> <ul style="list-style-type: none"> • <pre>do { ... } while(!ok);</pre> • <pre>int i=1; while(i<=100) { ... i=i+1; }</pre> • <pre>for(int i=1;i<=10;i++) ...</pre> • <pre>for(int i=1,j=10;j>i; i=i+2,j=j-3) ...</pre>

Reference Card (2/2)		
<p>Functions</p> <ul style="list-style-type: none"> • Definition: <pre>int plus(int a,int b){ int c=a+b; return c; } void display(int a) { cout << a << endl; }</pre> • Declaration: <pre>int plus(int a,int b);</pre> • Return: <pre>int sign(double x) { if (x<0) return -1; if (x>0) return 1; return 0; } void display(int x, int y) { if (x<0 y<0) return; if (x>=w y>=h) return; DrawPoint(x,y,RED); }</pre> • Call: <pre>int f(int a) { ... } int g() { ... } ... int i=f(2),j=g();</pre> • References: <pre>void swap(int& a, int& b){ int tmp=a; a=b;b=tmp; } ... int x=3,y=2; swap(x,y);</pre> • Overload: <pre>int chance(int n); int chance(int a, int b); double chance();</pre> 	<p>Misc.</p> <ul style="list-style-type: none"> • <code>i++;</code> <code>i--;</code> <code>i-=2;</code> <code>j+=3;</code> • <code>j=i%n;</code> // Modulo • <code>#include <cstdlib></code> ... <code>i=rand() %n;</code> <code>x=rand() /</code> <code>double (RAND_MAX);</code> • <code>#include <ctime></code> <code>// Single call</code> <code>srand((unsigned int)</code> <code>time(0));</code> • <code>#include <cmath></code> <code>double sqrt(double x);</code> <code>double cos(double x);</code> <code>double sin(double x);</code> <code>double acos(double x);</code> • <code>#include <string></code> <code>using namespace std;</code> <code>string s="hop";</code> <code>char c=s[0];</code> <code>int l=s.size();</code> <p>Input/Output</p> <ul style="list-style-type: none"> • <code>#include <iostream></code> <code>using namespace std;</code> ... <code>cout <<"I="<<i<<endl;</code> <code>cin >> i >> j;</code> <p>Frequent errors</p> <ul style="list-style-type: none"> • No definition of function inside a function! • <code>int q=r=4; // NO!</code> • <code>if (i=2) // NO!</code> <code>if i==2 // NO!</code> <code>if (i==2) then // NO!</code> • <code>for(int i=0,i<100,i++)</code> <code>// NO!</code> 	<ul style="list-style-type: none"> • <code>int f() {...}</code> <code>int i=f; // NO!</code> • <code>double x=1/3; // NO!</code> <code>int i,j;</code> <code>x=i/j; // NO!</code> <code>x=double(i/j); //NO!</code> • <code>double x[10],y[10];</code> <code>for(int i=1;i<=10;i++)</code> <code>y[i]=2*x[i];//NO</code> • <code>int n=5;</code> <code>int t[n]; // NO</code> • <code>int f()[4] { // NO!</code> <code>int t[4];</code> ... <code>return t; // NO!</code> <code>}</code> <code>int t[4]; t=f();</code> • <code>int s[3]={1,2,3},t[3];</code> <code>t=s; // NO!</code> • <code>int t[2];</code> <code>t={1,2}; // NO!</code> <p>Imagine++</p> <ul style="list-style-type: none"> • See documentation... <p>Keyboard</p> <ul style="list-style-type: none"> • Debug: F5  • Step over: F10  • Step inside: F11  • Indent: Ctrl+A, Ctrl+I <p>Advice</p> <ul style="list-style-type: none"> • Errors/warnings: click. • Indent! • Fix warnings. • Use the debugger. • Write functions. • Arrays: not to translate math formula!

Chapter 5

Structures

Functions and loops allowed us to group identical instructions, and arrays to group variables of the same type. However, to handle several variables together, it is necessary to build data structures...

5.1 Reminders

Before this, it may not be a waste of time to insert a small reminder that will be a repertory of classic errors of numerous beginners... and even some more rare but also more original of a few! Some would prevent compilation of the program, so are rather easy to fix; but others respect the syntax but do not do what is intended, they are harder to find. At last, we will repeat again the same advice.

5.1.1 Classic mistakes

With no particular order:

- Putting a single = in tests: `if (i=2)`
- Forgetting parentheses: `if i==2`
- Using then: `if (i==2) then`
- Putting commas in for: `for(int i=0, i<100, i++)`
- Forgetting parentheses when calling a function with no argument:

```
int f() {...}
...
int i=f;
```

- Assigning an array to another:

```
int s[4]={1,2,3,4}, t[4];
t=s;
```

5.1.2 Original mistakes

There, the beginner does not make a mistake, but invents squarely with the hope it would work. Often, neither does it exist, nor does it follow C++ syntax. Two examples:

- Mixing syntax (just a little!):

```
void set(int t[5]) {
    ...
}
...
int s[5];          // Fine up to this point
set(int s[5]);    // This is crazy!
```

instead of simply:

```
set(s);
```

- Wanting to do several things at once, or not understanding that **a program is a sequence of instructions to run one after another and not a formula.**¹ For instance, believing that `for` is a mathematical symbol as \sum_1^n or \bigcup_1^n . In this case, to run an instruction when all `ok(i)` are true, we have seen some trial like this:

```
if (for (int i=0; i<n; i++) ok(i)) // Big art ...
    ...
```

whereas we have to write:

```
bool allok=true;
for (int i=0; i<n; i++)
    allok=(allok && ok(i));
if (allok)
    ...
```

or even better (do you spot the difference?):

```
bool allok=true;
for (int i=0; i<n && allok; i++)
    allok=(allok && ok(i));
if (allok)
    ...
```

that may be simplified as:

```
bool allok=true;
for (int i=0; i<n && allok; i++)
    allok=ok(i);
if (allok)
    ...
```

¹Do not make me say what I did not! Theoretical computer scientists consider sometimes programs as formulas, but it is not relevant here!

It can be understood that the beginner could be subject to lack of knowledge, bad understanding of previous lessons, confusion with another language, or limitless imagination! However, it has to be remembered that a language is also a program, limited and conceived to do certain precise things. Therefore, it is wise to adopt the following prudent behavior:

Everything not presented as possible is impossible!

5.1.3 Advice

- Indent. Indent. **Indent, please!**²
- Click on error messages in your IDE to go directly to the right line!³
- Try not to leave warnings.
- The debugger is your friend, use it!

5.2 Structures

5.2.1 Definition

When arrays allow handling multiple variables of the same type, structures are used to group several variables to use them together. We create a *new type*, whose variables become “sub-variables” called *fields* of the structure. Here is an example of type `Point` with two fields of type `double` called `x` and `y`:

```
struct Point {
    double x, y;
};
```

Fields are defined with the same syntax as local variables in a function. Beware still:

Do not forget the semicolon after the definition of the structure!^a

^aAn unfortunate constraint imposed by compatibility with the C language.

The usage is then easy. The structure is a new type, handled exactly like others, with the specificity that **fields are accessed with a dot**:

```
Point a;
a.x = 2.3;
a.y = 3.4;
```

We can obviously define fields with different types, and even structures inside structures:

²Ctrl+A Ctrl+I in Qt Creator

³Window 1 “Issues” in Qt Creator

```

struct Circle {
    Point center;
    double radius;
    Color color;
};
Circle C;
C.center.x=12.;
C.center.y=13.;
C.radius=10.4;
C.color=RED;

```

The interest of structures is obvious and we must

Group in structures variables as soon as we understand they are logically linked. If a program becomes tiresome as we input systematically several identical variables to many functions, it is likely that the parameters could be grouped in a structure. It will become simpler and clearer.

5.2.2 Usage

Structures are used exactly as other types.⁴ Definition, assignment, initialization, argument passing, return of a function: everything is identical to basic, built-in types. Only innovation: **we use curly brackets to precise the values of fields in case of initialization.**⁵ We can of course have arrays of structures... and even have a field of type array! The following lines are not difficult to understand:

```

Point a={2.3,3.4},b=a,c; // Initialization
c=a; // Assignments
Circle C={{12,13},10.4,RED}; // Initialization
...
double dist(Point a,Point b) { // Value passing
    return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
}
void magnify(Circle& C,double scale) { // Reference passing
    C.radius*=scale; // Change radius
}
Point middle(Point a,Point b) { // return
    Point M;
    M.x=(a.x+b.x)/2;
    M.y=(a.y+b.y)/2;
    return M;
}
...
Point P[10]; // Array of structures
for(int i=0; i<10; i++) {
    P[i].x=i;
}

```

⁴Besides, we promised that arrays were particular (reference passing, no possible return, no assignment)

⁵As for an array!

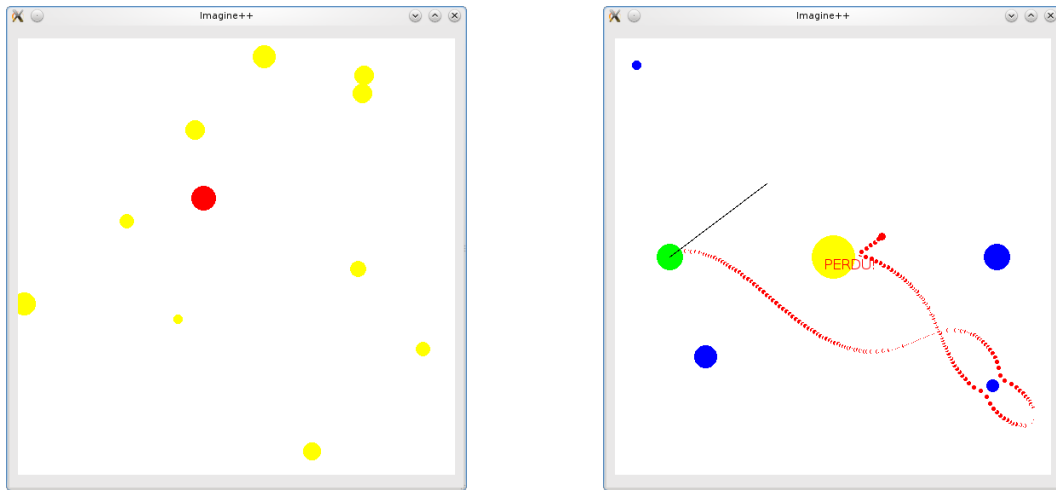


Figure 5.1: Celestial objects and duel...

```

    P[i].y=f(i);
}
...
// An embryo of a game of Yam's:
struct Drawing {
    int dice[5]; // Field of type array
};
Drawing roll_dice() {
    Drawing t;
    for(int i=0; i<5; i++)
        t.dice[i]=1+rand()%6; // Dice from 1 to 6
    return t;
}
...
Drawing t;
t=roll_dice();

```

Beware, like for arrays, the syntax used for initialization may not work for assignment:⁶

```

Point P;
P={1,2}; // Error!

```

Besides, let us repeat:

Everything not presented as possible is impossible!

⁶It will get better with objects. To be noticed: this restriction does not apply any more to simple structures with not too old compilers.

5.3 Recreation: Practical

We are now ready for Practical of Appendix A.4 to familiarize with structures. We will even have arrays of structures, yes!⁷ We will get a projectile navigating among stars then a space duel! (Figure 5.1)




5.4 Reference card

Once again, we complete, in **color**, the “reference card” with was has been learned in the chapter and Practical.

Reference Card (1/3)		
<p>Variables</p> <ul style="list-style-type: none"> • Definition: int i; int k, l, m; • Assignment: i=2; j=i; k=l=3; • Initialization: int n=5, o=n; • Constants: const int s=12; • Scope: int i; // i=j; forbidden! int j=2; i=j; // OK! if (j>1) { int k=3; j=k; // OK! } //i=k; forbidden! • Types: int i=3; double x=12.3; char c='A'; string s="hop"; bool t=true; float y=1.2f; unsigned int j=4; 	<pre>signed char d=-128; unsigned char d=25; complex<double> z(2,3);</pre> <ul style="list-style-type: none"> • Global variables: int n; const int m=12; void f() { n=10; // OK int i=m; // OK ... } • Conversion: int i=int(x), j; float x=float(i)/j; 	<ul style="list-style-type: none"> • bool t=(i==0); if (t) j=1; • switch (i) { case 1: ...; ...; break; case 2: case 3: ...; break; default: ...; }
	<p>Tests</p> <ul style="list-style-type: none"> • Comparison: == != < > <= >= • Negation: ! • Combinations: && • if (i==0) j=1; • if (i==0) j=1; else j=2; • if (i==0) { j=1; k=2; } 	<p>Loops</p> <ul style="list-style-type: none"> • do { ... } while (!ok); • int i=1; while (i<=100) { ... i=i+1; } • for(int i=1; i<=10; i++) ...; • for(int i=1, j=10; j>i; i=i+2, j=j-3) ...;

⁷For secondary school pupils: this Practice includes mathematics and physics for advanced students...but it can be done while ignoring all that!

Reference Card (2/3)		
<p>Functions</p> <ul style="list-style-type: none"> • Definition: <pre>int plus(int a,int b){ int c=a+b; return c; } void display(int a) { cout << a << endl; }</pre> • Declaration: <pre>int plus(int a,int b);</pre> • Return: <pre>int sign(double x) { if (x<0) return -1; if (x>0) return 1; return 0; } void display(int x, int y) { if (x<0 y<0) return; if (x>=w y>=h) return; DrawPoint(x,y,RED); }</pre> • Call: <pre>int f(int a) { ... } int g() { ... } ... int i=f(2),j=g();</pre> • References: <pre>void swap(int& a, int& b){ int tmp=a; a=b;b=tmp; } ... int x=3,y=2; swap(x,y);</pre> • Overload: <pre>int chance(int n); int chance(int a, int b); double chance();</pre> <hr/> <p>Tableaux</p> <ul style="list-style-type: none"> • Definition: <pre>- double x[5],y[5]; for(int i=0;i<5;i++) y[i]=2*x[i];</pre> 	<pre>- const int n=5; int i[n],j[2*n];</pre> <ul style="list-style-type: none"> • Initialization: <pre>int t[4]={1,2,3,4}; string s[2]={"ab","c"};</pre> • Assignment: <pre>int s[3]={1,2,3},t[3]; for (int i=0;i<3;i++) t[i]=s[i];</pre> • As parameter: <pre>- void init(int t[4]){ for(int i=0;i<4;i++) t[i]=0; } - void init(int t[], int n) { for(int i=0;i<n;i++) t[i]=0; }</pre> <hr/> <p>Structures</p> <ul style="list-style-type: none"> • <pre>struct Point { double x,y; Color c; }; ... Point a; a.x=2.3; a.y=3.4; a.c=RED; Point b={1,2.5,BLUE};</pre> <hr/> <p>Misc.</p> <ul style="list-style-type: none"> • <pre>i++; i--; i-=2; j+=3;</pre> • <pre>j=i%n; // Modulo</pre> • <pre>#include <cstdlib> ... i=rand()%n; x=rand()/ double(RAND_MAX);</pre> • <pre>#include <ctime> // Single call srand((unsigned int) time(0));</pre> • <pre>#include <cmath> double sqrt(double x); double cos(double x); double sin(double x); double acos(double x);</pre> 	<ul style="list-style-type: none"> • <pre>#include <string> using namespace std; string s="hop"; char c=s[0]; int l=s.size();</pre> <hr/> <p>Input/Output</p> <ul style="list-style-type: none"> • <pre>#include <iostream> using namespace std; ... cout <<"I="<<i<<endl; cin >> i >> j;</pre> <hr/> <p>Common errors</p> <ul style="list-style-type: none"> • No definition of function inside a function! • <pre>int q=r=4; // NO!</pre> • <pre>if (i=2) // NO! if i==2 // NO! if (i==2) then // NO!</pre> • <pre>for(int i=0,i<100,i++) // NO!</pre> • <pre>int f() {...} int i=f; // NO!</pre> • <pre>double x=1/3; // NO! int i,j; x=i/j; // NO! x=double(i/j); //NO!</pre> • <pre>double x[10],y[10]; for(int i=1;i<=10;i++) y[i]=2*x[i];//NO</pre> • <pre>int n=5; int t[n]; // NO</pre> • <pre>int f()[4] { // NO! int t[4]; ... return t; // NO! } int t[4]; t=f();</pre> • <pre>int s[3]={1,2,3},t[3]; t=s; // NO!</pre> • <pre>int t[2]; t={1,2}; // NO!</pre> • <pre>struct Point { double x,y; } // NO!</pre> • <pre>Point a; a={1,2}; // NO!</pre>

Reference Card (3/3)		
Imagine++ <ul style="list-style-type: none">• See documentation...	<ul style="list-style-type: none">• Step inside: F11 • Indent: Ctrl+A, Ctrl+I	<ul style="list-style-type: none">• Fix warnings.• Use the debugger.• Write functions.• Arrays: not to translate math formula!• Make structures.
Keyboard <ul style="list-style-type: none">• Debug: F5 • Step over: F10 	Advice <ul style="list-style-type: none">• Errors/warnings: click.• Indent!	

Chapter 6

Several Files!

At the latest Practical, we built two projects almost similar whose `main()` function was different. Modifying a function in the common part would require fixing it in both projects. We will now factorize this common part in a single file, so as to ease possible future modifications. Along the way¹ we will see how to define an operator on new types.

—

Let us sum up our progress in the programmer's know-how:

1. Programming all in `main()`: it is a beginning and it is already good!
2. Make functions: to be more readable and not to repeat! (Axis of instructions)
3. Make arrays and structures: to handle several variables together. (Axis of data)

We append now:

4. Make several files: to use common parts in different projects. (Again, axis of instructions)

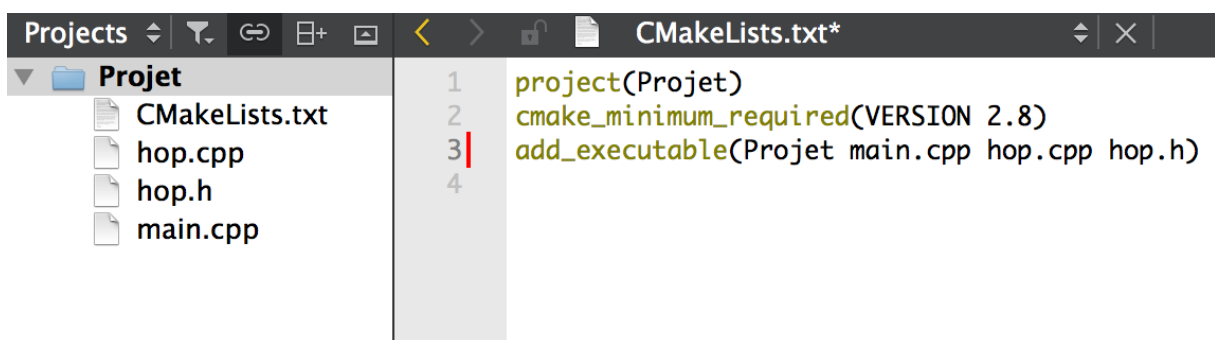


Figure 6.1: Several source files (in Qt Creator). Notice they are mentioned in line `add_executable` of file `CMakeLists.txt` with a simple blank space as separator.

¹Always the idea that we explore the components of the language when we need them.

6.1 Separate Files

We will distribute our source code into several files. Before all:

For maximum portability of code, choose file names with standard characters (neither accented characters nor spaces). Use also lower case letters only.^a

^aNothing prevents having upper case letters, but they may reduce portability if the case is not respected in the file `CMakeLists.txt`.

Besides, avoid also accents in identifiers (variable and function names), source code is not French...

6.1.1 Principle

Until now, a single source contained our full C++ program. This source file was transformed into object file by the compiler, then the linker completed the object file with C++ libraries to build an executable file. Actually, a project can contain **several source files**. It is enough to append a `.cpp` file to the list of project sources:

- In Qt Creator, open menu `File/New File or Project` or type `Ctrl+N`, then choose as model `C++ Source File`, give it a name.
- Append the file name in `CMakeLists.txt`:

```
add_executable(Hop main.cpp hop.cpp)
```

When adding the C++ file `hop.cpp` to a project with already `main.cpp`, we get in a situation identical to Figure 6.1 (be patient for the explanation of `hop.h` in the screenshot).

After that, each build of the project will include:

1. **Compilation**: every source file is transformed into object file (same name but with suffix `.obj`). Source files are thus compiled independently.
2. **Link**: all object files are grouped (and completed with C++ libraries) in a single executable file.

Part of the instructions of the main file (the one containing `main()`) can thus be transferred to another file. This part will be compiled separately and linked at the end of build. We then have the following problem: **how do we use in main file what is contained in other files?** Indeed, we knew (Section 3.2.4) that a function was “known” only in lines following its definition or declaration. By “known”, we meant that the compiler understands there exists somewhere a function with such name, such return type and arguments. Unfortunately:²

a function is not “known” outside its file. To use it in another file, we must declare it there!

²Fortunately, actually, as when we group files from different locations, it is better that not everything gets mixed in chaotic manner...

We can proceed as follows:

- File `hop.cpp`:

```
// Definitions
void f(int x) {
    ...
}
int g() {
    ...
}
// Other functions
...
```

- File `main.cpp`:

```
// Declarations
void f(int x);
int g();
...
int main() {
    ...
    // Usage
    int a=g();
    f(a);
    ...
}
```

We could also declare in `hop.cpp` some functions of `main.cpp` to use them. Beware still: if files use each other's functions, it may be the case that we did not separate sources conveniently.

6.1.2 Advantages

Our initial motivation was to put a part of the code in a separate file to **use it in another project**. Actually, cutting code in several files has other advantages:

- Make the code **more readable** and avoid too large files.
- **Make compilation faster**. When a program becomes long and complex, compilation time may increase significantly. When we build a project the programming environment only recompiles source files that were modified since the last build. It would be indeed useless to recompile a source file not modified and get the same object file!³ Therefore, changing a few lines in a file will not force a compilation of the full program but only of the concerned file.

Beware still not to separate into too many files! It becomes then more complex to grasp the logic and to navigate between files.

³It is actually slightly more complicated: a source may depend, via inclusions (Section 6.1.4), on other files, that may have been modified themselves! We need then to recompile a file whose dependency was modified. Such dependencies are automatically handled by CMake.

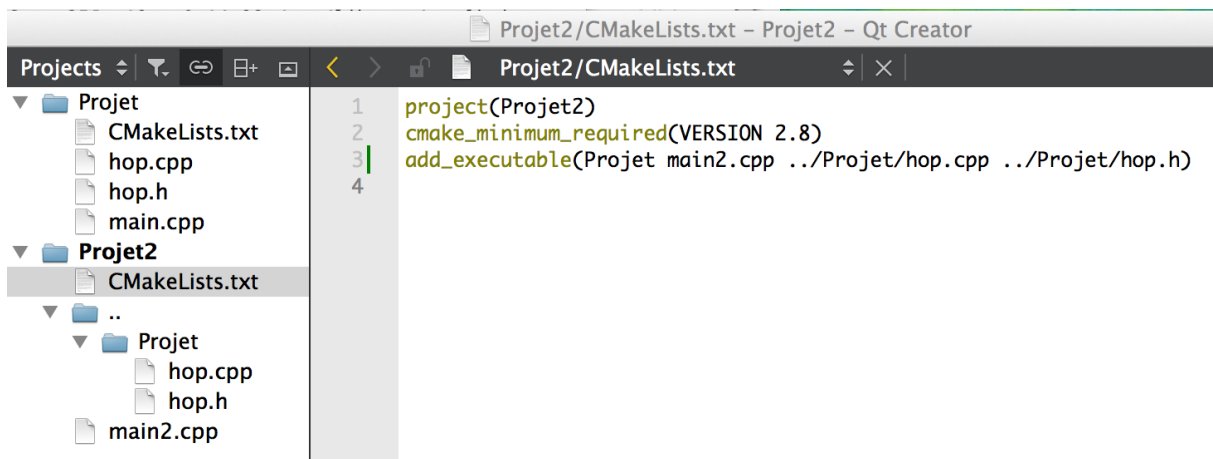


Figure 6.2: Same source file in two projects: `hop.cpp` and `hop.h` are shared between both projects, even though they are in same folder `Projet`

6.1.3 Usage in another project

6.1.4 Header files

A separate file allows use to factorize source code. However, we need to type declarations in all files using them. We can do better.⁴ For that, it is time to explain the instruction `#include` that we used from the beginning:

The line `#include "name"` is automatically replaced with the contents of file `name` before proceeding to compilation.

It's all about replacing by the complete contents of file `name` as with a simple copy/-paste. This operation is done just before the compilation step by a program we did not talk about: the *preprocessor*. Lines beginning with `#` are for it. We will see others. Notice that until now we used a slightly different syntax: `#include <name>`, which fetches file `name` in C++ libraries.⁵

Thanks to this possibility of the preprocessor, we just need to put the declarations related to the separate file in a third file and *include* it in main files. It is usual to use the same name for this additional file, but with extension `.h`: we call it a header file.⁶ To create a file, do as the source, but choose "C++ Header File" instead of "C++ Source File". Here is what we get:

- File `hop.cpp`:

```
// Definitions
void f(int x) {
    ...
}
```

⁴Again the least effort...

⁵Header files as `iostream`, etc. are sometimes called *system headers*. Their name does not end with the usual `.h` (see later), but nothing distinguishes them fundamentally from a usual header file. Some names of system headers begin with an unexpected letter `c`, like `cmath`; it is because they are inherited from language C.

⁶`.h` like header, we can see also sometimes `.hpp` to distinguish them from C headers.


```

int g() {
    ...
}
// Other functions
...

```

- File `hop.h`:

```

// Declarations
void f(int x);
int g();

```

- File `main.cpp` of first project:

```

#include "hop.h"
...
int main() {
    ...
    // Usage
    int a=g();
    f(a);
    ...
}

```

- File `main2.cpp` of second project (we may need to precise the location of the header file in the hierarchy, which may be in the folder of the first project⁷):

```

#include "../Project1/hop.h"
...
int main() {
    ...
    // Usage
    f(12);
    int b=g();
    ...
}

```

Actually, to be sure that functions defined in `hop.cpp` are coherent with their declaration in `hop.h`, and though not mandatory, we include also the header file in source, yielding:

- File `hop.cpp`:

```

#include "hop.h"
...
// Definitions
void f(int x) {
    ...
}
int g() {
    ...
}

```

⁷One can also precise to the compiler a list of folders where to fetch header files, see Section 6.1.8.

```

}
// Other functions
...

```

In practice, the header file does not contain only declaration of functions but also definition of new types (structures) used by the separate file. Indeed, such new types must be known from separate file but also from main file. We must therefore:

1. Include in header file the declaration of functions and definition of new types.
2. Include the header file in the main file and in the separate file.

For example:

- File `vect.h`:

```

// Types
struct Vector {
    double x,y;
};
// Declarations
double norm(Vector V);
Vector plus(Vector A,Vector B);

```

- File `vect.cpp`:

```

#include "vect.h" // Functions and types
// Definitions
double norm(Vector V) {
    ...
}
Vector plus(Vector A,Vector B) {
    ...
}
// Other functions
...

```

- File `main.cpp` of first project:

```

#include "vect.h"
...
int main() {
    ...
    // Usage
    Vector C=plus(A,B);
    double n=norm(C);
    ...
}

```

Your development environment allows navigating easily between files of your project. For instance with Qt Creator, you can switch from header to source or vice-versa by option “Switch Header/Source” (F4) or by contextual menu with right click in editor. To follow the include, you can also set position on the name of the file and choose option “Follow Symbol Under Cursor” (F2) of the same menu.

6.1.5 Don’ts...

It is “strongly” advised to

1. not declare in the header file all functions of separate file, but only the ones used by the main file. Secondary files do not need to appear.⁸
2. not include a separate file itself! It is generally a “big mistake”.⁹ Therefore

never #include "vect.cpp"!

6.1.6 Implementation

Finally, the philosophy of the system is that

- **The separate file and its header form a coherent set, *implementing certain functionalities*.**
- **The one using them, which may not be the one that programmed them, just adds these files to the project, include header in sources and take advantage of what the header declares.**
- **The header file must be clear and informative enough so that the user does not need to look in the separate file itself.^a**

^aBesides, the user looking at it may be tempted to use more than the header declares. The creator of the separate file and of the header can later be led to change in the source the manner to program the functions without changing the functionalities. The user has not respected the rule of the game and “cheated” by looking inside the separate file. We will come back to that with objects. We will be able to prevent the user from cheating. . . Anyway, at our level, creator and user are the same person.

6.1.7 Mutual inclusions

When practicing, the beginner discovers frequently problems not anticipated from the course. The most frequent happening with header files is the one of *mutual inclusion*. It may happen that header files need to include others. If file `A.h` includes `B.h` and if `B.h` includes `A.h` then every inclusion of `A.h` or `B.h` results in endless inclusions

⁸We should even hide them to prevent the principal file from using them. It would be possible to do this right away, but we will talk again about that with objects. . .

⁹A same function can be defined several times: in the separate file and in the principal file that includes it. Whereas it is possible to declare many times at will a function, it is forbidden to define it several times (do not mix that with overload that makes possible to existence of several functions with the same name—see Section 3.2.6)

that provoke eventually an error.¹⁰ To avoid the problem, we use an instruction of the preprocessor signaling that a file included once must not be included again: we add

#pragma once at the beginning of the header file.

Some compilers (or rather, preprocessors) may not understand `#pragma once` because it is not standard.¹¹ In that case, a trick is used, that we expose without explanation:

- Choose a name unique to the header file. For instance `VECT_H` for file `vect.h`.
- Write `#ifndef VECT_H` and `#define VECT_H` at beginning of file `vect.h` and `#endif` at the end.

This uses the command `ifdef` of the preprocessor. Let us note another usage sometimes useful when developping so that the compiler ignores a block of code (maybe because it is not ready yet):

```
#if 0
```

Whatever is here will be ignored by the compiler.

```
#endif
```

6.1.8 Inclusion path

Since `#include` takes a file, a legitimate question is where the preprocessor fetches it. For system headers, such as `#include <iostream>`, the location is known from the compiler and we do not need to care about them. For others, two rules are used:

- in current folder, the one containing the `cpp`;
- in a list of folders indicated by the user.

In the second case, there is an instruction `include_directories` in CMake, to be used in `CMakeLists.txt`. For example, when one writes¹²

```
#include "Imagine/Graphics.h"
```

it will search for a folder `Imagine` containing a file `Graphics.h`. (Always use direct slash `/`, which works on all platforms, not backslash `\` that works only under Windows.) To know where to find it, we have in `CMakeLists.txt`:

```
find_package(Imagine)
ImagineUseModules(Mastermind Graphics)
```

The command `ImagineUseModules` is specific to `Imagine++` (available after the command `find_package`), but calls `include_directories` by giving as path

```
<Imagine_DIR>/include
```

with `<Imagine_DIR>` the installation folder of `Imagine++`.

¹⁰Preprocessors are fortunately able to detect this.

¹¹The compiler you will use does, so prefer the advice we gave instead of the portable one.

¹²Note that preprocessor directives do not end with a semicolon.

6.2 Operators

C++ allows defining operators +, -, etc. when operands are new types. Here is how to do it. The reader is encouraged to discover alone which operators may be defined.

Let's consider the following example that defines a vector 2D and implements addition:¹³

```
struct vect {
    double x,y;
};
vect plus(vect m,vect n) {
    vect p={m.x+n.x,m.y+n.y};
    return p;
}
int main() {
    vect a={1,2},b={3,4};
    vect c=plus(a,b);
    return 0;
}
```

Here is how to define + between vect and replace function plus():

```
struct vect {
    double x,y;
};
vect operator+(vect m,vect n) {
    vect p={m.x+n.x,m.y+n.y};
    return p;
}
int main() {
    vect a={1,2},b={3,4};
    vect c=a+b;
    return 0;
}
```

We can also define a product by a scalar, a scalar product¹⁴, etc.¹⁵

```
// Product by a scalar
vect operator*(double s,vect m) {
    vect p={s*m.x,s*m.y};
    return p;
}
// Scalar product
double operator*(vect m,vect n) {
    return m.x*n.x+m.y*n.y;
}
```

¹³**For secondary school pupils:** You do not know what is a vector...but you are better in programming than the "old ones". Then, look at the sources that follow and you will know what a 2D vector is.

¹⁴In this case, we may use $a*b$ but not $a.b$, the dot being not amenable to redefinition and reserved for access to fields of a structure

¹⁵But beware, it is impossible to redefine operators on basic types! No way to give a different sense to $1+1$.

```

}
int main() {
    vect a={1,2},b={3,4};
    vect c=2*a;
    double s=a*b;
    return 0;
}

```

Notice that both defined functions are different even though with same name ([operator*](#)) since they take different parameters (remember overload Section 3.2.6).

6.3 Fun: Practical continued and finished

The program of preceding Practical was a perfect example of the need of separate files (well defined structures, shared by two projects), we propose you in Practice A.4 to convert (and finish?) the program of gravitation simulation and space duel!

6.4 Reference card

You know the drill...

Reference Card (1/3)		
<p>Variables</p> <ul style="list-style-type: none"> • Definition: int i; int k,l,m; • Assignment: i=2; j=i; k=l=3; • Initialization: int n=5,o=n; • Constants: const int s=12; • Scope: int i; // i=j; forbidden! int j=2; i=j; // OK! if (j>1) { int k=3; j=k; // OK! } //i=k; forbidden! • Types: int i=3; 	<pre> double x=12.3; char c='A'; string s="hop"; bool t=true; float y=1.2f; unsigned int j=4; signed char d=-128; unsigned char d=25; complex<double> z(2,3); </pre> <ul style="list-style-type: none"> • Global variables: int n; const int m=12; void f() { n=10; // OK int i=m; // OK ... • Conversion: int i=int(x),j; float x=float(i)/j; <hr/> <p>Tests</p> <ul style="list-style-type: none"> • Comparison: == != < > <= >= • Negation: ! 	<ul style="list-style-type: none"> • Combinations: && • if (i==0) j=1; • if (i==0) j=1; else j=2; • if (i==0) { j=1; k=2; } • bool t=(i==0); if (t) j=1; • switch (i) { case 1: ...; ...; break; case 2: case 3: ...; break; default: ...; }

Reference Card (2/3)		
<p>Loops</p> <ul style="list-style-type: none"> do { ... } while(!ok); int i=1; while(i<=100) { ... i=i+1; } for(int i=1;i<=10;i++) ... for(int i=1,j=10;j>i; i=i+2,j=j-3) ... 	<pre>a=b;b=tmp; } ... int x=3,y=2; swap(x,y);</pre> <ul style="list-style-type: none"> Overload: int chance(int n); int chance(int a, int b); double chance(); Operators: vect operator+(vect A,vect B) { ... } ... vect C=A+B; 	<pre>Point a; a.x=2.3; a.y=3.4; a.c=RED; Point b={1,2.5,BLUE};</pre> <p>Separate compilation</p> <ul style="list-style-type: none"> #include "vect.h", also in vect.cpp Functions: declarations in .h, definitions in .cpp Types: definitions in .h Declare in .h only useful functions #pragma once at beginning of header file Do not cut too much...
<p>Functions</p> <ul style="list-style-type: none"> Definition: int plus(int a,int b){ int c=a+b; return c; } void display(int a) { cout << a << endl; } Declaration: int plus(int a,int b); Return: int sign(double x) { if (x<0) return -1; if (x>0) return 1; return 0; } void display(int x, int y) { if (x<0 y<0) return; if (x>=w y>=h) return; DrawPoint(x,y,RED); } Call: int f(int a) { ... } int g() { ... } ... int i=f(2),j=g(); References: void swap(int& a, int& b){ int tmp=a; 	<p>Arrays</p> <ul style="list-style-type: none"> Definition: - double x[5],y[5]; for(int i=0;i<5;i++) y[i]=2*x[i]; - const int n=5; int i[n],j[2*n]; Initialization: int t[4]={1,2,3,4}; string s[2]={"ab","c"}; Assignment: int s[3]={1,2,3},t[3]; for (int i=0;i<3;i++) t[i]=s[i]; As parameter: - void init(int t[4]){ for(int i=0;i<4;i++) t[i]=0; } - void init(int t[], int n) { for(int i=0;i<n;i++) t[i]=0; } 	<p>Misc.</p> <ul style="list-style-type: none"> i++; i--; i-=2; j+=3; j=i%n; // Modulo #include <cstdlib> ... i=rand() %n; x=rand() / double(RAND_MAX); #include <ctime> // Single call srand((unsigned int) time(0)); #include <cmath> double sqrt(double x); double cos(double x); double sin(double x); double acos(double x); #include <string> using namespace std; string s="hop"; char c=s[0]; int l=s.size();
	<p>Structures</p> <ul style="list-style-type: none"> struct Point { double x,y; Color c; }; ... 	<p>Input/Output</p> <ul style="list-style-type: none"> #include <iostream> using namespace std; ... cout <<"I="<<i<<endl; cin >> i >> j;

Reference Card (3/3)

Common errors


- No definition of function inside a function!
- `int q=r=4; // NO!`
- `if (i=2) // NO!`
`if i==2 // NO!`
`if (i==2) then // NO!`
- `for(int i=0,i<100,i++)`
`// NO!`
- `int f() {...}`
`int i=f; // NO!`
- `double x=1/3; // NO!`
`int i,j;`
`x=i/j; // NO!`
`x=double(i/j); //NO!`
- `double x[10],y[10];`
`for(int i=1;i<=10;i++)`
`y[i]=2*x[i];//NO`
- `int n=5;`
`int t[n]; // NO`



- `int f()[4] { // NO!`
`int t[4];`
`...`
`return t; // NO!`
`}`
`int t[4]; t=f();`
- `int s[3]={1,2,3},t[3];`
`t=s; // NO!`
- `int t[2];`
`t={1,2}; // NO!`
- `struct Point {`
`double x,y;`
`} // NO!`
- `Point a;`
`a={1,2}; // NO!`
- `#include "tp.cpp">//NO`

Imagine++

- See documentation...

Keyboard

- Debug: F5 

- Step over: F10 
- Step inside: F11 
- Indent: Ctrl+A, Ctrl+I
- Switch source/header: F4
- Swith decl./def.: F2

Advice

- Errors/warnings: click.
- Indent!
- Fix warnings.
- Use the debugger.
- Write functions.
- Arrays: not to translate math formula!
- Make structures.
- Do separate source files
- The `.h` must be enough for the user (who must not look into `.cpp`)

Chapter 7

The memory

It is time to get back on the subject of memory and its use. We will then understand better local variables, the exact way a function call works, recursive functions, etc. After that, we will at last be able to use arrays with variable size (without really explaining the delicate notion of pointer).

—

7.1 Call of function

It is here a new opportunity to understand at last what happens in a program...

7.1.1 Example

Consider the following program:

```
1 #include <iostream>
2 using namespace std;
3
4 void check(int p, int q, int quo, int re) {
5     if (re<0 || re>=q || q*quo+re!=p)
6         cout << "What_the_f*k!" << endl;
7 }
8
9 int divide(int a, int b, int& r) {
10     int q;
11     q=a/b;
12     r=a-q*b;
13     check(a, b, q, r);
14     return q;
15 }
16 int main()
17 {
18     int num, denom;
19     do {
20         cout << "Input_two_positive_integers:_";
```

```

21     cin >> num >> denom;
22 } while (num<=0 || denom<=0);
23 int quotient, rem;
24 quotient=divide (num,denom, rem );
25 cout << num << "/" << denom << " = " << quotient
26     << "(remains " << rem << ")" << endl;
27 return 0;
28 }

```

Computing quotient and remainder of an integer division, and verifying the result, it is not so impressive and inordinately long (actually, only lines 11 and 12 do the job!). It is however a good example to illustrate our lesson. A good way to explain fully how it works is to fill out the following table, already met in Practical A.2. Putting only lines where variables change, assuming the user inputs 23 and 3 at the keyboard, and indexing with letters the different steps of a same line of code,¹ it gives:

Line	<i>num</i>	<i>denom</i>	<i>quotient</i>	<i>rem</i>	<i>a</i>	<i>b</i>	<i>r</i>	<i>q_d</i>	<i>ret_d</i>	<i>p_v</i>	<i>q_v</i>	<i>quo</i>	<i>re</i>
18	?	?											
21	23	3											
23	23	3	?	?									
24a	23	3	?	?									
9	23	3	?	?	23	3	[rem]						
10	23	3	?	?	23	3	[rem]	?					
11	23	3	?	?	23	3	[rem]	7					
12	23	3	?	2	23	3	[rem]	7					
13a	23	3	?	2	23	3	[rem]	7					
4	23	3	?	2	23	3	[rem]	7		23	3	7	2
5	23	3	?	2	23	3	[rem]	7		23	3	7	2
7	23	3	?	2	23	3	[rem]	7					
13b	23	3	?	2	23	3	[rem]	7					
14	23	3	?	2	23	3	[rem]	7	7				
15	23	3	?	2					7				
24b	23	3	7	2					7				
25	23	3	7	2									
28													

A posteriori, we can see that we have implicitly assumed that when the program is running `divide()`, the function `main()` and its variables still exist and that they simply wait for the end of `divide()`. In other words:

A call of function is a mechanism that allows going to run temporarily this function then recovering the next instructions and the variables left behind.

Functions call others, we get nested function calls: `main()` calls `divide()` that calls `check()`.² More precisely, this hierarchy is a *stack* and we talk about the **call stack**. To understand better this stack, we are going to use the debugger. Before that, just make clear what a computer scientist calls a *stack*.

¹such as 24a and 24b

²Besides, `main()` was itself called by a function to which it returns an `int`.



Stack/Queue

- A **stack** is a structure allowing memorizing data in which they are ordered such that the piece inserted last is the one extracted first. A stack is also called LIFO (last in first out). We *push* data on it and we *pop* from it. For example, after `push(1)`, `push(2)` then `push(3)`, the first `pop()` gives 3, the second `pop()` gives 2 and the last `pop()` gives 1.
- For a **queue**, it is the same except that the first in is the first out (FIFO). For instance, after `push(1)`, `push(2)` then `push(3)`, the first `pop()` gives 1, the second gives 2 and the last 3.

7.1.2 Call stack and debugger

Let us observe Figure 7.1, built by running the program under the debugger. Checking the part displayed at each step, we can see the call stack. The right part displays the contents of variables, parameters and return values from which we can check the coherence with the previous table.

- As the call stack indicates, we are line 24 in function `main()`, that happens to be the only element on the stack (main is the entry point of the program). Notice the variables take nonsense values (here 32764 for `quotient` and 276543888 for `rem`) as long as they are not assigned.
- Go on with step-by-step advance (key F11) until line 12. We are in function `divide()`, `q` has value 7, and line 24 of `main()` went down one step in the call stack.
- We are now at line 5 in `check()`. The call stack has one more level, `divide()` is on standby at line 13 and `main()` still at 24. Check that variable `q` is the one of `check()`, which is 3, and not the one of `divide()`.
- Here, **the program has not advanced** and we are still at line 5. However, the debugger allows, by clicking on the call stack, to watch the state at inferior levels, notably to display instructions and variables at said level. Here, clicking on line `divide()` in the window of the call stack, we see **line 13 and its variables in the state they had while the program was at 5**. Among others, displayed `q` is the one of `divide()` and worth 7.
- Still without progress, here is the state of `main()` and its variables (among others, `rem` went to 2 since line 12 of `divide()`).
- We run now the next instructions until line 24 at the return of `divide()`. For that, we can step-by-step advance, or simply twice exit step-by-step³ to continue until exit from `check()`, then until exit of `divide()`. We can see `quotient`, that is still not defined, and also the return value of `divide()`, not yet assigned to `quotient`.
- One step further and we are at 25/26. The variable `quotient` is worth 7 at last.

³Step Out or Maj-F11 or  Notice also the possibility of continuing the program until a certain line is reached without having to put a breakpoint but simply by clicking on the line with the right mouse button and choosing "Run to Line...", ()

(a)

Level	Function	File	Line
1	main	calls.cpp	24

Name	Value	Type
denom	3	int
num	23	int
quotient	32764	int
rem	276543888	int

(b)

Level	Function	File	Line
1	divide	calls.cpp	12
2	main	calls.cpp	24

Name	Value	Type
a	23	int
b	3	int
q	7	int
r	276543888	int &

(c)

Level	Function	File	Line
1	check	calls.cpp	5
2	divide	calls.cpp	13
3	main	calls.cpp	24

Name	Value	Type
p	23	int
q	3	int
quo	7	int
re	2	int

(d)

Level	Function	File	Line
1	check	calls.cpp	5
2	divide	calls.cpp	13
3	main	calls.cpp	24

Name	Value	Type
a	23	int
b	3	int
q	7	int
r	2	int &

(e)

Level	Function	File	Line
1	check	calls.cpp	5
2	divide	calls.cpp	13
3	main	calls.cpp	24

Name	Value	Type
denom	3	int
num	23	int
quotient	32764	int
rem	2	int

(f)

Level	Function	File	Line
1	main	calls.cpp	24

Name	Value	Type
denom	3	int
num	23	int
quotient	32764	int
rem	2	int
returned va...	7	int

(g)

Level	Function	File	Line
1	main	calls.cpp	25

Name	Value	Type
denom	3	int
num	23	int
quotient	7	int
rem	2	int

Figure 7.1: Function calls

stack	variable	value	stack	variable	value	stack	variable	value
				free space				
	free space		top→	p _c	23			
				q _c	3		free space	
				quo	7			
				re	2			
top→	a	23		a	23			
	b	3		b	3			
	q _d	7		q _d	7			
	r	[rem]		r	[rem]			
	denom	3		denom	3	top→	denom	3
	num	23		num	23		num	23
	quotient	?		quotient	?		quotient	7
	rem	?		rem	2		rem	2
	taken by	...		taken by	...		taken by	...
	functions	...		functions	...		function	...
	before main	...		before main	...		before main	...

step (b) (line 12)
step (c) (line 5)
step (g) (line 25)

Figure 7.2: Stack and local variables

7.2 Local variables

It will become important for what follows to know how parameters and local variables are stored in memory.

7.2.1 Parameters

For parameters, it's simple:

Parameters of functions are actually local variables! Their only specificity is that they are initialized from the start of the function with the values passed at the call (the call *arguments*).

7.2.2 The stack

Local variables (and thus function parameters also) are not stored at fixed memory locations,⁴ defined at build time. If so, such memory locations would need to be reserved for the whole run of the program: we could not use the memory for local variables of other functions. The solution is more economical in memory:⁵

Local variables are memorized in a stack:

- When a local variable is created, it is pushed on top of the stack.
- When it dies (in general, when we get out of the function), it is removed from the stack.

⁴Recall Chapter 2.

⁵And will allow having recursive functions, see next section!

Therefore, along function calls, local variables stack one of top one another: memory is used only the time necessary. Figure 7.2 shows three states of the stack when running our example.

7.3 Recursive functions

A recursive function is one that calls itself. The most classical function to illustrate recursion is the factorial.⁶ A compact and easy way to compute it follows:

```

5 int fact1(int n) {
6     if(n==1)
7         return 1;
8     return n*fact1(n-1);
9 }
```

We notice, it is obvious but important, it contains a *stop condition*: if n is 1, the function returns directly 1 without calling itself.⁷

7.3.1 Why does it work?

If functions had memorized their local variables at fixed addresses, recursion would not have worked: The recursive call would have crushed the values of the variables. For instance, `fact1(3)` would have crushed the value 3 memorized in n by a 2 when calling `fact1(2)`! That is precisely thanks to the stack that n of `fact1(2)` is not the same as the one of `fact1(3)`. We get the following table

Line	$n_{fact1(3)}$	$ret_{fact1(3)}$	$n_{fact1(2)}$	$ret_{fact1(2)}$	$n_{fact1(1)}$	$ret_{fact1(1)}$
5 _{fact1(3)}	3					
9a _{fact1(3)}	3					
5 _{fact1(2)}	3		2			
9a _{fact1(2)}	3		2			
5 _{fact1(1)}	3		2		1	
8 _{fact1(1)}	3		2		1	1
10 _{fact1(1)}	3		2			1
9b _{fact1(2)}	3		2	2		1
10 _{fact1(2)}	3			2		
9b _{fact1(3)}	3	6		2		
10 _{fact1(3)}		6				

The table becomes difficult to write now that we know that local variables do not depend only on the function but change at each call! We need to precise, for each line number, which function call is concerned. If we visualize the stack, we understand better how it works. At line 8 of `fact1(1)` for an initial call at `fact1(3)`, the stack looks like:

⁶**For secondary school pupils:** The factorial of an integer n , written $n!$, is $n! = 1 \times 2 \times \dots \times n$.

⁷The fact we can put a `return` at the middle of the function is convenient here! Notice also there is no need for `else`.

stack	variable	value
	free space	
top→	$n_{fact1(1)}$	1
	$n_{fact1(2)}$	2
	$n_{fact1(3)}$	3

which we can easily check with the debugger. Finally:

Recursive functions are not different from other functions. It is the mechanism of function calls that make recursion possible.

7.3.2 Efficiency

A recursive function is simple and elegant to write when the problem is adequate.⁸ We have seen that it is not easy to follow or debug. You also need to know that

The stack space is not infinite and even relatively limited.

The following program

```

22 // Stack overflow
23 int fact3(int n) {
24     if(n==1)
25         return 1;
26     return n*fact3(n+1); // error!
27 }
```

in which a slipped small error will call theoretically infinitely and in practice stop with a stack overflow.⁹ But the true reason recursion is sometimes avoided is that

Calling a function has a cost!

When the core of a function is small enough so that the cost of calling the function becomes not negligible w.r.t. the time spent running the function itself, it is better to avoid the function call.¹⁰ In the case of a recursive function, we therefore try if necessary to write a *derecurvised* version (or *iterative*). For the factorial, it is:

```

// Iterative version
int fact2(int n) {
    int f=1;
    for(int i=2; i<=n; i++)
        f*=i;
    return f;
}
```

which after all is not that terrible.

At last, some recursive function cannot be used for complexity reason. A classic example is the Fibonacci sequence defined by

$$\begin{cases} f_0 = f_1 = 1 \\ f_n = f_{n-1} + f_{n-2} \end{cases}$$

⁸Abusing recursion is a classic beginner mistake.

⁹Under Linux, at about $n = 260,000$.

¹⁰We will see in another chapter the `inline` functions that answer the problem.

that yields: 1, 1, 2, 3, 5, 8,... The recursive version is a direct copy of the formula:

```

32 // Very slow!
33 int fib1(int n) {
34     if(n<2)
35         return 1;
36     return fib1(n-2)+fib1(n-1);
37 }

```

This function has the bad idea to call itself too frequently: $n = 10$ calls $n = 9$ and $n = 8$, but $n = 9$ has first to call also $n = 8$ from its side and $n = 7$ on top, $n = 7$ that will be called by all $n = 8$, etc. This function will become quickly too slow. For $n = 40$, it calls itself already 300.000.000 times, which takes its toll! It is reasonable to program a non-recursive version:

```

39 // Not recursive
40 int fib2(int n) {
41     int fnm2=1,fnm1=1; // n minus 1 and 2
42     for(int i=2; i<=n; i++) {
43         int fn=fnm2+fnm1;
44         fnm2=fnm1;
45         fnm1=fn;
46     }
47     return fnm1;
48 }

```

7.4 The heap

The stack is not the only zone of memory used by programs. There is also the *heap*.

7.4.1 Limits

The stack is limited in size. The call stack being not infinite and local variables not coming in unlimited number, it is reasonable to reserve a stack of limited size. Try the small program:

```

32 int main() {
33     const int n=500000;
34     int t[n];
35     ...
36 }

```

It runs and stops with an error: “stack overflow”. The local variable `t` is not too large for the computer:¹¹ it is too large for the stack. Until now, we knew we were limited to fixed size arrays. In reality, we are also limited to **small arrays**. It is high time we learn using the heap!

¹¹500000x4 is 2 MB only!

7.4.2 Variable size arrays

We provide here a rule that is applied blindly. Its understanding will come later if necessary.

When we want to use an array of a variable size, there are only two things to do, but they are both essential:¹²

1. Replace `int t[n]` by `int* t=new int[n]` (or its equivalent for another type than `int`)
2. When the array must die (in general at the end of the function), append the line `delete[] t;`

Not obeying rule 2 has as consequence that the arrays stays in memory until the end of the program, which provokes an uncontrolled increase of used memory (we call that a *memory leak*). For the remainder, nothing changes. Programming in this manner makes the array memorized **in the heap and not any more in the stack**. We proceed like this:

1. For variable size arrays and/or
2. For arrays of big size.

It should be made clear that when we talk of arrays of *variable size*, it does not mean that the array can grow (or shrink!) once created. It only means that the size of the array can be a (non constant) variable, whose value is not known at compilation time. Actually, there is no mechanism to resize an array after creation.

Here is the result on a small program:

```

1 #include <iostream>
2 using namespace std;
3
4 void fill(int t[], int n) {
5     for (int i=0;i<n;i++)
6         t[i]=i+1;
7 }
8
9 int sum(int t[], int n) {
10    int s=0;
11    for (int i=0;i<n;i++)
12        s+=t[i];
13    return s;
14 }
15
16 void fixed() {
17    const int n=5000;
18    int t[n];
19    fill(t,n);
20    int s=sum(t,n);
21    cout << s << "_should_be_" << n*(n+1)/2 << endl;

```

¹²And the beginner forgets always the second one, which yields programs that grow in used memory...

```

22 }
23
24 void variable() {
25     int n;
26     cout << "An_integer_please:_";
27     cin >> n;
28     int* t=new int[n]; // Allocation
29     fill(t,n);
30     int s=sum(t,n);
31     cout << s << "_should_be_" << n*(n+1)/2 << endl;
32     delete [] t; // Deallocation: do not forget!
33 }
34
35 int main() {
36     fixed();
37     variable();
38     return 0;
39 }

```

7.4.3 Explanation (or trial of)

What follows is not essential for a beginner but can possibly answer some questions. If the reader understands, all the better, otherwise following preceding rules is good enough!

To have access to all memory of the computer,¹³ we use the heap. It is a memory zone that the program owns and that can grow on demand to the operating system (and if there is available memory left, obviously). To use the heap, we call an *allocation* function that reserves memory for a given amount of variables. That is the effect of `new int[n]`.

The function returns the address of the memory location the OS has reserved. We never met a variable able to memorize an address. They are called pointers, which will reappear later. A pointer to memory storing some `int` is of type `int*`. Hence the `int* t` to store the result of `new`.

Next, a pointer can be used as an array, even as an argument to function.

Finally, we must not forget to free the memory at the moment when the fixed size array would have disappeared: that's the job of instruction `delete [] t`, freeing the memory addressed by `t`.

7.5 The optimizer

Let us mention an important point, neglected up to this point, but that will be used in practicals.

There are several ways to translate a C++ source in machine language. The result of compilation may be different from one compiler to the next. At compilation time, we can seek to get an executable as fast as possible: we say the compiler *optimizes* the

¹³More exactly to the part of memory that the operating system is willing to allocate to each program, which may be tunable but anyway less than the total memory, although much more than stack memory.

code. In general, optimization requires a greater work but also transformations that make the resulting program unsuitable for debugging. In practice, we need to choose between possibility of debugging and optimization.

Until now, we used the compiler in mode “Debug”. When a program works (and not before), we can try it in mode “Release” to have a more efficient program. In some cases, the gains can be huge. An experienced programmer is even able to help the optimizer do its job. But a few rules must be respected:

- Do not try to debug in Release mode(!)
- Stay in Debug mode as long as possible to code the program.

7.6 Assertions

Here comes a very useful function to help avoid bugs! The function `assert()` warns when a test is false. It says the file and the line number where the error occurred and lets the debugger stop exactly there. It does not slow down the program as it disappears in Release mode. It is a function not well known among beginners, and it is a pity! For instance:

```
#include <cassert>
...
int n;
cin >> n;
assert(n>0);
int* t=new int[n]; // Allocation
```

If the user inputs a negative number, consequences could be serious. In particular, a negative value of `n` will be interpreted as a big integer (since the brackets `[]` expect an unsigned integer, `-1` is understood as the largest `int` possible in a modulo fashion) and the `new` would probably fail. Note that if `n==0`, an empty array, the allocation is a success (of 0 byte!). But in this case `t[0]` does not even exist! The only thing doable with a null array is to deallocate it with `delete[] t`. It is always useful to protect from such an exception by checking the value is reasonable.

7.7 Bidimensional arrays

Let us anticipate slightly on the next chapter with dimension 2 arrays. If `m` and `n` are *constant*, one can have an array `tab[m][n]`, that we can even put inside a structure:

```
const int m=5, n=3;
struct Array { double tab[m][n]; };
void f(Array& t) {
    for(int i=0; i<m; i++)
        for(int j=0; j<n; j++)
            t.tab[i][j] = cos(M_PI*i/m)*sin(M_PI*j/n);
    cout << t.tab[m-1][n-1] << endl;
}
```

Note the double bracket pairs to access an element, and of course the first element is `tab[0][0]` and the last one `tab[m-1][n-1]`.

7.8 Reference card

Reference Card (1/3)		
<p>Variables</p> <ul style="list-style-type: none"> • Definition: <pre>int i; int k,l,m;</pre> • Assignment: <pre>i=2; j=i; k=l=3;</pre> • Initialization: <pre>int n=5,o=n;</pre> • Constants: <pre>const int s=12;</pre> • Scope: <pre>int i; // i=j; forbidden! int j=2; i=j; // OK! if (j>1) { int k=3; j=k; // OK! } //i=k; forbidden!</pre> • Types: <pre>int i=3; double x=12.3; char c='A'; string s="hop"; bool t=true; float y=1.2f; unsigned int j=4;</pre> 	<pre>signed char d=-128; unsigned char d=25; complex<double> z(2,3);</pre> <ul style="list-style-type: none"> • Global variables: <pre>int n; const int m=12; void f() { n=10; // OK int i=m; // OK ...</pre> • Conversion: <pre>int i=int(x),j; float x=float(i)/j;</pre> • Stack/Heap <hr/> <p>Tests</p> <ul style="list-style-type: none"> • Comparison: <pre>== != < > <= >=</pre> • Negation: ! • Combinations: && • <pre>if (i==0) j=1;</pre> • <pre>if (i==0) j=1; else j=2;</pre> • <pre>if (i==0) { j=1; k=2; }</pre> 	<ul style="list-style-type: none"> • <pre>bool t=(i==0); if (t) j=1;</pre> • <pre>switch (i) { case 1: ...; ...; break; case 2: case 3: ...; break; default: ...; }</pre> <hr/> <p>Loops</p> <ul style="list-style-type: none"> • <pre>do { ... } while(!ok);</pre> • <pre>int i=1; while(i<=100) { ... i=i+1; }</pre> • <pre>for(int i=1;i<=10;i++) ...</pre> • <pre>for(int i=1,j=10;j>i; i=i+2,j=j-3) ...</pre>

Reference Card (2/2)

Functions

- **Definition:**

```
int plus(int a,int b){
    int c=a+b;
    return c;
}
void display(int a) {
    cout << a << endl;
}
```
- **Declaration:**

```
int plus(int a,int b);
```
- **Return:**

```
int sign(double x) {
    if (x<0)
        return -1;
    if (x>0)
        return 1;
    return 0;
}
void display(int x,
             int y) {
    if (x<0 || y<0)
        return;
    if (x>=w || y>=h)
        return;
    DrawPoint(x,y,RED);
}
```
- **Call:**

```
int f(int a) { ... }
int g() { ... }
...
int i=f(2),j=g();
```
- **References:**

```
void swap(int& a,
          int& b){
    int tmp=a;
    a=b;b=tmp;
}
...
int x=3,y=2;
swap(x,y);
```
- **Overload:**

```
int chance(int n);
int chance(int a,
           int b);
double chance();
```
- **Operators:**

```
vect operator+(
    vect A,vect B) {
    ...
}
...
vect C=A+B;
```
- **Call stack**

• Iterative/Recursive

Arrays

- **Definition:**

```
- double x[5],y[5];
   for(int i=0;i<5;i++)
       y[i]=2*x[i];
- const int n=5;
   int i[n],j[2*n];
```
- **Initialization:**

```
int t[4]={1,2,3,4};
string s[2]={"ab","c"};
```
- **Assignment:**

```
int s[3]={1,2,3},t[3];
for (int i=0;i<3;i++)
    t[i]=s[i];
```
- **As parameter:**

```
- void init(int t[4]){
    for(int i=0;i<4;i++)
        t[i]=0;
}
- void init(int t[],
           int n) {
    for(int i=0;i<n;i++)
        t[i]=0;
}
```
- **Variable size:**

```
int* t=new int[n];
...
delete[] t;
```

Structures

- ```
struct Point {
 double x,y;
 Color c;
};
...
Point a;
a.x=2.3; a.y=3.4;
a.c=RED;
Point b={1,2.5,BLUE};
```

## Separate compilation

- #include "vect.h", also in vect.cpp
- Functions: declarations in .h, definitions in .cpp
- Types: definitions in .h
- Declare in .h only useful functions

- #pragma once at beginning of header file
- Do not cut too much...





## Misc.

- i++;  
i--;  
i-=2;  
j+=3;
- j=i%n; // Modulo
- #include <cstdlib>  
...  
i=rand() %n;  
x=rand() /  
double(RAND\_MAX);
- #include <ctime>  
// Single call  
srand((unsigned int)  
time(0));
- #include <cmath>  
double sqrt(double x);  
double cos(double x);  
double sin(double x);  
double acos(double x);
- #include <string>  
using namespace std;  
string s="hop";  
char c=s[0];  
int l=s.size();
- #include <ctime>  
s=double(clock())  
/CLOCKS\_PER\_SEC;

## Input/Output

- #include <iostream>  
using namespace std;  
...  
cout <<"I="<<i<<endl;  
cin >> i >> j;

## Keys

- Debug: F5 
- Step over: F10 
- Step inside: F11 
- Indent: Ctrl+A, Ctrl+I
- Switch source/header: F4
- Swith decl./def.: F2
- Step out: Maj+F11 

## Reference Card (3/3)

**Frequent errors**

- No definition of function inside a function!
- `int q=r=4; // NO!`
- `if (i=2) // NO!`  
`if i==2 // NO!`  
`if (i==2) then // NO!`
- `for(int i=0,i<100,i++)`  
`// NO!`
- `int f() {...}`  
`int i=f; // NO!`
- `double x=1/3; // NO!`  
`int i,j;`  
`x=i/j; // NO!`  
`x=double(i/j); //NO!`
- `double x[10],y[10];`  
`for(int i=1;i<=10;i++)`  
`y[i]=2*x[i];//NO`
- `int n=5;`  
`int t[n]; // NO`

- `int f()[4] { // NO!`  
`int t[4];`  
`...`  
`return t; // NO!`  
`}`  
`int t[4]; t=f();`
- `int s[3]={1,2,3},t[3];`  
`t=s; // NO!`
- `int t[2];`  
`t={1,2}; // NO!`
- `struct Point {`  
`double x,y;`  
`} // NO!`
- `Point a;`  
`a={1,2}; // NO!`
- `#include "tp.cpp">//NO`

**Imagine++**

- See documentation...

**Advice**

- Errors/warnings: click.

- Indent!
- Fix warnings.
- Use the debugger.
- Write functions.
- Arrays: not to translate math formula!
- Make structures.
- Do separate source files
- The `.h` must be enough for the user (who must not look into `.cpp`)
- **Don't abuse recursion.**
- **Don't forget delete.**
- **Compile regularly.**
- `#include <cassert>`  
`...`  
`assert(x!=0);`  
`y=1/x;`

# Chapter 8

## Dynamic Allocation

We come back once more on the use of the heap to handle arrays of variable size. After mentioning bidimensional arrays of fixed size, we present in details dynamic allocation<sup>1</sup> already seen in Section 7.4.2 and explain at last pointers, at least partially. Through the example of matrices (and images in practical session) we mix structures and dynamic allocation. It will be our most complex data structure up to now before the long awaited arrival—then justified—of objects...

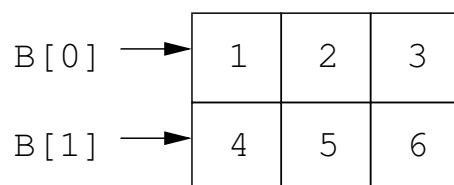
### 8.1 Bidimensional arrays

#### 8.1.1 Principle

There are in C++ bidimensional arrays. Their use is similar to standard arrays:

- We use brackets (lines 1 and 4 of program below). Beware: `[i][j]` and not `[i,j]`.
- Initialization is possible with curly brackets (line 5). Beware: nested curly brackets.
- Their dimensions must be constant (lines 6 and 7).

```
1 int A[2][3];
2 for(int i=0; i<2; i++)
3 for(int j=0; j<3; j++)
4 A[i][j]=i+j;
5 int B[2][3]={{1,2,3},{4,5,6}};
6 const int M=2,N=3;
7 int C[M][N];
```



2D array. Note that B[0] is the 1D array {1,2,3} and B[1] the 1D array {4,5,6}.

The figure above shows the array B. Note that B[0] and B[1] are 1D arrays representing the lines of B.

---

<sup>1</sup>that is, allocation in heap memory with `new` and `delete`.

### 8.1.2 Limitations

Compared to functions, specifics are the same as in 1D:

- Impossible to return a 2D array.
- Argument passing only by variable (not by value).

but with an additional restriction:

**We need to give the dimensions of a 2D array used as parameter of a function.**

It is then impossible to program functions that work on arrays of different sizes as we did in 1D case (see Section 4.3.1). It is a strong restriction and explains that this form of 2D arrays is rarely used. We can have the following program:

```

1 // Argument passing
2 double trace(double A[2][2]) {
3 double t=0;
4 for(int i=0; i<2; i++)
5 t+=A[i][i];
6 return t;
7 }
8
9 // Always by variable
10 void set(double A[2][3]) {
11 for(int i=0; i<2; i++)
12 for(int j=0; j<3; j++)
13 A[i][j]=i+j;
14 }
15
16 ...
17 double D[2][2]={{1,2},{3,4}};
18 double t=trace(D);
19 double E[2][3];
20 set(E);
21 ...

```

but it is not possible to program a function `trace()` or `set()` that works with 2D arrays of different sizes:

```

1 // OK
2 void set(double A[],int n,double x) {
3 for(int i=0; i<n; i++)
4 A[i]=x;
5 }
6 // NO!!!!!!!!!!!!!!!!!!!!!!
7 // double A[][] is not accepted
8 void set(double A[][],double m,double n,double x) {
9 for(int i=0; i<m; i++)
10 for(int j=0; j<n; j++)

```



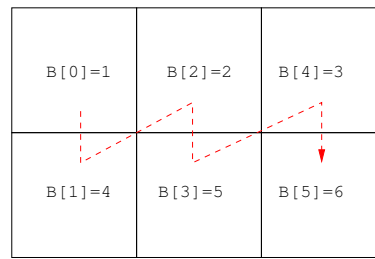


Figure 8.1: Matrix B of the previous example stored in 1D.

```

11 A[i][j]=x;
12 }

```

### 8.1.3 Solution

In practice, as soon as need to handle 2D arrays (or higher-dimension!) of different sizes, we store them in 1D arrays by sequencing the columns (for example) to take advantage of 1D arrays. We will store a matrix  $A$  of  $m$  rows and  $n$  columns in an array  $T$  of size  $mn$  by storing logical element  $A(i, j)$  in  $T(i + mj)$ . Figure 8.1 shows an array B of the previous example stored in a 1D array. We can then write:

```

1 void set(double A[], int m, int n) {
2 for(int i=0; i<m; i++)
3 for(int j=0; j<n; j++)
4 A[i+m*j]=i+j;
5 }
6 ...
7 double F[2*3];
8 set(F,2,3);
9 double G[3*5];
10 set(G,3,5);

```

or for instance, this product matrix-vector in which matrix as well as vector are stored in a 1D array:

```

1 // y=Ax
2 void product(double A[], int m, int n, double x[], double y[])
3 {
4 for(int i=0; i<m; i++) {
5 y[i]=0;
6 for(int j=0; j<n; j++)
7 y[i]+=A[i+m*j]*x[j];
8 }
9 }
10 ...
11 double P[2*3],x[3],y[2];
12 ...
13 // P=... x=...
14 product(P,2,3,x,y); // y=Px

```

## 8.2 Dynamic allocation

There is no dynamic allocation for 2D arrays. We must therefore store them in 1D arrays as explained above to be able to use dynamic allocation. The following example shows how to proceed. It uses the function `product()` of the code above without requiring redefinition:

```

1 int m,n;
2 ...
3 double* A=new double [m*n];
4 double* x=new double [n];
5 double* y=new double [m];
6 ...
7 // A=... x=...
8 product(A,m,n,x,y); // y=Ax
9 ...
10 delete [] A;
11 delete [] x;
12 delete [] y;
```

### 8.2.1 Why does it work?

It is now time we explain why, once allocated, we can use dynamic arrays exactly as fixed size arrays. The following points should be sufficient:

1. `int t[n]` defines a local variable, hence memory on the stack, capable of storing  $n$  variables `int`.
2. `int* t` defines a variable of type “pointer” of `int`, that is,  $t$  can memorize the address of a memory zone containing some `int`.
3. `new int[n]` allocates in heap a memory zone able of storing  $n$  `int` and returns the address of the zone. Hence the `int* t=new int[n]`
4. `delete [] t` frees (in heap) the memory zone memorized in  $t$ .
5. When  $t$  is a fixed size array, `t[i]` indicates its  $i^{\text{th}}$  element. When  $t$  is a pointer on `int`, `t[i]` is the variable `int` stored  $i$  places<sup>2</sup> further in memory than the one at address  $t$ . After `int t[n]` as well as after `int* t=new int[n]`, the syntax `t[i]` indicates what we mean.
6. When  $t$  is a fixed size array, the syntax  $t$  alone designates the address (in the stack) at which the array is stored. Moreover, when a function takes as argument an array, the syntax `int s[]` indicates in reality that  $s$  is the address of the array. Which explains after all:
  - a function `f(int s[])` is meant so that its argument is an address  $s$
  - it works with dynamically allocated arrays, which are only addresses after all

---

<sup>2</sup>Here, a place is obviously the number of bytes for the storage of an `int`.

- it is rather the call  $f(t)$ , with  $t$  fixed size array, which needs to adapt by passing to  $f$  the address of the array.
- logically, we should declare  $f$  by  $f(\text{int}^* s)$  instead of  $f(\text{int } s[])$ . Both are actually possible and equivalent.

You can thus now program, *while understanding*, such code:

```

1 double sum(double* t, int n) { // Syntax "pointer"
2 double s=0;
3 for(int i=0; i<n; i++)
4 s+=t[i];
5 return s;
6 }
7 ...
8 int t1[4];
9 ... // Fill t1
10 double s1=sum(t1,4); // Call with static array
11 int* t2=new int[n];
12 ... // Fill t2
13 double s2=sum(t2,n); // Call with dynamic array
14 ...
15 delete [] t2;

```

### 8.2.2 Classical errors

You are now able to understand the following classic errors (without avoiding making them though!).

1. Forgetting to allocate:

```

int *t;
for(int i=0; i<n; i++)
 t[i]=... // Horror: uninitialized t
 // is a random address

```

2. Forgetting to deallocate:

```

void f(int n) {
 int *t=new int[n];
 ...
} // We forget delete[] t;
 // Each call to f() loses n int in the heap!

```

3. Deallocating the wrong array:

```

int* t=new int[n];
int* s=new int[n];
...
s=t; // No! Then, s is the same address as t
 // (we did not copy the zone pointed by t in the one

```

```

 // pointed by s!)
...
delete [] t; // OK
delete [] s; // Havoc: Not only some memory is not freed
 // (initially stored in s), but we deallocate
 // also once more the one we just did!

```

### 8.2.3 Consequences

#### When to deallocate?

Now you have understood `new` and `delete`, you can imagine that we do not need to wait the end of existence of an array to deallocate it. The sooner the better and we deallocate as soon as the memory is not used anymore:

```

1 void f() {
2 int t[10];
3 int* s=new int[n];
4 ...
5 delete [] s; // if s is not used in the following...
6 // why not deallocate here?
7 ...
8 } // On the contrary, t has to wait here to die.

```

Actually, the array of address memorized in `s` is allocated line 3 and freed line 5. The variable `s` itself, that stores this address, is created line 3 and dies line 8!

#### Pointers and functions

It is frequent that `new` and `delete` do not happen in the same function (beware then to not forget!). They are often inside different functions. By the way, when functions manipulate pointers, some questions may be raised. It is enough to follow the logic:

- A function that returns a pointer is declared `int* f()`;

```

1 int* allocate(int n) {
2 return new int[n];
3 }
4
5 int* t=allocate(10);
6 ...

```

- A pointer passed to a function is by value. Do not confuse with the fact that an array is passed by reference! Let's consider the following program:

```

1 void f(int* t, int n) {
2
3 t[i]=...; // Modify t[i] but not t!
4 t=... // Such a line would not change argument s
5 // of the call
6 }

```

```

7 ...
8 int* s=new int [m];
9 f(s,m);

```

Actually, it is because we pass the address of an array that we can modify its elements. By ignorance, we pretended that arrays are passed by reference and presented this as an exception. We can now rectify:

**An array is passed by its address. This address is passed by value. But this mechanism allows the function to modify the array. Saying that an array was passed by reference was an abuse of language.**

- If we want really to pass the pointer by reference (seldom), the syntax is logical: `int*& t`. A typical need is:

```

1 // t and n will be modified (not only t[i])
2 void allocate(int*& t,int& n) {
3 cin >> n; // n chosen by user
4 t=new int [n];
5 }
6 ...
7 int* t;
8 int n;
9 allocate(t,n); // t and n both assigned by allocate()
10 ...
11 delete [] t; // Still do not forget!

```

**Strange syntax?** Lines 7 and 8 above could be written in one line `int* t,n`. Actually, we need to put a star before each variable when one defines several pointers at once. Hence, `int *t,s,*u;` defines two pointers of `int` (variables `t` and `u`) and one `int` (variable `s`).

## 8.3 Structures and dynamic allocation

Passing systematically an array and its size to all functions is obviously a hassle. We need to unite them in a structure. I leave you meditate the following example that could be part of a program using matrices and their product.<sup>3</sup>

```

1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 //=====
6 // functions on matrices
7 // could be in matrix.h and matrix.cpp

```

<sup>3</sup>For secondary school pupils: matrices and vectors are unknown to you. Don't worry. Understand the source code and go to the Practical, playing with images that look like matrices.

```

8
9 struct Matrix {
10 int m,n;
11 double* t;
12 };
13
14 Matrix create(int m,int n) {
15 Matrix M;
16 M.m=m;
17 M.n=n;
18 M.t=new double[m*n];
19 return M;
20 }
21
22 void destroy(Matrix M) {
23 delete [] M.t;
24 }
25
26 Matrix product(Matrix A,Matrix B) {
27 if(A.n!=B.m) {
28 cout << "Error!" << endl;
29 exit(1);
30 }
31 Matrix C=create(A.m,B.n);
32 for(int i=0; i<A.m; i++)
33 for(int j=0;j<B.n;j++) {
34 // Cij=Ai0*B0j+Ai1*B1j+...
35 C.t[i+C.m*j]=0;
36 for(int k=0; k<A.n; k++)
37 C.t[i+C.m*j]+=A.t[i+A.m*k]*B.t[k+B.m*j];
38 }
39 return C;
40 }
41
42 void display(string s,Matrix M) {
43 cout << s << "␣=" << endl;
44 for(int i=0; i<M.m; i++) {
45 for(int j=0; j<M.n; j++)
46 cout << M.t[i+M.m*j] << "␣";
47 cout << endl;
48 }
49 }
50
51 //=====
52 // Usage
53
54 int main()
55 {

```

```

56 Matrix A=create (2 ,3);
57 for(int i=0; i<2; i++)
58 for(int j=0; j<3; j++)
59 A.t[i+2*j]=i+j;
60 display ("A" ,A);
61 Matrix B=create (3 ,5);
62 for(int i=0; i<3; i++)
63 for(int j=0; j<5; j++)
64 B.t[i+3*j]=i+j;
65 display ("B" ,B);
66 Matrix C=product(A,B);
67 display ("C" ,C);
68 destroy(C);
69 destroy(B);
70 destroy(A);
71 return 0;
72 }

```

The user now only needs to remember to allocate and free the matrices by calling the functions but without needing to know what these functions do. In this logic, we will add functions so that the user does not need either to keep in mind how the elements of the matrix are stored. And there is then no need to remember that there is a field named *t*! (We are getting really close to object oriented programming...) We add:

```

10 double get(Matrix M,int i ,int j) {
11 return M.t[i+M.m*j];
12 }
13
14 void set(Matrix M,int i ,int j ,double x) {
15 M.t[i+M.m*j]=x;
16 }

```

that the user can use so:

```

51 for(int i=0; i<2; i++)
52 for(int j=0; j<3; j++)
53 set(A,i ,j ,i+j);

```

and that the designer of these matrices could also use:

```

39 void display(string s ,Matrix M) {
40 cout << s << " = " << endl;
41 for(int i=0; i<M.m; i++) {
42 for(int j=0; j<M.n; j++)
43 cout << get(M,i ,j) << " ";
44 cout << endl;
45 }
46 }

```

Beware, it is easy in this context to:

- Forget allocation.

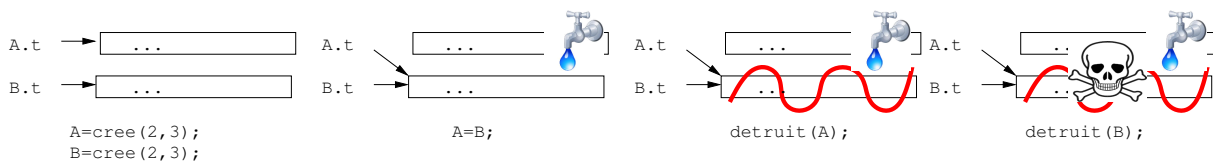


Figure 8.2: Beware of the dreaded double delete: the code `A=B` makes pointers address the same memory zone whereas there is no more pointer on the top array (hence a memory leak since it is not possible anymore to deallocate it). The `detruit(B)` frees a memory that was already freed, with dire consequences...

- Forget deallocation.
- Not deallocating what must if one writes `A=B` between two matrices. (It is then twice the zone initially allocated for B that is deallocated when one frees A and B while the initial memory for A will never be, as illustrated by Figure 8.2).

Object oriented programming will make sure we don't make these mistakes. It will also make sure the user does not know what needs not be known, so as to make the conception of matrices really independent on their usage.

## 8.4 Loops and `continue`

We will use in the Practical the convenient instruction `continue`. Here is its effect: met in a loop, all the rest of loop iteration is skipped and the next iteration can begin:

```
for (...) {
 ...
 if (A)
 continue;
 ...
 if (B)
 continue;
 ...
}
```

is equivalent to (and is better for clarity):

```
for (...) {
 ...
 if (!A) {
 ...
 if (!B) {
 ...
 }
 }
}
```

It can be compared with the usage of `return` in the middle of function to avoid particular cases (Section 7.3).





Figure 8.3: Two images and different processing of the second one (negative, blur, relief, deformation, contrast and edges).

## 8.5 Practical

The Practical proposed at [A.6](#) illustrates this way of manipulating 2D dynamical arrays through data structures. To change from matrices (however fascinating they can be!), we will work on images (Figure [8.3](#)).

## 8.6 Reference card

| Reference Card (1/4)                                                                                                                                            |                                                                                                         |                                                                                                                                                                                   |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Loops</b></p> <ul style="list-style-type: none"> <li>do {<br/>  ...<br/>} while(!ok);</li> <li>int i=1;<br/>while(i&lt;=100) {<br/>  ...<br/>}</li> </ul> | <pre> i=i+1; } • for(int i=1;i&lt;=10;i++)   ... • for(int i=1,j=10;j&gt;i;   i=i+2,j=j-3)   ... </pre> | <ul style="list-style-type: none"> <li>for (int i=...)       for (int j=...) {         //skip case i==j         if (i==j)           continue;         ...       }     </li> </ul> |

| Reference Card (2/4)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Variables</b></p> <ul style="list-style-type: none"> <li>• <b>Definition:</b><br/> <pre>int i; int k,l,m;</pre> </li> <li>• <b>Assignment:</b><br/> <pre>i=2; j=i; k=1=3;</pre> </li> <li>• <b>Initialization:</b><br/> <pre>int n=5,o=n;</pre> </li> <li>• <b>Constants:</b><br/> <pre>const int s=12;</pre> </li> <li>• <b>Scope:</b><br/> <pre>int i; // i=j; forbidden! int j=2; i=j; // OK! if (j&gt;1) {     int k=3;     j=k; // OK! } //i=k; forbidden!</pre> </li> <li>• <b>Types:</b><br/> <pre>int i=3; double x=12.3; char c='A'; string s="hop"; bool t=true; float y=1.2f; unsigned int j=4; signed char d=-128; unsigned char d=25; complex&lt;double&gt;     z(2,3);</pre> </li> <li>• <b>Global variables:</b><br/> <pre>int n; const int m=12; void f() {     n=10; // OK     int i=m; // OK     ... }</pre> </li> <li>• <b>Conversion:</b><br/> <pre>int i=int(x),j; float x=float(i)/j;</pre> </li> <li>• <b>Stack/Heap</b></li> </ul> | <p><b>Keys</b></p> <ul style="list-style-type: none"> <li>• <b>Debug:</b> F5 </li> <li>• <b>Step over:</b> F10 </li> <li>• <b>Step inside:</b> F11 </li> <li>• <b>Indent:</b> Ctrl+A, Ctrl+I</li> <li>• <b>Switch source/header:</b> F4</li> <li>• <b>Switch decl./def.:</b> F2</li> <li>• <b>Step out:</b> Maj+F11 </li> </ul> <p><b>Functions</b></p> <ul style="list-style-type: none"> <li>• <b>Definition:</b><br/> <pre>int plus(int a,int b){     int c=a+b;     return c; } void display(int a) {     cout &lt;&lt; a &lt;&lt; endl; }</pre> </li> <li>• <b>Declaration:</b><br/> <pre>int plus(int a,int b);</pre> </li> <li>• <b>Return:</b><br/> <pre>int sign(double x) {     if (x&lt;0)         return -1;     if (x&gt;0)         return 1;     return 0; } void display(int x,              int y) {     if (x&lt;0    y&lt;0)         return;     if (x&gt;=w    y&gt;=h)         return;     DrawPoint(x,y,RED); }</pre> </li> <li>• <b>Call:</b><br/> <pre>int f(int a) { ... } int g() { ... } ... int i=f(2),j=g();</pre> </li> <li>• <b>References:</b><br/> <pre>void swap(int&amp; a,           int&amp; b){</pre> </li> </ul> | <pre>int tmp=a; a=b;b=tmp; } ... int x=3,y=2; swap(x,y);</pre> <ul style="list-style-type: none"> <li>• <b>Overload:</b><br/> <pre>int chance(int n); int chance(int a,            int b); double chance();</pre> </li> <li>• <b>Operators:</b><br/> <pre>vect operator+(     vect A,vect B) {     ... }</pre> </li> <li>• <b>Call stack</b></li> <li>• <b>Iterative/Recursive</b></li> </ul> <p><b>Arrays</b></p> <ul style="list-style-type: none"> <li>• <b>Definition:</b><br/> <pre>- double x[5],y[5];   for(int i=0;i&lt;5;i++)       y[i]=2*x[i]; - const int n=5;   int i[n],j[2*n];</pre> </li> <li>• <b>Initialization:</b><br/> <pre>int t[4]={1,2,3,4}; string s[2]={"ab","c"};</pre> </li> <li>• <b>Assignment:</b><br/> <pre>int s[3]={1,2,3},t[3]; for (int i=0;i&lt;3;i++)     t[i]=s[i];</pre> </li> <li>• <b>As parameter:</b><br/> <pre>- void init(int t[4]){     for(int i=0;i&lt;4;i++)         t[i]=0; } - void init(int t[],            int n) {     for(int i=0;i&lt;n;i++)         t[i]=0; }</pre> </li> </ul> |

| Reference Card (3/4)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Variable size:</p> <pre>int* t=new int[n]; ... delete[] t;</pre> <p><b>As argument:</b></p> <ul style="list-style-type: none"> <li>• <code>void f(int* t,int n) { t[i]=... }</code></li> <li>• <code>void alloc(int*&amp; t){ t=new int[n]; }</code></li> </ul> <p><b>2D:</b></p> <pre>int A[2][3]; A[i][j]=...; int A[2][3]=     {{1,2,3},{4,5,6}}; void f(int A[2][2]);</pre> <p><b>2D in 1D:</b></p> <pre>int A[2*3]; A[i+2*j]=...;</pre> <p><b>Variable size:</b></p> <pre>int *t,*s,n;</pre> | <ul style="list-style-type: none"> <li>• <b>Combinations:</b> <code>&amp;&amp;   </code></li> <li>• <code>if (i==0) j=1;</code></li> <li>• <code>if (i==0) j=1; else j=2;</code></li> <li>• <code>if (i==0) { j=1; k=2; }</code></li> <li>• <code>bool t=(i==0); if (t) j=1;</code></li> <li>• <code>switch (i) { case 1: ...; ...; break; case 2: ...; case 3: ...; break; default: ...; }</code></li> </ul>                                                                            | <ul style="list-style-type: none"> <li>• <code>#include &lt;string&gt; using namespace std; string s="hop"; char c=s[0]; int l=s.size();</code></li> <li>• <code>#include &lt;ctime&gt; s=double(clock()) /CLOCKS_PER_SEC;</code></li> <li>• <code>#include &lt;cmath&gt; double pi=M_PI;</code></li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <p><b>Structures</b></p> <ul style="list-style-type: none"> <li>• <code>struct Point { double x,y; Color c; }; ... Point a; a.x=2.3; a.y=3.4; a.c=RED; Point b={1,2.5,BLUE};</code></li> </ul>                                                                                                                                                                                                                                                                                                       | <p><b>Input/Output</b></p> <ul style="list-style-type: none"> <li>• <code>#include &lt;iostream&gt; using namespace std; ... cout &lt;&lt;"I="&lt;&lt;i&lt;&lt;endl; cin &gt;&gt; i &gt;&gt; j;</code></li> </ul>                                                                                                                                                                                                                                                                        | <p><b>Frequent errors</b></p> <ul style="list-style-type: none"> <li>• No definition of function inside a function!</li> <li>• <code>int q=r=4; // NO!</code></li> <li>• <code>if (i=2) // NO!</code><br/><code>if i==2 // NO!</code><br/><code>if (i==2) then // NO!</code></li> <li>• <code>for(int i=0,i&lt;100,i++) // NO!</code></li> <li>• <code>int f() {...} int i=f; // NO!</code></li> <li>• <code>double x=1/3; // NO!</code><br/><code>int i,j; x=i/j; // NO!</code><br/><code>x=double(i/j); //NO!</code></li> <li>• <code>double x[10],y[10]; for(int i=1;i&lt;=10;i++) y[i]=2*x[i];//NO</code></li> <li>• <code>int n=5; int t[n]; // NO</code></li> <li>• <code>int f()[4] { // NO! int t[4]; ... return t; // NO! }</code><br/><code>int t[4]; t=f();</code></li> <li>• <code>int s[3]={1,2,3},t[3]; t=s; // NO!</code></li> <li>• <code>int t[2]; t={1,2}; // NO!</code></li> <li>• <code>struct Point { double x,y; } // NO!</code></li> <li>• <code>Point a; a={1,2}; // NO!</code></li> <li>• <code>#include "tp.cpp"//NO</code></li> </ul> |
| <p><b>Separate compilation</b></p> <ul style="list-style-type: none"> <li>• <code>#include "vect.h", also in vect.cpp</code></li> <li>• Functions: declarations in <code>.h</code>, definitions in <code>.cpp</code></li> <li>• Types: definitions in <code>.h</code></li> <li>• Declare in <code>.h</code> only useful functions</li> <li>• <code>#pragma once</code> at beginning of header file</li> <li>• Do not cut too much...</li> </ul>                                                      | <p><b>Misc.</b></p> <ul style="list-style-type: none"> <li>• <code>i++; i--; i-=2; j+=3;</code></li> <li>• <code>j=i%n; // Modulo</code></li> <li>• <code>#include &lt;cstdlib&gt; ... i=rand() %n; x=rand() / double(RAND_MAX);</code></li> <li>• <code>#include &lt;ctime&gt; // Single call srand((unsigned int) time(0));</code></li> <li>• <code>#include &lt;cmath&gt; double sqrt(double x); double cos(double x); double sin(double x); double acos(double x);</code></li> </ul> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <p><b>Tests</b></p> <ul style="list-style-type: none"> <li>• Comparison:<br/><code>== != &lt; &gt; &lt;= &gt;=</code></li> <li>• Negation: <code>!</code></li> </ul>                                                                                                                                                                                                                                                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

**Reference Card (4/4)**

```

int f(int t[][]); //NO
int t[2,3]; // NO!
t[i,j]=...; // NO!

int* t;
t[1]=...; // NO!

int* t=new int[2];
int* s=new int[2];
s=t; // lost s!
delete[] t;
delete[] s; //Crash!

int *t,s; // s is int
 // not int*
t=new int[n];
s=new int[n]; // NO!

```

**Imagine++**

- See documentation...

**Advice**

- Errors/warnings: click.
- Indent!
- Fix warnings.
- Use the debugger.
- Write functions.
- Arrays: not to translate math formula!

- Make structures.
- Do separate source files
- The .h must be enough for the user (who must not look into .cpp)
- Don't abuse recursion.
- Don't forget delete.
- Compile regularly.
- #include <cassert>
 

```

...
assert(x!=0);
y=1/x;

```

# Chapter 9

## First objects

We begin now our last step in the quest for a better organization of the code. We already structured instructions (functions, files) and data (structures, arrays). We are going now to think about data and instructions together: it is the first idea of objects, even though they have many other aspects.<sup>1</sup> Finally, we are going to justify the usage of “interface”.<sup>2</sup>

—

### 9.1 Philosophy

Uniting instructions in functions or files is a good idea. Uniting data in arrays or structures also. Often both are linked. It is what we saw naturally in the examples of previous chapters, in which a file grouped a structure and several related functions. It is in this case that we use **objects**.

The idea is natural: an object is a data type with some specific functionalities.<sup>3</sup> So that:

**It is not functions that work on data anymore. Data come with some functionalities.**

Such “functionalities” are often called the **methods** of the object. In practice, the usage of an object will replace this kind of instructions:

```
obj a;
int i=f(a); // function f() applied to a
```

by:

```
obj a;
int i=a.f(); // call method f() of a
```

You’ve got it, it is all about “**inserting**” functions in objects. But first a big warning:

---

<sup>1</sup>The most important one is inheritance, which we will not see in this course, putting the preference on other aspects of C++ more necessary and neglected until now. . .

<sup>2</sup>We will present an easy way to create interfaces. An experienced C++ programmer will use rather *inheritance* and *pure virtual functions*, which goes well beyond this course.

<sup>3</sup>It can even happen that an object has functionalities without storing any data. We will not use this way of presenting things, which the beginner may quickly abuse.

**Do not abuse objects, especially as a beginner. Dangers are indeed:**

- to see objects where there is none. Instructions and data are not always linked.
- to organize badly the data or instructions in objects.

**An advice thus: when it becomes too complex for you, drop the objects.**

Which does not mean that a beginner should avoid objects. Some small ones in simple cases are perfectly fine. Only experience allows organizing correctly the program, with the right objects, the right functions, etc. A simple example: when a function works on two data types, the beginner would often strive to put it as a method of one of the objects, and transform:

```
obj1 a;
obj2 b;
int i=f(a,b); // f() applied to a and b
```

to:

```
obj1 a;
obj2 b;
int i=a.f(b); // method f() of a applied to b
 // Is is the right thing to do?
```

Only some hindsight and experience allow to keep it simple when required. The first code was more logical: function `f()` is at home neither in `a` nor in `b`.

## 9.2 Simple example

It should be clear from previous examples, methods of objects are considered as part of the object type, in the same manner as its fields. Besides, the fields of an object are often called *members* of the object, and its methods the *member functions*. Here is the C++ syntax:

```
struct obj {
 int x; // field x
 int f(); // method f()
 int g(int y); // method g()
};
...
int main() {
 obj a;
 a.x=3;
 int i=a.f();
 int j=a.g(2);
 ...
}
```

There is just one detail but important: the definition of the structure `obj` above only *declares the methods*. They are *defined* nowhere in the preceding code. To define them, we do as for usual functions except

**to allow several objects to have the same method names, we prefix their definition by the name of the object followed by ::<sup>a</sup>**

<sup>a</sup>This mechanism exists also for usual functions. They are the namespaces, that we met and went around immediately with some `using namespace std` to avoid writing `std::cout`

Here comes how it is written:

```

struct obj1 {
 int x; // defined x
 int f(); // method f() (declaration)
 int g(int y); // method g() (declaration)
};
struct obj2 {
 double x; // field x
 double f(); // method f() (declaration)
};
...
int obj1::f() { // method f() of obj1 (definition)
 ...
 return ...
}
int obj1::g(int y) { // method g() of obj1 (definition)
 ...
 return ...
}
double obj2::f() { // method f() of obj2 (definition)
 ...
 return ...
}
...
int main() {
 obj1 a;
 obj2 b;
 a.x=3; // field x of a is int
 b.x=3.5; // field x of b is double
 int i=a.f(); // method f() of a (thus obj1::f())
 int j=a.g(2); // method g() of a (thus obj1::g())
 double y=b.f(); // method f() of b (thus obj2::f())
 ...
}

```

## 9.3 Visibility

There is a rule that we did not see on namespaces but that we can easily understand: when we are “in” a namespace, one can use all variables and functions of the namespace without precising the namespace. So that those who have programmed `cout` and `endl` have defined the namespace `std` then “went inside” to program without having to prefix everything with `std::` in front of `cout`, `cin`, `endl` and others... Following the same logic,

**in its methods, an object accesses directly its fields and methods, without prefix!<sup>a</sup>**

<sup>a</sup>You may see sometimes the keyword `this` that is useful in C++ and that programmers coming from Java or Python (called “self”) put everywhere, besides mistaking its type. You should not need it for the time being.

For example, the function `obj1::f()` above could be written:

```

1 int obj1::f() { // method f() of obj1 (definition)
2 int i=g(3); // method g() of object whose method f() is
3 // running
4 int j=x+i; // field x of object whose method f() is
5 // running
6 return j;
7 }
8 ...
9 int main() {
10 obj1 a1,a2;
11 int i1=a1.f(); // This call will use a1.g() line 2
12 // and a1.x line 4
13 int i2=a2.f(); // This call will use a2.g() line 2
14 // and a2.x line 4

```

It is rather natural that an object accesses simply its fields from its methods since

**if an object does not use its fields in a method, it means likely that we are putting in the object a function that has nothing to do with it (see abuse mentioned above).**

## 9.4 Example with matrices

When teaching programming, an example of source is worth more than the talk. If up to now we were in the blur, things should get clearer! Here is how our example of Chapter 8 looks like with objects:

```

#include <iostream>
#include <string>
using namespace std;

//=====
// functions on matrices
// could be in matrix.h and matrix.cpp

// ===== declarations (in .h)
struct Matrix {
 int m,n;
 double* t;
 void create(int m1,int n1);
 void destroy();
 double get(int i,int j);

```



```

 void set(int i,int j,double x);
 void display(string s);
};
Matrix operator*(Matrix A,Matrix B);

// ===== definitions (in .cpp)
void Matrix::create(int m1,int n1) {
 // Note the parameters are not called m and n
 // to avoid confusion with fields!
 m=m1;
 n=n1;
 t=new double[m*n];
}

void Matrix::destroy() {
 delete[] t;
}

double Matrix::get(int i,int j) {
 return t[i+m*j];
}

void Matrix::set(int i,int j,double x) {
 t[i+m*j]=x;
}

void Matrix::display(string s) {
 cout << s << "␣=" << endl;
 for(int i=0; i<m; i++) {
 for(int j=0; j<n; j++)
 cout << get(i,j) << "␣";
 cout << endl;
 }
}

Matrix operator*(Matrix A,Matrix B) {
 if(A.n!=B.m) {
 cout << "Error!" << endl;
 exit(1);
 }
 Matrix C;
 C.create(A.m,B.n);
 for(int i=0; i<A.m; i++)
 for(int j=0; j<B.n; j++) {
 // Cij=Ai0*B0j+Ai1*B1j+...
 C.set(i,j,0);
 for(int k=0; k<A.n; k++)
 C.set(i,j,

```

```

 C.get(i , j)+A.get(i ,k)*B.get(k , j));
 }
 return C;
}

// ===== main =====
int main()
{
 Matrix A;
 A.create(2 ,3);
 for(int i=0; i<2; i++)
 for(int j=0; j<3; j++)
 A.set(i ,j , i+j);
 A.display("A");
 Matrix B;
 B.create(3 ,5);
 for(int i=0; i<3; i++)
 for(int j=0; j<5; j++)
 B.set(i ,j , i+j);
 B.display("B");
 Matrix C=A*B;
 C.display("C");
 C.destroy();
 B.destroy();
 A.destroy();
 return 0;
}

```

## 9.5 Case of operators

It may seem a pity that operator `*` is not in object `Matrix`. To remedy it, the following convention is used:

**Let `A` be an object. If it has a method `operatorop(objB B)`, then `AopB` calls this method for any `B` of type `objB`.**

To be clear, the program:

```

struct objA {
 ...
};
struct objB {
 ...
};
int operator+(objA A, objB B) {
 ...
}
...

```

```
int main() {
 objA A;
 objB B;
 int i=A+B; // calls operator+(A,B)
 ...

```

could also be written

```
struct objA {
 ...
 int operator+(objB B);
};
struct objB {
 ...
};
int objA::operator+(objB B) {
 ...
}
...
int main() {
 objA A;
 objB B;
 int i=A+B; // calls A.operator+(B)
 ...

```

which for matrices gives:

```
struct Matrix {
 ...
 Matrix operator*(Matrix B);
};
...
// A*B calls A.operator*(B) thus all
// fields and functions used directly
// concern what was previously prefixed by A.
Matrix Matrix::operator*(Matrix B) {
 // we are in object A of the call A*B
 if(n!=B.m) { // n of A
 cout << "Error!" << endl;
 exit(1);
 }
 Matrix C;
 C.create(m,B.n);
 for(int i=0; i<m; i++)
 for(int j=0; j<B.n; j++) {
 // Cij=Ai0*B0j+Ai1*B1j+...
 C.set(i,j,0);
 for(int k=0; k<n; k++)
 // get(i,j) is the one of A
 C.set(i,j,
 C.get(i,j)+get(i,k)*B.get(k,j));
 }
}

```

```

 }
 return C;
}

```

Note also that the parameter of the operator needs not be an object. To write the product  $B=A*2$ , we will use the method

```

Matrix Matrix::operator*(double lambda) {
 ...
}
...
B=A*2; // Calls A.operator*(2)

```

On the contrary, to write  $B=2*A$ , it is not possible to use:

```

Matrix double::operator*(Matrix A) // IMPOSSIBLE as type double
// is not an object!

```

since it would amount to define a method for type `double`, that is not an object.<sup>4</sup> We need to simply use a standard operator, which would be well inspired by calling the method `Matrix::operator*(double lambda)` if it already exists:

```

Matrix operator*(double lambda, Matrix A) {
 return A*lambda; // defined previously, nothing more to do!
}
...
B=2*A; // calls operator*(2,A) that calls in turn
// A.operator*(2)

```

We will see in the next chapter other useful operators for objects...

## 9.6 Interface

If we examine `main()` of the matrix example, we can observe that it does not use any field of `Matrix` but only the methods. Actually, only the part

```

struct Matrix {
 void create(int m1, int n1);
 void destroy();
 double get(int i, int j);
 void set(int i, int j, double x);
 void display(string s);
 Matrix operator*(Matrix B);
};

```

is useful to the user. That the dimensions are stored in fields `int m` and `int n` and the elements in field `double* t` do not matter to the user: it concerns only the programmer of matrices. If the latter finds another way<sup>5</sup> to store the bidimensional array of `double`, said programmer is free to do it. Actually

<sup>4</sup>and anyway does not belong to the programmer!

<sup>5</sup>And there are some! For example to store efficiently sparse matrices, that is, whose most elements are null. Or also, using some objects implementing arrays efficiently and securely.

If the user of `Matrix` respects the declarations of the defined methods, their designer can program them freely. They can even be reprogrammed in another manner: the user's programs will still work! It is precisely the concept of an *interface*:

- The designer and the user agree on the methods that should exist.
- The designer implements them.<sup>a</sup>
- The user uses (!) them.
- The designer can improve them without bothering the user.

In particular the header file of the object is the only one that is useful to the user. It is its role to precise the interface, without going into implementation details. Hence, *linked only by interface, usage and implementation become independent.*<sup>b</sup>

<sup>a</sup>And has to choose the right implementation: some ways of storing data can make some methods efficient but at the expense of others, or use more memory, etc. The algorithmic problems must be handled. That is why also in general that for a same interface, the user may prefer such implementation instead of another: the designer is subject to concurrency!

<sup>b</sup>In any case, both are winners: the builder can improve the implementation with bothering the user, who has also the possibility to change to a concurrent implementation with modifying much its program.

## 9.7 Protection

### 9.7.1 Principle

All this is quite interesting, you may say, but implementation details are not hidden: the definition of the structure in the header file shows the fields used by the implementation. Hence, the user may be tempted to use them! Nothing prevents the user to make mistakes:

```
Matrix A;
A.create(3,2);
A.m=4; // Ouch! Some accesses will be wrong!
```

or simply not bothering by replacing

```
for (int i=0;i<3;i++)
 for (int j=0;j<2;j++)
 A.set(i,j,0);
```

by

```
for (int i=0;i<6;i++)
 A.t[i]=0; // No! And if implemented otherwise?
```

In this case, the usage is not any more independent on implementation and we lost a large share of the interest of object programming... Here appears the possibility to prevent the user from accessing some fields of even certain methods. For that:

1. Replace **struct** by the new keyword **class**: all fields and methods become *private*: only methods of the object itself or other objects of the same type<sup>a</sup> can use them.
2. Put the declaration **public**: in the object definition to define the zone<sup>b</sup> from which are declared fields and methods that are *public*, that is, freely accessible to all.

<sup>a</sup>To sum up, only the methods of the class!

<sup>b</sup>We could again declare private zones with `private`;, then another public zone, etc. There could even exist *protected* zones, used only for inheritance and not our concern in this course...

Here is an example:

```
class obj {
 int x,y;
 void mine();
public:
 int z;
 void for_all();
 void another(obj A);
};
void obj::mine() {
 x=..; // OK
 ..=y; // OK
 z=..; // OK
}
void obj::for_all() {
 x=..; // OK
 mine(); // OK
}
void obj::another(obj A) {
 x=A.x; // OK
 A.mine(); // OK
}
...
int main() {
 obj A,B;
 A.x=..; // NO!
 A.z=..; // OK
 A.mine(); // NO!
 A.for_all(); // OK
 A.another(B); // OK
}
```

In the case of our matrices, that we already implemented, it is enough to define them as:

```
class Matrix {
 int m,n;
 double* t;
```

```
public:
 void create(int m1, int n1);
 void destroy();
 double get(int i, int j);
 void set(int i, int j, double x);
 void display(string s);
 Matrix operator*(Matrix B);
};
```

to prevent their usage being dependent on their implementation.

## 9.7.2 Structures vs Classes

Note that finally a structure is a class where everything is public... Old C programmers believe wrongly that structures in C++ are the same as in C, that is that they are not objects and that they have no method.<sup>6</sup>

## 9.7.3 Accessors

Methods `get()` and `set()` that allow reading and writing access to our class, are called *accessors*. Now that our fields are private, the user has no way to retrieve the dimensions of a matrix. We thus add two accessors (here only *getters*) for reading:

```
int Matrice::nbRow() {
 return m;
}
int Matrice::nbCol() {
 return n;
}
int main() {
 ...
 for(int i=0; i<A.nbRow(); i++)
 for(int j=0; j<A.nbCol(); j++)
 A.set(i, j, 0);
```

but not for writing, which makes sense with the fact that changing  $m$  would make mistakes with functions using `t[i+m*j]`!

## 9.8 Practical

You are ready for Practical [A.6](#) that draws some fractal curves (Figure [9.1](#)) while illustrating the concept of object.

---

<sup>6</sup>Without mentioning that they declare them as in C with useless `typedef`. Anyway, it is not your concern!

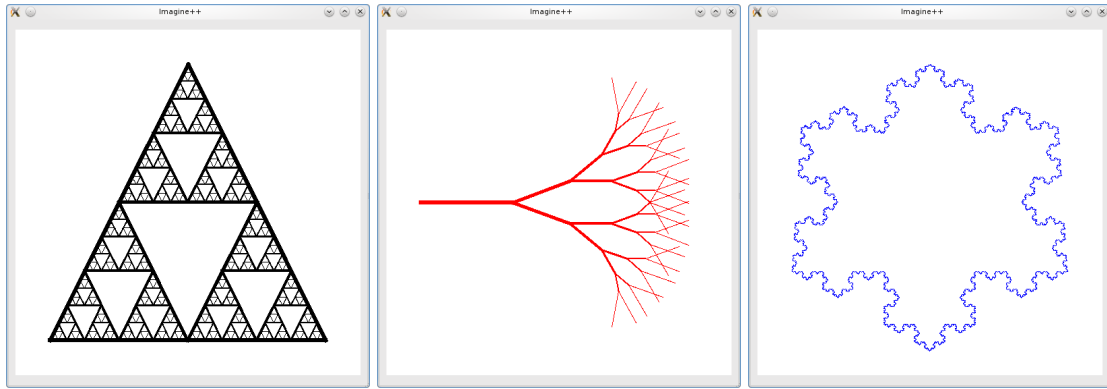


Figure 9.1: Fractals



## 9.9 Reference card

| Reference Card (1/3)                                                                                                                                                                                                                                                                                                                                                                                                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Loops</b></p> <ul style="list-style-type: none"> <li>do {<br/>  ...<br/>} while(!ok);</li> <li>int i=1;<br/>while(i&lt;=100) {<br/>  ...<br/>  i=i+1;<br/>}</li> <li>for(int i=1;i&lt;=10;i++)<br/>  ...</li> <li>for(int i=1,j=10;j&gt;i;<br/>  i=i+2,j=j-3)<br/>  ...</li> <li>for (int i=...)<br/>  for (int j=...) {<br/>    //skip case i==j<br/>    if (i==j)<br/>      continue;<br/>    ...<br/>  }</li> </ul> | <ul style="list-style-type: none"> <li><b>Constants:</b><br/>const int s=12;</li> <li><b>Scope:</b><br/>int i;<br/>// i=j; forbidden!<br/>int j=2;<br/>i=j; // OK!<br/>if (j&gt;1) {<br/>  int k=3;<br/>  j=k; // OK!<br/>}</li> <li><b>Types:</b><br/>int i=3;<br/>double x=12.3;<br/>char c='A';<br/>string s="hop";<br/>bool t=true;<br/>float y=1.2f;<br/>unsigned int j=4;<br/>signed char d=-128;<br/>unsigned char d=25;<br/>complex&lt;double&gt;<br/>  z(2,3);</li> <li><b>Global variables:</b><br/>int n;<br/>const int m=12;<br/>void f() {<br/>  n=10; // OK<br/>  int i=m; // OK<br/>  ...<br/>}</li> <li><b>Conversion:</b><br/>int i=int(x),j;<br/>float x=float(i)/j;</li> <li><b>Stack/Heap</b></li> </ul> | <ul style="list-style-type: none"> <li><b>Declaration:</b><br/>int plus(int a,int b);</li> <li><b>Return:</b><br/>int sign(double x) {<br/>  if (x&lt;0)<br/>    return -1;<br/>  if (x&gt;0)<br/>    return 1;<br/>  return 0;<br/>}<br/>void display(int x,<br/>              int y) {<br/>  if (x&lt;0    y&lt;0)<br/>    return;<br/>  if (x&gt;w    y&gt;h)<br/>    return;<br/>  DrawPoint(x,y,RED);<br/>}</li> <li><b>Call:</b><br/>int f(int a) { ... }<br/>int g() { ... }<br/>...<br/>int i=f(2),j=g();</li> <li><b>References:</b><br/>void swap(int&amp; a,<br/>          int&amp; b){<br/>  int tmp=a;<br/>  a=b;b=tmp;<br/>}</li> <li><b>Overload:</b><br/>int chance(int n);<br/>int chance(int a,<br/>           int b);<br/>double chance();</li> <li><b>Operators:</b><br/>vect operator+(<br/>  vect A,vect B) {<br/>  ...<br/>}</li> <li><b>Call stack</b></li> <li><b>Iterative/Recursive</b></li> </ul> |
| <p><b>Keys</b></p> <ul style="list-style-type: none"> <li>Debug: F5 </li> <li>Step over: F10 </li> <li>Step inside: F11 </li> <li>Indent: Ctrl+A, Ctrl+I</li> <li>Switch source/header: F4</li> <li>Swith decl./def.: F2</li> <li>Step out: Maj+F11 </li> <li>Gest. tâches: Ctrl+Maj+Ech</li> </ul>                                                                                                                          | <p><b>Variables</b></p> <ul style="list-style-type: none"> <li><b>Definition:</b><br/>int i;<br/>int k,l,m;</li> <li><b>Assignment:</b><br/>i=2;<br/>j=i;<br/>k=l=3;</li> <li><b>Initialization:</b><br/>int n=5,o=n;</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | <p><b>Functions</b></p> <ul style="list-style-type: none"> <li><b>Definition:</b><br/>int plus(int a,int b){<br/>  int c=a+b;<br/>  return c;<br/>}<br/>void display(int a) {<br/>  cout &lt;&lt; a &lt;&lt; endl;<br/>}</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

| Reference Card (2/3)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Arrays</b></p> <ul style="list-style-type: none"> <li>• <b>Definition:</b> <ul style="list-style-type: none"> <li>- <code>double x[5],y[5];</code><br/> <code>for(int i=0;i&lt;5;i++)</code><br/> <code>  y[i]=2*x[i];</code></li> <li>- <code>const int n=5;</code><br/> <code>int i[n],j[2*n];</code></li> </ul> </li> <li>• <b>Initialization:</b><br/> <code>int t[4]={1,2,3,4};</code><br/> <code>string s[2]={"ab","c"};</code></li> <li>• <b>Assignment:</b><br/> <code>int s[3]={1,2,3},t[3];</code><br/> <code>for (int i=0;i&lt;3;i++)</code><br/> <code>  t[i]=s[i];</code></li> <li>• <b>As parameter:</b> <ul style="list-style-type: none"> <li>- <code>void init(int t[4]){</code><br/> <code>  for(int i=0;i&lt;4;i++)</code><br/> <code>    t[i]=0;</code><br/> <code>  }</code></li> <li>- <code>void init(int t[],</code><br/> <code>          int n) {</code><br/> <code>  for(int i=0;i&lt;n;i++)</code><br/> <code>    t[i]=0;</code><br/> <code>  }</code></li> </ul> </li> <li>• <b>Variable size:</b><br/> <code>int* t=new int[n];</code><br/> <code>...</code><br/> <code>delete[] t;</code></li> <li>• <b>As argument:</b> <ul style="list-style-type: none"> <li>- <code>void f(int* t,int n)</code><br/> <code>  { t[i]=... }</code></li> <li>- <code>void alloc(int*&amp; t){</code><br/> <code>  t=new int[n];</code><br/> <code>  }</code></li> </ul> </li> <li>• <b>2D:</b><br/> <code>int A[2][3];</code><br/> <code>A[i][j]=...;</code><br/> <code>int A[2][3]=</code></li> </ul> | <pre>       {{1,2,3},{4,5,6}}; void f(int A[2][2]); </pre> <ul style="list-style-type: none"> <li>• <b>2D in 1D:</b><br/> <code>int A[2*3];</code><br/> <code>A[i+2*j]=...;</code></li> <li>• <b>Variable size:</b><br/> <code>int *t,*s,n;</code></li> </ul> <hr/> <p><b>Structures</b></p> <ul style="list-style-type: none"> <li>• <code>struct Point {</code><br/> <code>  double x,y;</code><br/> <code>  Color c;</code><br/> <code>};</code><br/> <code>...</code><br/> <code>Point a;</code><br/> <code>a.x=2.3; a.y=3.4;</code><br/> <code>a.c=RED;</code><br/> <code>Point b={1,2.5,BLUE};</code></li> <li>• <b>A structure is an objet fully public (→ see objects!)</b></li> </ul> <hr/> <p><b>Objects</b></p> <ul style="list-style-type: none"> <li>• <code>struct obj {</code><br/> <code>  int x; // field</code><br/> <code>  int f(); // method</code><br/> <code>  int g(int y);</code><br/> <code>};</code><br/> <code>int obj::f() {</code><br/> <code>  int i=g(3); // my g</code><br/> <code>  int j=x+i; // my x</code><br/> <code>  return j;</code><br/> <code>}</code><br/> <code>...</code><br/> <code>int main() {</code><br/> <code>  obj a;</code><br/> <code>  a.x=3;</code><br/> <code>  int i=a.f();</code><br/> <code>}</code></li> <li>• <code>class obj {</code><br/> <code>  int x,y;</code><br/> <code>  void mine();</code><br/> <code>public:</code><br/> <code>  int z;</code><br/> <code>  void for_all();</code><br/> <code>}</code></li> </ul> | <pre>   void another(obj A); }; void obj::mine() {   x=..; // OK   ..=y; // OK   z=..; // OK } void obj::for_all(){   x=..; // OK   mine(); // OK } void another(obj A) {   x=A.x; // OK   A.mine(); // OK } ... int main() {   obj A,B;   A.x=..; //NO   A.z=..; //OK   A.mine(); //NO   A.for_all(); //OK   A.another(B); //OK } </pre> <ul style="list-style-type: none"> <li>• <code>class obj {</code><br/> <code>  obj operator+(obj B);</code><br/> <code>};</code><br/> <code>...</code><br/> <code>int main() {</code><br/> <code>  obj A,B,C;</code><br/> <code>  C=A+B;</code><br/> <code>  // C=A.operator+(B)</code><br/> <code>}</code></li> </ul> <hr/> <p><b>Separate compilation</b></p> <ul style="list-style-type: none"> <li>• <code>#include "vect.h",</code> also in <code>vect.cpp</code></li> <li>• <b>Functions:</b> declarations in <code>.h</code>, definitions in <code>.cpp</code></li> <li>• <b>Types:</b> definitions in <code>.h</code></li> <li>• <b>Declare in .h</b> only useful functions</li> <li>• <code>#pragma once</code> at beginning of header file</li> <li>• <b>Do not cut too much...</b></li> </ul> |

## Reference Card (3/3)

## Tests

- Comparison:  
== != < > <= >=
- Negation: !
- Combinations: && ||
- if (i==0) j=1;
- if (i==0) j=1;  
else j=2;
- if (i==0) {  
j=1;  
k=2;  
}
- bool t=(i==0);  
if (t)  
j=1;
- switch (i) {  
case 1:  
...;  
...;  
break;  
case 2:  
case 3:  
...;  
break;  
default:  
...;  
}

## Input/Output

- #include <iostream>  
using namespace std;  
...  
cout <<"I="<<i<<endl;  
cin >> i >> j;

## Frequent errors

- No definition of function inside a function!
- int q=r=4; // NO!
- if (i=2) // NO!  
if i==2 // NO!  
if (i==2) then // NO!
- for(int i=0,i<100,i++)  
// NO!
- int f() {...}  
int i=f; // NO!
- double x=1/3; // NO!  
int i,j;  
x=i/j; // NO!  
x=double(i/j); //NO!

- double x[10],y[10];  
for(int i=1;i<=10;i++)  
y[i]=2\*x[i];//NO
- int n=5;  
int t[n]; // NO
- int f()[4] { // NO!  
int t[4];  
...  
return t; // NO!  
}  
int t[4]; t=f();
- int s[3]={1,2,3},t[3];  
t=s; // NO!
- int t[2];  
t={1,2}; // NO!
- struct Point {  
double x,y;  
} // NO!
- Point a;  
a={1,2}; // NO!
- #include "tp.cpp"//NO
- int f(int t[][]);//NO  
int t[2,3]; // NO!  
t[i,j]=...; // NO!
- int\* t;  
t[1]=...; // NO!
- int\* t=new int[2];  
int\* s=new int[2];  
s=t; // lost s!  
delete[] t;  
delete[] s;//Crash!
- int \*t,s;// s is int  
// not int\*  
t=new int[n];  
s=new int[n];// NO!

## Misc.

- i++;  
i--;  
i-=2;  
j+=3;
- j=i%n; // Modulo
- #include <cstdlib>  
...  
i=rand()%n;  
x=rand()/  
double(RAND\_MAX);
- #include <ctime>  
// Single call  
srand((unsigned int)  
time(0));

- #include <cmath>  
double sqrt(double x);  
double cos(double x);  
double sin(double x);  
double acos(double x);
- #include <string>  
using namespace std;  
string s="hop";  
char c=s[0];  
int l=s.size();
- #include <ctime>  
s=double(clock())  
/CLOCKS\_PER\_SEC;
- #include <cmath>  
double pi=M\_PI;

## Imagine++

- See documentation...

## Advice

- Errors/warnings: click.
- Indent!
- Fix warnings.
- Use the debugger.
- Write functions.
- Arrays: not to translate math formula!
- Make structures.
- Do separate source files
- The .h must be enough for the user (who must not look into .cpp)
- Don't abuse recursion.
- Don't forget delete.
- Compile regularly.
- #include <cassert>  
...  
assert(x!=0);  
y=1/x;
- **Make objects.**
- **Do not always make objects!**
- **Think interface / implementation / usage.**



# Chapter 10

## Constructors

*In this important chapter, we will see how C++ allows controlling what happens when a new object is created. This fundamental mechanism relies on the notion of **constructor**. This is very useful, even for the beginner who should be able to know at least its simplest form. We will also discover something fairly useful, as much for the efficiency of programs as for the discovery of bugs at compilation time: a new usage of **const**. At this point, we will be only halfway in the lifespan of objects. We keep for next chapter the counterpart notion of destructor.*

—

### 10.1 The problem

With the appearance of objects, we transformed:

```
struct point {
 int x,y;
};
...
 point a;
 a.x=2;a.y=3;
 i=a.x;j=a.y;
```

in:

```
class point {
 int x,y;
public:
 void get(int&X, int&Y);
 void set(int X,int Y);
};
...
 point a;
 a.set(2,3);
 a.get(i,j);
```

As a consequence:

```
 point a={2,3};
```

does not work anymore because in a sense it allows assigning private members of an object.<sup>1</sup>

## 10.2 The solution

The solution is the notion of **constructor**:

```
class point {
 int x,y;
public:
 point(int X,int Y);
};
point::point(int X,int Y) {
 x=X;
 y=Y;
}
...
point a(2,3);
```

**A constructor is a method whose name is the name of the class itself. It returns nothing, but its return type is not even mentioned as `void`. The constructor is called at object creation and its arguments are passed with the syntax above. It is impossible to call a constructor on an already created object.**

Here, it is the constructor `point::point(int X,int Y)` that is defined. Let us repeat, it is impossible to call a constructor on an object already existing:

```
point a(1,2); // OK! Values initialized
// We cannot change fields like this:
a.point(3,4); // ERROR!
// But like that:
a.set(3,4); // OK!
```

Let us notice that since the standard 2011 of C++,<sup>2</sup> *uniform initialization* was introduced, which allows to write the equivalent forms that all call the constructor:

```
point a(1,2); // usual constructor call
point b={1,2}; // alternate syntax
point c{1,2}; // still alternate syntax
```

<sup>1</sup>Actually, there is another reason, more fundamental and too difficult to explain here, that makes such initialization impossible.

<sup>2</sup>Called C++11. It was the second standard norm for C++, the first one dating from 1998. Since C++11, a new standard was approved every 3 years (there was C++14, C++17, C++20), each adding new syntax and new functions in the standard library. The compiler obeys a standard default, which is at least C++11 nowadays.

## 10.3 General case

### 10.3.1 Empty constructor

When an object is created without precision, it is the *empty constructor*, or *default constructor*, that is called, the one without parameters. The program:

```
class obj {
public:
 obj();
};
obj::obj() {
 cout << "hello" << endl;
}
...
obj a; // calls the constructor without argument
```

displays "hello".

**The empty constructor `obj::obj()` is called each time an object is created without argument.**

To be more precise, consider the following program:

```
#include <iostream>
using namespace std;

class obj {
public:
 obj();
};

obj::obj() {
 cout << "obj_";
}

void f(obj d) {
}

obj g() {
 obj e;
 cout << 6 << "_";
 return e;
}

int main()
{
 cout << 0 << "_";
 obj a;
 cout << 1 << "_";
 for(int i=2; i<=4; i++) {
```

```

 obj b;
 cout << i << " ";
 }
 f(a);
 cout << 5 << " ";
 a=g();
 return 0;
}

```

It displays:

```
0 obj 1 obj 2 obj 3 obj 4 5 obj 6
```

Observe how each call to `obj::obj()` can be spotted by the display. However, the parameter  $d$  of `f()`, copy of argument  $a$ , does *not* call the constructor; more precisely, it calls the *copy constructor* (which we did not define), and we will learn about it in next chapter. The situation for the return value of `g()`, object  $e$ , is also a bit complicated,<sup>3</sup> we leave it also for next chapter.

### 10.3.2 Several constructors

It is perfectly fine for an object to have several constructors:

```

class point {
 int x,y;
public:
 point(int X,int Y);
 point(int V);
};
point::point(int X,int Y) {
 x=X;
 y=Y;
}
point::point(int V) {
 x=y=V;
}
...
point a(2,3); // built with point(X,Y)
point b(4); // built with point(V)

```

As an alternative to the last syntax, for a constructor with a single parameter, it is possible to write:

```
point b = 4; // Equivalent to point b(4)
```

It is still important to remember this:

**If no constructor is defined, everything happens as if there were only an empty constructor, that does nothing special. But: as soon as we define a constructor, the empty constructor does not exist anymore, except if we define it explicitly.**

<sup>3</sup>For the time being, just be confident that it works as expected.



For example, the program:

```
class point {
 int x,y;
 void set(...)
};
...
point a;
a.set(2,3);
point b; // OK
```

becomes, with a constructor, a program not compiling:

```
class point {
 int x,y;
public:
 point(int X,int Y);
};
point::point(int X,int Y) {
 x=X;
 y=Y;
}
...
point a(2,3); // built with point(X,Y)
point b; // ERROR! point() does not exist any more
```

and we need to append an empty constructor, even if it does nothing:

```
class point {
 int x,y;
public:
 point();
 point(int X,int Y);
};
point::point() {
 // Better to initialize x,y, but free to do nothing!
}
point::point(int X,int Y) {
 x=X;
 y=Y;
}
...
point a(2,3); // built with point(X,Y)
point b; // OK! built with point()
```

### 10.3.3 Array of objects

There is no way to specify globally which constructor is called for the elements of an array. It is always the empty constructor that is called:

```
point t[3]; // built 3 times with empty constructor
```

```

 // on each element of array
point* s=new point[n]; // Idem, n times
point* u=new point(1,2)[n]; // ERROR and HORROR!
 // A trial to build u[i]
 // with point(1,2)

```

To achieve the same goal, we will have to write:

```

point* u=new point[n];
for (int i=0;i<n;i++)
 u[i].set(1,2);

```

which is not identical, since we build first the points with the empty constructor, then assign them.

Still, it is possible to write:

```

point t[3]={point(1,2),point(2,3),point(3,4)};

```

which is not feasible for variable size array.

As a consequence,

**To be able to have an array of objects, the object type must have an empty constructor, be it an explicit one, or the default one (if no constructor is defined).**

### 10.3.4 Field of object type

When a class contains a field of object type, the question arises: how is the field constructed? This is controlled by the user with the following syntax:

```

class Circle {
 point center;
 int radius;
public:
 Circle();
 Circle(int x, int y, int r);
};
Circle::Circle() { radius = 0; }
Circle::Circle(int x,int y,int r) : center(x,y), radius(r) {}

```

In the second constructor, the field `center` is constructed before the braces, calling the constructor `point::point(int, int)`. Actually, when reaching the opening brace, the object (hence its fields) is already *constructed*, the code between braces consisting of further *initialization* if necessary. To be consistent, the language allows also the initialization before the braces for the field `radius`, even though it does not have a constructor, since it is a basic type. For the first constructor, since no instruction indicates how to construct the field `center`, the empty constructor is called, `point::point()`, which implies that this constructor must exist for the code to build.

## 10.4 Temporary objects

We can, calling directly a constructor, build an object without storing it in a variable. Actually, it is a temporary object, without variable name, we call it anonymous.

The program:

```
void f(point p) {
 ...
}
point g() {
 point e(1,2); // to be returned
 return e;
}
...
point a(3,4); // only to call f()
f(a);
point b;
b=g();
point c(5,6); // we could want to do
b=c; // this to put b at (5,6)
```

can be made widely lighter, with no storage in variables the points for which it is not useful:

```
1 void f(point p) {
2 ...
3 }
4 point g() {
5 return point(1,2); // return anonymous object point(1,2)
6 }
7 ...
8 f(point(3,4)); // pass directly temporary object point(3,4)
9 point b;
10 b=g();
11 b=point(5,6); // assign b with temp. object point(5,6)
```

**Be attentive to line 11:** it is executed when `b` already exists but you need to understand that **we build an anonymous point(5,6) that is then assigned to `b`. We do not fill `b` directly with (5,6) as we would do with `b.set(5,6)`.**

Beware also on the **very frequent error** of creating a temporary object to create really a variable:

```
point p=point(1,2); // NO!
```

is uselessly complex and should be written:

```
point p(1,2); // YES!
```

The usefulness of temporary objects is more noticeable on a real example:

```
point point::operator+(point b) {
```

```

 point c(x+b.x,y+b.y);
 return c;
}
...
point a(1,2),b(2,3);
c=a+f(b);

```

would be replaced by:

```

point point::operator+(point b) {
 return point(x+b.x,y+b.y);
}
...
c=point(1,2)+f(point(2,3));

```

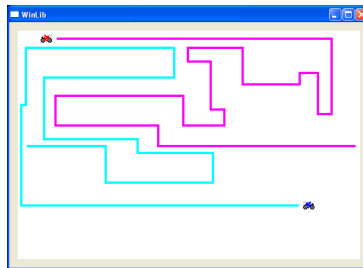


Figure 10.1: Game of Tron.

## 10.5 Practical

We can now begin the Practical proposed at [A.7](#). It is a game of Tron (single or two players), see [Figure 10.1](#). This mini-project will be completed with the contents of next chapter.

## 10.6 Constant References

### 10.6.1 Principle

When an object is passed as argument to a function, it is copied.<sup>4</sup> This copy is a source of lack of efficiency. For this reason, in the following program:

```

const int N=1000;
class vector {
 double t[N];
 ...
};
class matrix {
 double t[N][N];

```

<sup>4</sup>The precise mechanism of this copy will be studied in the next chapter.

```

...
};
// solve system of equation AX=B
void solve(matrix A,vector B,vector& X) {
...
}
...
vector b,x;
matrix a;
...
solve(a,b,x);

```

variables  $A$  and  $B$  of function `solve()` are copies of objects  $a$  and  $b$  of the calling scope. Note that, passed by reference, the parameter  $X$  is not a copy but just a link to the variable  $x$ .

The copy of  $a$  in  $A$  is not a good thing. The variable  $a$  weighs in our case 8 millions bytes: copying them in  $A$  takes time! Even for lighter objects, if a function is called often, this copy of argument may slow down the program. For a short function, the time of copy may even be more than the one spent in the function itself!

The idea is then, for heavy objects, to pass them also by reference to avoid the copy, even if the function does not intend to modify them! We just have to define the function `solve()` in this way:

```

void solve(matrix& A,vector& B,vector& X) {
...

```

and no copy of matrix happens.

However, this solution is not without danger. There is no guarantee that `solve` does not modify its parameters  $A$  and  $B$ . It is then possible, according to how `solve` is implemented, that at exit of `solve(a,b,x)`,  $a$  and  $b$  be modified themselves, whereas before it was the copies  $A$  and  $B$  that were modified. It is obviously a drawback! The C++ language offers fortunately the possibility of *asking the compiler to check that a variable passed by reference is not modified*. It is enough to add a `const` at the adequate place:

```

void solve(const matrix& A,const vector& B,vector& X) {
...

```

If anywhere in `solve` (or in functions called by `solve`!), the variable  $A$  or  $B$  is modified, the compiler would refuse it. The rule is thus:

**When a function argument `obj o` is of significant size<sup>a</sup>, it is a good idea to replace it by `const obj& o`.**

<sup>a</sup>Actually, the program is faster for most objects, whatever their size.

## 10.6.2 Constant methods

Let us consider the program:

```

void g(int& x) {
 cout << x << endl;

```

```

}
void f(const int& y) {
 double z=y; // OK does not modify y
 g(y); // OK?
}
...
int a=1;
f(a);

```

The function `f()` does not modify its parameter `y` and everything is fine. Now consider a second version of `g()`:

```

void g(int& x) {
 x++;
}

```

Then `y` would be modified in `f()` because of the call to `g()`. The program should not compile... Actually, the first version of `g()` would also be refused because

**to know whether a called function modifies or not one of its arguments passed by reference, the compiler relies only on its declaration and not its complete definition.<sup>a</sup>**

<sup>a</sup>The compiler does not try to guess if a function modifies its arguments since the logic is that the programmer indicates with a `const` what is intended; the compiler only checks a constant argument is not modified.

Indeed, our first program would not compile either since the call `g(y)` with `const int& y` imposes that `g()` be declared `void g(const int& x)`. The right program is thus:

```

void g(const int& x) {
 cout << x << endl;
}
void f(const int& y) {
 double z=y; // OK does not modify y
 g(y); // OK! No need to check inside g()
}
...
int a=1;
f(a);

```

(of course, it is a bit silly to pass an integer as `const int&`, while a simple `int` would do).

Replacing the `int` above by an object, we need a new notion. Consider now this:

```

void f(const obj& o) {
 o.g(); // OK?
}

```

We need to indicate to the compiler if the method `g()` modifies the object `o`. The following syntax is used:

```

class obj {
 ...

```

```
 void g() const;
 ...
};
void obj::g() const {
 ...
}
void f(const obj& o) {
 o.g(); // OK! Constant method
}
```

This is not so complicated:

**We indicate that a *method is constant*, that is, it does not modify its object, by putting `const` after its parentheses both in declaration and definition.**





We could wonder whether this is really necessary: our starting argument was just passing quickly arguments to functions by using references. In reality, the `const` in methods is a very good thing. Do not consider that as a hassle, but as a way of express precisely your thoughts: “am I adding a method that modifies the object?”. The compiler will then check for you the coherency of this `const` with the rest. It has two important effects:

- Early discovery of bugs, at compile time (we thought an object was not modified, it actually is).
- Optimization of the program.<sup>5</sup>

---

<sup>5</sup>When the compiler knows an object remains constant in part of the program, it can avoid rereading it each time. The `const` is then precious information for the optimizing engine of the compiler.

## 10.7 Reference card

| Reference Card (1/4)                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Loops</b></p> <ul style="list-style-type: none"> <li>do {<br/>    ...<br/>} while(!ok);</li> <li>int i=1;<br/>while(i&lt;=100) {<br/>    ...<br/>    i=i+1;<br/>}</li> <li>for(int i=1;i&lt;=10;i++)<br/>    ...</li> <li>for(int i=1,j=10;j&gt;i;<br/>    i=i+2,j=j-3)<br/>    ...</li> <li>for (int i=...)<br/>    for (int j=...) {<br/>        //skip case i==j<br/>        if (i==j)<br/>            continue;<br/>        ...<br/>    }</li> </ul>                               | <ul style="list-style-type: none"> <li>Step out: Maj+F11 </li> <li>Gest. tâches: Ctrl+Maj+Ech</li> </ul> <hr/> <p><b>Structures</b></p> <ul style="list-style-type: none"> <li>struct Point {<br/>    double x,y;<br/>    Color c;<br/>};<br/>...<br/>Point a;<br/>a.x=2.3; a.y=3.4;<br/>a.c=RED;<br/>Point b={1,2.5,BLUE};</li> <li>A structure is an objet fully public (→ see objects!)</li> </ul> <hr/> <p><b>Variables</b></p> <ul style="list-style-type: none"> <li>Definition:<br/>int i;<br/>int k,l,m;</li> <li>Assignment:<br/>i=2;<br/>j=i;<br/>k=l=3;</li> <li>Initialization:<br/>int n=5,o=n;</li> <li>Constants:<br/>const int s=12;</li> <li>Scope:<br/>int i;</li> </ul> | <pre>// i=j; forbidden!<br/>int j=2;<br/>i=j; // OK!<br/>if (j&gt;1) {<br/>    int k=3;<br/>    j=k; // OK!<br/>}<br/>//i=k; forbidden!</pre> <ul style="list-style-type: none"> <li>Types:<br/>int i=3;<br/>double x=12.3;<br/>char c='A';<br/>string s="hop";<br/>bool t=true;<br/>float y=1.2f;<br/>unsigned int j=4;<br/>signed char d=-128;<br/>unsigned char d=25;<br/>complex&lt;double&gt;<br/>    z(2,3);</li> <li>Global variables:<br/>int n;<br/>const int m=12;<br/>void f() {<br/>    n=10; // OK<br/>    int i=m; // OK<br/>    ...<br/>}</li> <li>Conversion:<br/>int i=int(x),j;<br/>float x=float(i)/j;</li> <li>Stack/Heap</li> </ul> |
| <p><b>Keys</b></p> <ul style="list-style-type: none"> <li>Debug: F5 </li> <li>Step over: F10 </li> <li>Step inside: F11 </li> <li>Indent: Ctrl+A, Ctrl+I</li> <li>Switch source/header: F4</li> <li>Swith decl./def.: F2</li> </ul> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |



| Reference Card (2/4)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Functions</b></p> <ul style="list-style-type: none"> <li>• <b>Definition:</b><br/> <pre>int plus(int a,int b){     int c=a+b;     return c; } void display(int a) {     cout &lt;&lt; a &lt;&lt; endl; }</pre> </li> <li>• <b>Declaration:</b><br/> <pre>int plus(int a,int b);</pre> </li> <li>• <b>Return:</b><br/> <pre>int sign(double x) {     if (x&lt;0)         return -1;     if (x&gt;0)         return 1;     return 0; } void display(int x,              int y) {     if (x&lt;0    y&lt;0)         return;     if (x&gt;=w    y&gt;=h)         return;     DrawPoint(x,y,RED); }</pre> </li> <li>• <b>Call:</b><br/> <pre>int f(int a) { ... } int g() { ... } ... int i=f(2),j=g();</pre> </li> <li>• <b>References:</b><br/> <pre>void swap(int&amp; a,           int&amp; b){     int tmp=a;     a=b;b=tmp; } ...</pre> </li> </ul> | <pre>int x=3,y=2; swap(x,y);</pre> <ul style="list-style-type: none"> <li>• <b>Overload:</b><br/> <pre>int chance(int n); int chance(int a,            int b); double chance();</pre> </li> <li>• <b>Operators:</b><br/> <pre>vect operator+(     vect A,vect B) {     ... } ... vect C=A+B;</pre> </li> <li>• <b>Call stack</b></li> <li>• <b>Iterative/Recursive</b></li> <li>• <b>Constant references (avoid copy):</b><br/> <pre>void f(const obj&amp; x) {     ... } void g(const obj&amp; x) {     f(x); // OK }</pre> </li> </ul> <hr/> <p><b>Arrays</b></p> <ul style="list-style-type: none"> <li>• <b>Definition:</b><br/> <pre>- double x[5],y[5]; for(int i=0;i&lt;5;i++)     y[i]=2*x[i];</pre> <pre>- const int n=5; int i[n],j[2*n];</pre> </li> <li>• <b>Initialization:</b><br/> <pre>int t[4]={1,2,3,4}; string s[2]={"ab","c"};</pre> </li> </ul> | <ul style="list-style-type: none"> <li>• <b>Assignment:</b><br/> <pre>int s[3]={1,2,3},t[3]; for (int i=0;i&lt;3;i++)     t[i]=s[i];</pre> </li> <li>• <b>As parameter:</b><br/> <pre>- void init(int t[4]){     for(int i=0;i&lt;4;i++)         t[i]=0; } - void init(int t[],            int n) {     for(int i=0;i&lt;n;i++)         t[i]=0; }</pre> </li> <li>• <b>Variable size:</b><br/> <pre>int* t=new int[n]; ... delete[] t;</pre> </li> <li>• <b>As argument:</b><br/> <pre>- void f(int* t,int n)     { t[i]=... } - void alloc(int*&amp; t){     t=new int[n]; }</pre> </li> <li>• <b>2D:</b><br/> <pre>int A[2][3]; A[i][j]=...; int A[2][3]=     {{1,2,3},{4,5,6}}; void f(int A[2][2]);</pre> </li> <li>• <b>2D in 1D:</b><br/> <pre>int A[2*3]; A[i+2*j]=...;</pre> </li> <li>• <b>Variable size:</b><br/> <pre>int *t,*s,n;</pre> </li> </ul> |

| Reference Card (3/4)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Objects</b></p> <ul style="list-style-type: none"> <li>• struct obj {<br/>  int x; // field<br/>  int f(); // method<br/>  int g(int y);<br/>};<br/>int obj::f() {<br/>  int i=g(3); // my g<br/>  int j=x+i; // my x<br/>  return j;<br/>}<br/>...<br/>int main() {<br/>  obj a;<br/>  a.x=3;<br/>  int i=a.f();<br/>}</li> <li>• class obj {<br/>  int x,y;<br/>  void mine();<br/>public:<br/>  int z;<br/>  void for_all();<br/>  void another(obj A);<br/>};<br/>void obj::mine() {<br/>  x=..; // OK<br/>  ..=y; // OK<br/>  z=..; // OK<br/>}<br/>void obj::for_all(){<br/>  x=..; // OK<br/>  mine(); // OK<br/>}<br/>void another(obj A) {<br/>  x=A.x; // OK<br/>  A.mine(); // OK<br/>}<br/>...<br/>int main() {<br/>  obj A,B;<br/>  A.x=..; //NO<br/>  A.z=..; //OK<br/>  A.mine(); //NO<br/>  A.for_all(); //OK<br/>  A.another(B); //OK<br/>}</li> <li>• class obj {<br/>  obj operator+(obj B);</li> </ul> | <pre>}; ... int main() {   obj A,B,C;   C=A+B;   // C=A.operator+(B) } </pre> <ul style="list-style-type: none"> <li>• <b>Constant methods:</b><br/>void obj::f() const{<br/>  ...<br/>}<br/>void g(const obj&amp; x){<br/>  x.f(); // OK<br/>}</li> <li>• <b>Constructor:</b><br/>class point {<br/>  int x,y;<br/>public:<br/>  point(int X,int Y);<br/>};<br/>point::point(int X,<br/>              int Y){<br/>  x=X;<br/>  y=Y;<br/>}<br/>...<br/>  point a(2,3);</li> <li>• <b>Empty constructor:</b><br/>obj::obj() {<br/>  ...<br/>}</li> <li>• <b>Temporary objects:</b><br/>vec vec::operator+(<br/>                    vec b) {<br/>  return vec(x+b.x,<br/>              y+b.y);<br/>}<br/>...<br/>  c=vec(1,2)<br/>      +f(vec(2,3));</li> </ul> <p><b>Separate compilation</b></p> <ul style="list-style-type: none"> <li>• #include "vect.h", also in vect.cpp</li> <li>• Functions: declarations in .h, definitions in .cpp</li> </ul> | <ul style="list-style-type: none"> <li>• Types: definitions in .h</li> <li>• Declare in .h only useful functions</li> <li>• #pragma once at beginning of header file</li> <li>• Do not cut too much...</li> </ul> <hr/> <p><b>Tests</b></p> <ul style="list-style-type: none"> <li>• Comparison:<br/>== != &lt; &gt; &lt;= &gt;=</li> <li>• Negation: !</li> <li>• Combinations: &amp;&amp;   </li> <li>• if (i==0) j=1;</li> <li>• if (i==0) j=1;<br/>  else j=2;</li> <li>• if (i==0) {<br/>  j=1;<br/>  k=2;<br/>}</li> <li>• bool t=(i==0);<br/>  if (t)<br/>    j=1;</li> <li>• switch (i) {<br/>  case 1:<br/>    ...;<br/>    ...;<br/>    break;<br/>  case 2:<br/>  case 3:<br/>    ...;<br/>    break;<br/>  default:<br/>    ...;<br/>}</li> </ul> <hr/> <p><b>Input/Output</b></p> <ul style="list-style-type: none"> <li>• #include &lt;iostream&gt;<br/>  using namespace std;<br/>  ...<br/>  cout &lt;&lt;"I="&lt;&lt;i&lt;&lt;endl;<br/>  cin &gt;&gt; i &gt;&gt; j;</li> </ul> |

| Reference Card (4/4)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Misc.</b></p> <ul style="list-style-type: none"> <li>• <code>i++;</code><br/><code>i--;</code><br/><code>i-=2;</code><br/><code>j+=3;</code></li> <li>• <code>j=i%n;</code> // Modulo</li> <li>• <code>#include &lt;cstdlib&gt;</code><br/>...<br/><code>i=rand() %n;</code><br/><code>x=rand() /</code><br/><code>double (RAND_MAX);</code></li> <li>• <code>#include &lt;ctime&gt;</code><br/>// Single call<br/><code>srand((unsigned int)</code><br/><code>time(0));</code></li> <li>• <code>#include &lt;cmath&gt;</code><br/><code>double sqrt(double x);</code><br/><code>double cos(double x);</code><br/><code>double sin(double x);</code><br/><code>double acos(double x);</code></li> <li>• <code>#include &lt;string&gt;</code><br/><code>using namespace std;</code><br/><code>string s="hop";</code><br/><code>char c=s[0];</code><br/><code>int l=s.size();</code></li> <li>• <code>#include &lt;ctime&gt;</code><br/><code>s=double(clock())</code><br/><code>/CLOCKS_PER_SEC;</code></li> <li>• <code>#include &lt;cmath&gt;</code><br/><code>double pi=M_PI;</code></li> </ul> | <ul style="list-style-type: none"> <li>• Arrays: not to translate math formula!</li> <li>• Make structures.</li> <li>• Do separate source files</li> <li>• The <code>.h</code> must be enough for the user (who must not look into <code>.cpp</code>)</li> <li>• Don't abuse recursion.</li> <li>• Don't forget delete.</li> <li>• Compile regularly.</li> <li>• <code>#include &lt;cassert&gt;</code><br/>...<br/><code>assert(x!=0);</code><br/><code>y=1/x;</code></li> <li>• Make objects.</li> <li>• Do not always make objects!</li> <li>• Think interface / implementation / usage.</li> </ul>                                                                                                         | <ul style="list-style-type: none"> <li>• <code>int n=5;</code><br/><code>int t[n];</code> // NO</li> <li>• <code>int f()[4] {</code> // NO!<br/><code>int t[4];</code><br/>...<br/><code>return t;</code> // NO!<br/><code>}</code><br/><code>int t[4]; t=f();</code></li> <li>• <code>int s[3]={1,2,3},t[3];</code><br/><code>t=s;</code> // NO!</li> <li>• <code>int t[2];</code><br/><code>t={1,2};</code> // NO!</li> <li>• <code>struct Point {</code><br/><code>double x,y;</code><br/><code>}</code> // NO!</li> <li>• <code>Point a;</code><br/><code>a={1,2};</code> // NO!</li> <li>• <code>#include "tp.cpp"//NO</code></li> <li>• <code>int f(int t[][]);</code>//NO<br/><code>int t[2,3];</code> // NO!<br/><code>t[i,j]=...;</code> // NO!</li> <li>• <code>int* t;</code><br/><code>t[1]=...;</code> // NO!</li> <li>• <code>int* t=new int[2];</code><br/><code>int* s=new int[2];</code><br/><code>s=t;</code> // lost s!<br/><code>delete[] t;</code><br/><code>delete[] s;</code>//Crash!</li> <li>• <code>int *t,s;</code>// s is int<br/>// not int*<br/><code>t=new int[n];</code><br/><code>s=new int[n];</code>// NO!</li> <li>• <code>class vec {</code><br/><code>int x,y;</code><br/><code>public:</code><br/>...<br/><code>};</code><br/>...<br/><code>vec a={2,3};</code> // NO</li> <li>• <code>vec v=vec(1,2);</code>//NO<br/><code>vec v(1,2);</code> // Yes</li> </ul> |
| <p><b>Imagine++</b></p> <ul style="list-style-type: none"> <li>• See documentation...</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | <p><b>Frequent errors</b></p> <ul style="list-style-type: none"> <li>• No definition of function inside a function!</li> <li>• <code>int q=r=4;</code> // NO!</li> <li>• <code>if (i=2) // NO!</code><br/><code>if i==2 // NO!</code><br/><code>if (i==2) then // NO!</code></li> <li>• <code>for(int i=0,i&lt;100,i++)</code><br/><code>// NO!</code></li> <li>• <code>int f() {...}</code><br/><code>int i=f;</code> // NO!</li> <li>• <code>double x=1/3;</code> // NO!<br/><code>int i,j;</code><br/><code>x=i/j;</code> // NO!<br/><code>x=double(i/j);</code> //NO!</li> <li>• <code>double x[10],y[10];</code><br/><code>for(int i=1;i&lt;=10;i++)</code><br/><code>y[i]=2*x[i];</code>//NO</li> </ul> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <p><b>Advice</b></p> <ul style="list-style-type: none"> <li>• Errors/warnings: click.</li> <li>• Indent!</li> <li>• Fix warnings.</li> <li>• Use the debugger.</li> <li>• Write functions.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |



# Chapter 11

## Destructor

*This chapter can be considered as difficult. Still, we recommend you make the effort to understand it. It is about the management of resources, notably memory, which relies on the notion of **destructor**. We will see how its usage will help in automatic management of the heap.*

—

### 11.1 Destructor

When an object dies, another of its methods is called: the *destructor*.

**The destructor:**

- is called when the object dies.
- has as “identifier” the class name prefixed by ~.
- like constructors, has no return type.
- takes no argument

A consequence of the last item is that *there is a single destructor per class*. An example will be more telling. Let us add a destructor to the program of Section 10.3:

```
#include <iostream>
using namespace std;

class obj {
public:
 obj();
 ~obj();
};

obj::obj() {
 cout << "obj_";
}

obj::~~obj() {
```

```

 cout << "~\n";
}

void f(obj d) {
}

obj g() {
 obj e;
 cout << 6 << "\n";
 return e;
}

int main()
{
 cout << 0 << "\n";
 obj a;
 cout << 1 << "\n";
 for (int i=2;i<=4;i++) {
 obj b;
 cout << i << "\n";
 }
 f(a);
 cout << 5 << "\n";
 a=g();
 return 0;
}

```

It displays now:

```
0 obj 1 obj 2 ~ obj 3 ~ obj 4 ~ ~ 5 obj 6 ~ ~ ~
```

Try to follow the program step by step and spot where destructors are called. Notice that for each iteration of the `for` loop, 2, 3 and 4, an object `b` is created (constructor called) and at the end of iteration `b` dies so that its destructor is called. After the call to function `f`, its parameter `d` dies, hence the `~` before displaying 5. The behavior of function `g` is more complex to understand, we will come back to it; however, a local object `e` is created, hence destroyed at exit. The last `~` marks the death of variable `a` when exiting the main function. Notice that we can spot only 5 constructors compared to 7 destructors. We could have expected the same number of both. Actually, 2 objects were constructed by copy, as will be explained in this chapter.

Concerning a field of object type, as in Section 10.3.4, there is nothing special to do: the destructor `Circle::~~Circle()` calls automatically the destructor `point::~~point()` for its field `center`. The programmer *must not* call the destructor explicitly. The general rule is that

**A destructor must never be called explicitly, since the call happens automatically.**

## 11.2 Destructors and arrays

For an array, the destructor is called for each element of the array:

```

1 if (a==b) {
2 obj t[10];
3 ...
4 }
```

will call 10 times the empty constructor at line 2 and 10 times the destructor at line 4. In case of a dynamic array, it is at the moment of `delete []` that destructors are called (before deallocation of the array).

```

if (a==b) {
 obj* t=new obj[n]; // n calls to obj()
 ...
 delete [] t; // n calls to ~obj();
}
```

**Beware: it is possible to write `delete t` without brackets `[]`. It is a mistake! That simpler syntax is reserved to another usage of `new/delete` with pointers. Using it has for consequence the deallocation but not the call of the destructor on each `t[i]`.**

## 11.3 Copy constructors

Let us finally discover this famous constructor we alluded to several times. It is a constructor taking as argument another object of same type, as a constant reference.

**The *copy constructor*:**

- Is declared as: `obj::obj(const obj& o);`
- can be used explicitly as:
 

```
obj a;
obj b(a); // b from a
```
- But is also (surprisingly) used in:
 

```
obj a;
obj b=a; // b from a, identical to b(a)
```

 Do not mix with:
 

```
obj a,b;
b=a; // this is not a constructor!
```
- And above all to build parameters to functions and return value.

The last item is the difficult part, let us see our completed program:

```

obj::obj(const obj& o) {
 cout << "copy_";
}
```

which prints:

```
0 obj 1 obj 2 ~ obj 3 ~ obj 4 ~ copy ~ 5 obj 6 copy ~ ~ ~
```

We have at last 7 constructors for the 7 destructors! The call to function `f` should be now easy to understand. The call to `g` is slightly harder: local object `e` should be destroyed at exit, as expected. However, how can we return an object that will die just next? Actually, before destroying `e`, a copy constructor is called with `e` as argument to create an anonymous object. Said anonymous object then has to die just after usage in the calling instruction, here after `a=g()`.

There is an important point to know about this constructor, which will bite us later if we are not careful:

**When a copy constructor is not defined explicitly, a default copy constructor is created and its copies the fields of its arguments in the built object.**

Remember also that when we define a constructor, the default empty constructor does not exist anymore. Yet, the default copy constructor still exists!

For a field of object type, as in the example of Section 10.3.4, the default copy constructor calls the copy constructor of the field type. In other words, the copy constructor `Circle::Circle(const Circle& c)` calls `point::point(const point& p)` for its field `center` (default or defined one).

## 11.4 Assignment

There is one last thing that is possible to reprogram for an object: assignment. If not defined explicitly, the default assignation behaves naturally by assigning all fields individually. To provide an explicit assignment operator, we redefine `=`. Therefore `a=b` is the nice version of `a.operator=(b)`. Let us add it:

```
void obj::operator=(const obj& o) {
 cout << "=_" ;
}
```

to our program, which now prints:

```
0 obj 1 obj 2 ~ obj 3 ~ obj 4 ~ copy ~ 5 obj 6 copy ~ = ~ ~
```

Notice the assignment operator called by instruction `a=g()`, with argument the anonymous object returned by function `g`.

Generally, we refine it a little bit. The instruction `a=b=c`; between basic types works for two reasons:

- It is interpreted as `a=(b=c)`; (associativity from right to left)
- The instruction `b=c` assigns `c` to `b` **and** returns the value of `c`.

To be able to do the same between three objects, we can define the assignment as follows:

```
obj obj::operator=(const obj& o) {
 cout << "=_" ;
 return o;
}
```



```

}
...
obj a,b,c;
a=b=c; // OK since a=(b=c)

```

or even like that, which goes beyond our current knowledge, but that should be followed as it avoids copying an object at `return` time:

```

const obj& obj::operator=(const obj& o) {
 cout << "=_" ;
 return o; // or return *this if you prefer
}
...
obj a,b,c;
a=b=c; // OK since a=(b=c)

```

If you look at the code of an experienced programmer, you will see something like:

```

obj& obj::operator=(const obj& o) {
 if (this != &obj) {
 ... }
 return *this;
}

```

which on top may prevent mistakes with silly code like `a=a`, which could turn catastrophic as we will see in the next section.

For an object with a field of object type, like in Section 10.3.4, the default assignment operator `Circle` calls the assignment operator of `point` for the field `center`, whether the default one or the defined one. But if you define the assignment operator for `Circle`, you must indicate that you wish to copy the `point`:

```

Circle& Circle::operator=(const Circle& c) { // My own assignment
 if (this != &c) {
 center = c.center; // Calls point::operator=(const point&)
 radius = radius;
 }
 return *this;
}

```

In this case the definition of the assignment operator for `Circle` does nothing different from the default one, and the code we wrote is just bloat. The idea is to write the minimum required code:

**Do not be pedantic! It is only useful to have explicit copy constructor and assignment operator when we want them to do something else that their default behavior.<sup>a</sup>**

<sup>a</sup>On the contrary to empty constructor that is not provided by default when another regular (that is, not copy) constructor is defined, and that may need to be provided, even to reproduce the default behavior!

## 11.5 Objects with dynamic allocation

All we saw here was a bit abstract, and the interest of defining explicitly destructor, copy constructor and assignment operator may look remote. Let us discover at last how that helps. Consider the following class, which we will improve step by step:

```
#include <iostream>
using namespace std;

class vect {
 int n;
 double *t;
public:
 void allocate(int N);
 void free();
};

void vect::allocate(int N) {
 n=N;
 t=new double[n];
}

void vect::free() {
 delete[] t;
}

int main()
{
 vect v;
 v.allocate(10); // necessary before usage
 ...
 v.free(); // do not forget
 return 0;
}
```

### 11.5.1 Construction and destruction

It is obvious that constructor and destructor are here to help us:

```
#include <iostream>
using namespace std;

class vect {
 int n;
 double *t;
public:
 vect(int N);
 ~vect();
};
```

```

vect::vect(int N) { // replace allocate
 n=N;
 t=new double[n];
}

vect::~~vect() { // replace free
 delete [] t;
}

int main()
{
 vect v(10);
 ...
 return 0;
}

```

**Thanks to constructor and destructor, we can at last let allocations and deallocations happen by themselves!**

### 11.5.2 Problems!

The dark side is that this way of doing things will lead us to go a bit far for beginners. We will face two types of problems.

#### A simple problem

Since there is a single destructor for several constructors, we need to be careful with what happens in the destructor. Let us add for example an empty constructor:

```

vect::vect() { // A bit silly but licit to have a usable object
}

```

then the destruction of an object created *ex nihilo* will deallocate an absurd field *t*, since never initialized. We can fix it easily:

```

vect::vect() {
 n=0;
}
vect::~~vect() {
 if(n!=0)
 delete [] t;
}

```

#### Complex problems

The following usage of our class does not work:

```
int main()
{
 vect v(10),w(10);
 w=v;
 return 0;
}
```

Why? Because the assignment (by default, since not defined) copies the fields of  $v$  into the ones of  $w$ . Thus,  $v$  and  $w$  have the same value for  $t$ ! Remember such a value is an address in memory. Not only will the two objects share the same array (hence modifying one coefficient will also modify the other, without the user realizing), but on top *a same zone of the heap will be deallocated twice, while another will never be!*<sup>1</sup>

For protection, we must then redefine assignment, which is not trivial, but could be reproduced once understood. The simpler option is to allocate new memory and copy the elements of the array.

```
vect& vect::operator=(const vect& v) {
 if(n!=0)
 delete [] t; // Deallocation if necessary
 n=v.n;
 if(n!=0) {
 t=new double[n]; // Reallocation and copy
 for (int i=0;i<n;i++)
 t[i]=v.t[i];
 }
 return *this;
}
```

This version will have dire consequences if the user does  $v=v$  (can you see why?).<sup>2</sup>

### 11.5.3 Solution!

Identical problems happen for copy constructor. This being said, factorizing the code in a few private functions, the solution is fairly easy and fairly light. We present it as it comes, you should be able to understand the logic.<sup>3</sup>

```
1 #include <iostream>
2 using namespace std;
3
4 class vect {
5 int n; // Dimension
6 double *t;
7 // private functions
8 void alloc(int N);
9 void kill();
```

<sup>1</sup>That is, until the end of program. This is an instance of our dreaded *memory leaks*, errors that are not fatal but still annoying. However, deallocating twice the same array is likely to crash the program!

<sup>2</sup>As mentioned before, the test ( $\&v==this$ ) is a lifesaver...

<sup>3</sup>This is only the first step to different ways to manage objects. Must we copy the arrays? Share them so that the last user deallocates? Apply "copy on write" (COW)? Let's leave that to an advanced course.

```

10 void copy(const vect& v);
11 public:
12 // "mandatory" constructors
13 vect();
14 vect(const vect& v);
15 vect(int N); // additional constructor
16 ~vect(); // destructor
17 vect& operator=(const vect& v); // assignment
18 };
19
20 void vect::alloc(int N) {
21 n=N;
22 if(n!=0)
23 t=new double[n];
24 }
25
26 void vect::kill() {
27 if(n!=0)
28 delete [] t;
29 }
30
31 void vect::copy(const vect& v) {
32 alloc(v.n);
33 for(int i=0; i<n; i++) // OK even if n==0
34 t[i]=v.t[i];
35 }
36
37 vect::vect() { alloc(0); }
38 vect::vect(const vect& v) { copy(v); }
39 vect::~~vect() { kill(); }
40
41 vect& vect::operator=(const vect& v) {
42 if(this!=&v) {
43 kill();
44 copy(v);
45 }
46 return *this;
47 }
48
49 vect::vect(int N) {
50 alloc(N);
51 }
52
53 // To test copy constructor
54 vect f(vect a) { return a; }
55 // To test everything works
56 int main() {
57 vect a,b(10),c(12),d;

```

```

58 a=b;
59 a=a;
60 a=c;
61 a=d;
62 a=f(a);
63 b=f(b);
64 return 0;
65 }

```

Notice that the assignment operator always behaves in the same pattern: 1. do the same as destructor (without calling it, you must never call a destructor explicitly); 2. do the same as copy constructor (without calling it, there is no new object to create!).

## 11.6 Reference card

| Reference Card (1/4)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Loops</b></p> <ul style="list-style-type: none"> <li>do {<br/>...<br/>} while(!ok);</li> <li>int i=1;<br/>while(i&lt;=100) {<br/>...<br/>i=i+1;<br/>}</li> <li>for(int i=1;i&lt;=10;i++)<br/>...</li> <li>for(int i=1,j=10;j&gt;i;<br/>i=i+2,j=j-3)<br/>...</li> <li>for (int i=...)<br/>for (int j=...) {<br/>//skip case i==j<br/>if (i==j)<br/>continue;<br/>...<br/>}</li> </ul> <p><b>Keys</b></p> <ul style="list-style-type: none"> <li>Debug: F5 </li> <li>Step over: F10 </li> <li>Step inside: F11 </li> <li>Indent: Ctrl+A, Ctrl+I</li> <li>Switch source/header: F4</li> <li>Switch decl./def.: F2</li> </ul> | <ul style="list-style-type: none"> <li>Step out: Maj+F11 </li> <li>Gest. tâches: Ctrl+Maj+Ech</li> </ul> <hr/> <p><b>Structures</b></p> <ul style="list-style-type: none"> <li>struct Point {<br/>double x,y;<br/>Color c;<br/>};<br/>...<br/>Point a;<br/>a.x=2.3; a.y=3.4;<br/>a.c=RED;<br/>Point b={1,2.5,BLUE};</li> <li>A structure is an objet fully public (→ see objects!)</li> </ul> <hr/> <p><b>Variables</b></p> <ul style="list-style-type: none"> <li>Definition:<br/>int i;<br/>int k,l,m;</li> <li>Assignment:<br/>i=2;<br/>j=i;<br/>k=l=3;</li> <li>Initialization:<br/>int n=5,o=n;</li> <li>Constants:<br/>const int s=12;</li> <li>Scope:<br/>int i;</li> </ul> | <pre> // i=j; forbidden! int j=2; i=j; // OK! if (j&gt;1) {     int k=3;     j=k; // OK! } //i=k; forbidden! </pre> <ul style="list-style-type: none"> <li>Types:<br/>int i=3;<br/>double x=12.3;<br/>char c='A';<br/>string s="hop";<br/>bool t=true;<br/>float y=1.2f;<br/>unsigned int j=4;<br/>signed char d=-128;<br/>unsigned char d=25;<br/>complex&lt;double&gt;<br/>z(2,3);</li> <li>Global variables:<br/>int n;<br/>const int m=12;<br/>void f() {<br/>n=10; // OK<br/>int i=m; // OK<br/>...}</li> <li>Conversion:<br/>int i=int(x),j;<br/>float x=float(i)/j;</li> <li>Stack/Heap</li> </ul> |

| Reference Card (2/4)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Functions</b></p> <ul style="list-style-type: none"> <li>• <b>Definition:</b><br/> <pre>int plus(int a,int b){     int c=a+b;     return c; } void display(int a) {     cout &lt;&lt; a &lt;&lt; endl; }</pre> </li> <li>• <b>Declaration:</b><br/> <pre>int plus(int a,int b);</pre> </li> <li>• <b>Return:</b><br/> <pre>int sign(double x) {     if (x&lt;0)         return -1;     if (x&gt;0)         return 1;     return 0; } void display(int x,              int y) {     if (x&lt;0    y&lt;0)         return;     if (x&gt;=w    y&gt;=h)         return;     DrawPoint(x,y,RED); }</pre> </li> <li>• <b>Call:</b><br/> <pre>int f(int a) { ... } int g() { ... } ... int i=f(2),j=g();</pre> </li> <li>• <b>References:</b><br/> <pre>void swap(int&amp; a,           int&amp; b){     int tmp=a;     a=b;b=tmp; } ...</pre> </li> </ul> | <pre>int x=3,y=2; swap(x,y);</pre> <ul style="list-style-type: none"> <li>• <b>Overload:</b><br/> <pre>int chance(int n); int chance(int a,            int b); double chance();</pre> </li> <li>• <b>Operators:</b><br/> <pre>vect operator+(                vect A,vect B) {     ... } ... vect C=A+B;</pre> </li> <li>• <b>Call stack</b></li> <li>• <b>Iterative/Recursive</b></li> <li>• <b>Constant references (avoid copy):</b><br/> <pre>void f(const obj&amp; x){     ... } void g(const obj&amp; x){     f(x); // OK }</pre> </li> </ul> <hr/> <p><b>Arrays</b></p> <ul style="list-style-type: none"> <li>• <b>Definition:</b><br/> <pre>- double x[5],y[5];   for(int i=0;i&lt;5;i++)       y[i]=2*x[i];</pre> <pre>- const int n=5;   int i[n],j[2*n];</pre> </li> <li>• <b>Initialization:</b><br/> <pre>int t[4]={1,2,3,4}; string s[2]={"ab","c"};</pre> </li> </ul> | <ul style="list-style-type: none"> <li>• <b>Assignment:</b><br/> <pre>int s[3]={1,2,3},t[3]; for (int i=0;i&lt;3;i++)     t[i]=s[i];</pre> </li> <li>• <b>As parameter:</b><br/> <pre>- void init(int t[4]){     for(int i=0;i&lt;4;i++)         t[i]=0; } - void init(int t[],            int n) {     for(int i=0;i&lt;n;i++)         t[i]=0; }</pre> </li> <li>• <b>Variable size:</b><br/> <pre>int* t=new int[n]; ... delete[] t;</pre> </li> <li>• <b>As argument:</b><br/> <pre>- void f(int* t,int n)     { t[i]=... } - void alloc(int*&amp; t){     t=new int[n]; }</pre> </li> <li>• <b>2D:</b><br/> <pre>int A[2][3]; A[i][j]=...; int A[2][3]=     {{1,2,3},{4,5,6}}; void f(int A[2][2]);</pre> </li> <li>• <b>2D in 1D:</b><br/> <pre>int A[2*3]; A[i+2*j]=...;</pre> </li> <li>• <b>Variable size:</b><br/> <pre>int *t,*s,n;</pre> </li> </ul> |

| Reference Card (3/4)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Objects</b></p> <ul style="list-style-type: none"> <li>• struct obj {<br/>  int x; // field<br/>  int f(); // method<br/>  int g(int y);<br/>};<br/>int obj::f() {<br/>  int i=g(3); // my g<br/>  int j=x+i; // my x<br/>  return j;<br/>}<br/>...<br/>int main() {<br/>  obj a;<br/>  a.x=3;<br/>  int i=a.f();<br/>}</li> <li>• class obj {<br/>  int x,y;<br/>  void mine();<br/>public:<br/>  int z;<br/>  void for_all();<br/>  void another(obj A);<br/>};<br/>void obj::mine() {<br/>  x=..; // OK<br/>  ..=y; // OK<br/>  z=..; // OK<br/>}<br/>void obj::for_all(){<br/>  x=..; // OK<br/>  mine(); // OK<br/>}<br/>void another(obj A) {<br/>  x=A.x; // OK<br/>  A.mine(); // OK<br/>}<br/>...<br/>int main() {<br/>  obj A,B;<br/>  A.x=..; //NO<br/>  A.z=..; //OK<br/>  A.mine(); //NO<br/>  A.for_all(); //OK<br/>  A.another(B); //OK<br/>}</li> <li>• class obj {<br/>  obj operator+(obj B);<br/>};<br/>...<br/>int main() {<br/>  obj A,B,C;<br/>  C=A+B;<br/>  // C=A.operator+(B)<br/>}</li> <li>• Constant methods:<br/>void obj::f() const{</li> </ul> | <pre> ... } void g(const obj&amp; x){   x.f(); // OK } </pre> <ul style="list-style-type: none"> <li>• <b>Constructor:</b><br/>class point {<br/>  int x,y;<br/>public:<br/>  point(int X,int Y);<br/>};<br/>point::point(int X,<br/>                  int Y){<br/>  x=X;<br/>  y=Y;<br/>}<br/>...<br/>  point a(2,3);</li> <li>• <b>Empty constructor:</b><br/>obj::obj() {<br/>  ...<br/>}<br/>...<br/>  obj a;</li> <li>• <b>Temporary objects:</b><br/>vec vec::operator+(<br/>                          vec b) {<br/>  return vec(x+b.x,<br/>                          y+b.y);<br/>}<br/>...<br/>  c=vec(1,2)<br/>  +f(vec(2,3));</li> <li>• <b>Destructor:</b><br/>obj::~~obj() {<br/>  ...<br/>}</li> <li>• <b>Copy constructor:</b><br/>obj::obj(const obj&amp; o)<br/>  { ... }<br/><b>Used by:</b><br/>- obj b(a);<br/>- obj b=a;<br/>//Not obj b;b=a;<br/><b>- function argument</b><br/><b>- return value</b></li> <li>• <b>Assignment:</b><br/>obj&amp; obj::operator=(<br/>                          const obj&amp;o){<br/>  ...<br/>  return *this;<br/>}</li> <li>• <b>Objects with automatic dynamic allocation: Section 11.5</b></li> </ul> | <p><b>Separate compilation</b></p> <ul style="list-style-type: none"> <li>• #include "vect.h", also in vect.cpp</li> <li>• Functions: declarations in .h, definitions in .cpp</li> <li>• Types: definitions in .h</li> <li>• Declare in .h only useful functions</li> <li>• #pragma once at beginning of header file</li> <li>• Do not cut too much...</li> </ul> <p><b>Tests</b></p> <ul style="list-style-type: none"> <li>• Comparison:<br/>== != &lt; &gt; &lt;= &gt;=</li> <li>• Negation: !</li> <li>• Combinations: &amp;&amp;   </li> <li>• if (i==0) j=1;</li> <li>• if (i==0) j=1;<br/>  else j=2;</li> <li>• if (i==0) {<br/>  j=1;<br/>  k=2;<br/>}</li> <li>• bool t=(i==0);<br/>  if (t)<br/>    j=1;</li> <li>• switch (i) {<br/>  case 1:<br/>    ...;<br/>    break;<br/>  case 2:<br/>  case 3:<br/>    ...;<br/>    break;<br/>  default:<br/>    ...;<br/>}</li> </ul> <p><b>Input/Output</b></p> <ul style="list-style-type: none"> <li>• #include &lt;iostream&gt;<br/>  using namespace std;<br/>  ...<br/>  cout &lt;&lt;"I="&lt;&lt;i&lt;&lt;endl;<br/>  cin &gt;&gt; i &gt;&gt; j;</li> </ul> |



| Reference Card (4/4)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Misc.</b></p> <ul style="list-style-type: none"> <li>• <code>i++;</code><br/><code>i--;</code><br/><code>i-=2;</code><br/><code>j+=3;</code></li> <li>• <code>j=i%n;</code> // Modulo</li> <li>• <code>#include &lt;cstdlib&gt;</code><br/>...<br/><code>i=rand()%n;</code><br/><code>x=rand()/</code><br/><code>double(RAND_MAX);</code></li> <li>• <code>#include &lt;ctime&gt;</code><br/>// Single call<br/><code>srand((unsigned int)</code><br/><code>time(0));</code></li> <li>• <code>#include &lt;cmath&gt;</code><br/><code>double sqrt(double x);</code><br/><code>double cos(double x);</code><br/><code>double sin(double x);</code><br/><code>double acos(double x);</code></li> <li>• <code>#include &lt;string&gt;</code><br/><code>using namespace std;</code><br/><code>string s="hop";</code><br/><code>char c=s[0];</code><br/><code>int l=s.size();</code></li> <li>• <code>#include &lt;ctime&gt;</code><br/><code>s=double(clock())</code><br/><code>/CLOCKS_PER_SEC;</code></li> <li>• <code>#include &lt;cmath&gt;</code><br/><code>double pi=M_PI;</code></li> </ul> | <ul style="list-style-type: none"> <li>• Arrays: not to translate math formula!</li> <li>• Make structures.</li> <li>• Do separate source files</li> <li>• The <code>.h</code> must be enough for the user (who must not look into <code>.cpp</code>)</li> <li>• Don't abuse recursion.</li> <li>• Don't forget <code>delete</code>.</li> <li>• Compile regularly.</li> <li>• <code>#include &lt;cassert&gt;</code><br/>...<br/><code>assert(x!=0);</code><br/><code>y=1/x;</code></li> <li>• Make objects.</li> <li>• Do not always make objects!</li> <li>• Think interface / implementation / usage.</li> </ul> <p><b>Frequent errors</b></p> <ul style="list-style-type: none"> <li>• No definition of function inside a function!</li> <li>• <code>int q=r=4; // NO!</code></li> <li>• <code>if (i=2) // NO!</code><br/><code>if i==2 // NO!</code><br/><code>if (i==2) then // NO!</code></li> <li>• <code>for(int i=0,i&lt;100,i++)</code><br/><code>// NO!</code></li> <li>• <code>int f() {...}</code><br/><code>int i=f; // NO!</code></li> <li>• <code>double x=1/3; // NO!</code><br/><code>int i,j;</code><br/><code>x=i/j; // NO!</code><br/><code>x=double(i/j); //NO!</code></li> <li>• <code>double x[10],y[10];</code><br/><code>for(int i=1;i&lt;=10;i++)</code><br/><code>y[i]=2*x[i];//NO</code></li> <li>• <code>int n=5;</code><br/><code>int t[n]; // NO</code></li> </ul> | <ul style="list-style-type: none"> <li>• <code>int f()[4] { // NO!</code><br/><code>int t[4];</code><br/>...<br/><code>return t; // NO!</code><br/><code>}</code><br/><code>int t[4]; t=f();</code></li> <li>• <code>int s[3]={1,2,3},t[3];</code><br/><code>t=s; // NO!</code></li> <li>• <code>int t[2];</code><br/><code>t={1,2}; // NO!</code></li> <li>• <code>struct Point {</code><br/><code>double x,y;</code><br/><code>} // NO!</code></li> <li>• <code>Point a;</code><br/><code>a={1,2}; // NO!</code></li> <li>• <code>#include "tp.cpp"//NO</code></li> <li>• <code>int f(int t[][]);//NO</code><br/><code>int t[2,3]; // NO!</code><br/><code>t[i,j]=...; // NO!</code></li> <li>• <code>int* t;</code><br/><code>t[1]=...; // NO!</code></li> <li>• <code>int* t=new int[2];</code><br/><code>int* s=new int[2];</code><br/><code>s=t; // lost s!</code><br/><code>delete[] t;</code><br/><code>delete[] s;//Crash!</code></li> <li>• <code>int *t,s;// s is int</code><br/><code>// not int*</code><br/><code>t=new int[n];</code><br/><code>s=new int[n];// NO!</code></li> <li>• <code>class vec {</code><br/><code>int x,y;</code><br/><code>public:</code><br/>...<br/><code>};</code><br/>...<br/><code>vec a={2,3}; // NO</code></li> <li>• <code>vec v=vec(1,2);//NO</code><br/><code>vec v(1,2); // Yes</code></li> <li>• <code>obj* t=new obj[n];</code><br/><code>delete t;// missing []</code></li> </ul> |
| <p><b>Imagine++</b></p> <ul style="list-style-type: none"> <li>• See documentation...</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <p><b>Advice</b></p> <ul style="list-style-type: none"> <li>• Errors/warnings: click.</li> <li>• Indent!</li> <li>• Fix warnings.</li> <li>• Use the debugger.</li> <li>• Write functions.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |



# Chapter 12

## Strings, files

*The most important aspects for an introductory programming course have been covered in the previous chapters. In this one and the next one, we present some useful (and sometimes necessary) stuff. We begin here with strings and files, and miscellaneous functionalities.*

—

You know enough to create many programs. What you need more is practice. However, if you want to create your own project, you will realize you need a few additional things. This chapter may help you...

### 12.1 Strings

The *strings* are variables storing characters, that is, text. We already met them:

```
#include <string>
using namespace std;
...
string s="hop";
char c=s[0];
int l=s.size();
```

From the syntax, you have realized that a string is actually a class. The type of "hop" is actually *not* string but `const char*`. Such an array is not easy to manipulate, and it is better to transform it into a string. This is what is done above, creating *s* with the constructor taking a `const char*` as argument. Then, many other functionalities are available. Let us introduce a few of them:

1. Strings can be compared. The lexicographic order is used:

```
if (s1==s2) ...
if (s1!=s2) ...
if (s1<s2) ...
if (s1>s2) ...
if (s1>=s2) ...
if (s1<=s2) ...
```

2. We can search a character in a string:

```
size_t i=s.find('h'); // position of 'h' in s?
size_t j=s.find('h',3); // position of 'h' in s
 // from position 3,
 // ignoring s[0] to s[2]
```

- Beware, it the type `size_t`<sup>1</sup> that is used, not `int`. It is an unsigned integer, whose number of bytes is chosen by C++...
- If the character is not found, `find` returns `string::npos` (a constant, actually the largest number storable in `size_t`).

3. We can also look for a sub-string:

```
size_t i=s.find("hop"); // find "hop" in s?
size_t j=s.find("hop",3); // find "hop" in s from position 3
```

4. Append a string to another:

```
string a="How_are_you";
string b="my_friends?";
string txt=a+" "+b;
```

5. Extract a substring:

```
string s1="one_two_three";
string s2=string(s1,4,3); // substring of length 3
 // starting at s1[4] (here "two")
```

6. Beware: getting a string from the user input (keyboard) cuts the string at the first blank found. If we enter "hello friends", the program

```
string s;
cin >> s; // Until "Enter" or space
```

will get "hello" in `s` (and then "friends" if another instruction `cin>>t` is present). To get the full line, spaces included:

```
getline(cin,s); // Full text line until "New line"
```

We can possibly choose another break character:

```
getline(cin,s,':'); // Everything up to ':' (not included)
```

7. Convert back a string to C format (array of characters): C strings are arrays of characters ending in 0.<sup>2</sup> Some functions take as parameter `char*` or `const char*`.<sup>3</sup> They would need to be fed with `s.c_str()` giving access to the internal C array stored by `s` (see Section 12.2.2).

```
string s="hop_hop";
const char *t=s.c_str();
```

<sup>1</sup>To be more verbose, even pedantic, you could use `string::size_type`.

<sup>2</sup>Not the character '0', whose code is not 0.

<sup>3</sup>We have not yet seen the role of `const` in arrays.

You will find other functions in the online help of your IDE (hopefully), or discover them yourself by looking at what the IDE proposes when you begin to call a method of a string.

## 12.2 Files

### 12.2.1 Principle

To read and write in a file, we can proceed exactly as with `cout` and `cin`. We simply have to create variables of type *ofstream* to write in a file, or of type *ifstream* to read. It is surprisingly simple.

1. Here is how to do

```
#include <fstream>
using namespace std;
...
 ofstream f("hop.txt");
 f << 1 << ' ' << 2.3 << ' ' << "hello" << endl;
 f.close();

 ifstream g("hop.txt");
 int i;
 double x;
 string s;
 g >> i >> x >> s;
 g.close();
```

2. It is advised to check that the file opening went well. A frequent error is to give an erroneous file name: the file is not open then.

```
ifstream g("../data/hop.txt");
if (!g.is_open()) {
 cout << "help!" << endl;
 return 1;
}
```

(use always slash `/`, portable, and not backslash `\`, even under Windows). We can also need to know if we reached the end of file while reading:

```
do {
 ...
} while (!(g.eof()));
```

3. A useful function (actually a preprocessor macro) from `Imagine++` (in module `Common`) is `srcPath`, which replaces a relative path to absolute path by prefixing it with the folder containing the source file. Doing so ensures that the file will be found whatever the current folder of the running program. If my source folder is `/home/pascal/Test/`,

```
ifstream g(srcPath("hop.txt"));
```

will look for the file `/home/pascal/Test/hop.txt`, even if my running program is in the different build folder. The equivalent for the type `string` is `stringSrcPath`.

4. A file can be opened after construction:

```
ofstream f;
f.open("hop.txt");
...
```

5. Less frequent, but very useful to know: We can write in a file directly the byte stream in memory corresponding to a variable or array. The file is then lighter in size, reading and writing are faster (no need to translate a number or string of characters or the inverse).

```
double x[10];
double y;
ofstream f("hop.bin", ios::binary);
f.write((const char*)x, 10 * sizeof(double));
f.write((const char*)&y, sizeof(double));
f.close();
...
ifstream g("hop.bin", ios::binary);
g.read((char*)x, 10 * sizeof(double));
g.read((const char*)&y, sizeof(double));
g.close();
```

In such case, do not forget to use *opening mode* `ios::binary`. The downside is that the file is not human-readable; it is better to reserve it for large files.<sup>4</sup>

### 12.2.2 String and file

1. To open a file, you need to give the name as a C string, hence the conversion:

```
void read(string name) {
 ifstream f(name.c_str()); // Conversion is mandatory
 ...
}
```

2. To read a string including spaces, use the same as for `cin`:

```
getline(g, s);
getline(g, s, ':');
```

3. At last, a bit technical but very convenient: the `stringstream`, that are strings simulating virtual files (with no name). They are used notably to convert a string into number or the inverse:

---

<sup>4</sup>For example, image file formats do that, on top of compression.

```

#include <sstream>
using namespace std;

 string s="12";
 stringstream f;
 int i;
 // String to integer
 f << s; // Write string
 f >> i; // Read as integer! (i is worth 12)
 i++;
 // Integer to string
 f.clear(); // Do not forget if f was already used
 f << i; // Write integer
 f >> s; // Read as string (s is "13")

```

Note in the last case a convenient function to convert a numerical value (`int`, `double`, etc) in `string` (since C++ 2011): `s = std::to_string(i)`.

### 12.2.3 Objects and files

An attractive feature of operators `<<` and `>>`<sup>5</sup> is to possibility to redefine them for objects! It is technical, but you don't need to really understand it to reproduce! Here is how:

```

struct point {
 int x,y;
};

ostream& operator<<(ostream& f, const point& p) {
 f << p.x << ' ' << p.y; // or any format you wish!
 // We decided here to write
 // coordinates separated by a space
 return f;
}

istream& operator>>(istream& f, point& p) {
 f >> p.x >> p.y; // Just be sure to be compatible with <<
 return f;
}

...
point p;
cin >> p;
cout << p;
ofstream f("../hop.txt");
f << p;
...

```

---

<sup>5</sup>They may seem a bit painful for the programmer used to `printf` and `scanf` of C. We see here at last their power!

```
ifstream g("../hop.txt");
g >> p;
```

## 12.3 Default values for parameters

### 12.3.1 Principle

Often useful! We can give default values for the last parameters of a function, which they take if the values are not given as arguments:

```
void f(int a, int b=0, int c=0) {
 // ...
}

void g() {
 f(12); // Same as f(12,0,0);
 f(10,2); // Same as f(10,2,0);
 f(1,2,3); // Same as f(1,2,3);
}
```

If there is a declaration separate from the definition, we precise the default value *only* in declaration;

```
void f(int a, int b=0); // declaration

void g() {
 f(12); // Same as f(12,0);
 f(10,2); // Same as f(10,2);
}

void f(int a, int b) { // Do not repeat default value of b
 // ...
}
```

### 12.3.2 Usefulness

In general, we start from a function:

```
int f(int a, int b) {
 ...
}
```

Then, we wish to add a special behavior in a certain case:

```
int f(int a, int b, bool special) {
 ...
}
```

Rather than transforming all former calls to `f(..)` in `f(.., false)`, we just need:



```
int f(int a, int b, bool special=false) {
 ...
}
```

to leave former calls unchanged, and call only `f(.,., true)` in future particular cases.

### 12.3.3 Frequent errors

Some frequent errors related to default values:

1. Wanting to give default value to argument in the middle:

```
void f(int a, int b=3, int c) { // NO! Only last parameters
 // not at middle!
}
```

2. Having an overloaded function:

```
void f(int a) {
 ...
}
void f(int a, int b=0) { // Overload problem!
 ... // How to understand f(1)?
}
```

## 12.4 Accessors

Here come, in five steps, the points to know to write convenient and efficient accessors.

### 12.4.1 Reference as return type

Here is a beginner's error that makes those in the know cry:

```
int i; // Global variable
int f() {
 return i;
}
...
f()=3; // Nonsense (not better than 2=3)
```

We cannot store a value in the return of a function, in the same manner we do not write `2=3`! Well, actually, yes! It is possible. But only if the function returns a reference, that is a "link" to a variable:

```
int i; // Global variable
int& f() {
 return i;
}
...
f()=3; // OK! Put 3 in i!
```

**Warning:** teaching that to a beginner is dangerous, but I take the risk. In general, said beginner will write the horrendous:

```
int& f() {
 int i; // Local variable
 return i; // reference to variable that will die!
 // SERIOUS MISTAKE!
}
...
f()=3; // NO!!! i does not exist anymore. What will happen?
```

### 12.4.2 Usage

Even if an object is not a global variable, a field of the object does not die when exiting one of its methods! We can, starting from the program:

```
class point {
 double x[N];
public:
 void set(int i, double v);
};
void point::set(int i, double v) {
 x[i]=v;
}
...
point p;
p.set(1,2.3);
```

transform it into:

```
class point {
 double x[N];
public:
 double& element(int i);
};
double& point::element(int i) {
 return x[i];
}
...
point p;
p.element(1)=2.3;
```

### 12.4.3 operator()

Next step: it becomes even more useful when one knows `operator()` that allows re-defining parentheses:

```
class point {
 double x[N];
public:
```

```

 double& operator()(int i);
};
double& point::operator()(int i) {
 return x[i];
}
...
point p;
p(1)=2.3; // Smart, no?

```

Note that we can pass several arguments to the operator, which is quite useful for matrices for instance:

```

class mat {
 double x[M*N];
public:
 double& operator()(int i, int j);
};
double& mat::operator()(int i, int j) {
 return x[i+M*j];
}
...
mat A;
A(1,2)=2.3; // Nice! Like in Matlab.

```

Actually, `operator()` is the only operator that can take diverse number of arguments: others are unary (like `operator[]`) or binary (like `operator+`). In particular, we *cannot* use `operator[]` to index a matrix.

#### 12.4.4 Overload and constant method

We face now a **problem**. The preceding program does not allow writing:

```

void f(mat& A) {
 A(1,1)=2; // OK
}
void f(const mat& A) {
 double x=A(1,1); // NO! The compiler does not guess
 // that this line does not modify A!
}

```

since `operator()` is not guaranteed constant. There is fortunately a solution: programming two accessors, taking profit from the fact that between a regular method and a constant method, overload is possible, even if they have the same parameters! It looks like:

```

class mat {
 double x[M*N];
public:
 // Same name, same parameters, but one is const!
 // Overload is possible
 double& operator()(int i, int j);
 double operator()(int i, int j) const;
}

```

```

};
double mat::operator()(int i,int j) const {
 return x[i+M*j];
}
double& mat::operator()(int i,int j) {
 return x[i+M*j];
}
void f(mat& A) {
 A(1,1)=2; // OK, calls operator()
}
void f(const mat& A) {
 double x=A(1,1); // OK, calls the second one
}

```

### 12.4.5 Inline functions

#### Principle

Last step: *Calling a function and getting its return value is a complex mechanism, hence costly.* Calling `A(i,j)` instead of `A.x[i+M*j]` can be wasteful: more time is taken calling function `A.operator()(i,j)` and getting its return value than running the function itself! *This could lead us to go back to structures in preference to classes!*<sup>6</sup>

There is a means to remove this call mechanism by giving a hint to the compiler to copy the function core at the calling site. For that, we declare the function as `inline`. For example:

```

inline double sqr(double x) {
 return x*x;
}
...
double y=sqr(z-3);

```

does the same as writing `y=(z-3)*(z-3)`, without the cost of a function call!

#### Safeguards

Be sure to understand what follows:

- An `inline` function is recompiled at each calling site, which **slows down compilation and increases the program size!**
- `inline` is thus **reserved for short functions for which a call is costly compared to function core!**
- If the function were declared in header `.h` and defined in source `.cpp`, we now need to **put it only in .h** since the function user needs the definition, to replace the function call by its core!

---

<sup>6</sup>C programmers could also be tempted to use *macros* (that is, shortcuts with `#define`, which we did not learn on purpose, since they are frowned upon by C++ programmers). They are less powerful than `inline` since they do not check types, cannot access private members, etc. The C++ programmer will avoid them except when necessary!

- To be able to execute functions step by step with the debugger, functions that are `inline` are compiled as normal functions in mode Debug. Only the release mode will benefit from the speed-up.

### Case of methods

In the case of a method, do not forget to put the inline function definition in header `.h`. It is the time to reveal what we hid until now:

**It is possible to DEFINE A METHOD ENTIRELY IN THE CLASS DEFINITION, instead of only declaring it then put its definition outside the class. However, it is not mandatory,<sup>a</sup> slows down the compilation and goes against the idea that we would rather mask the contents of methods to the class user. It is thus RESERVED TO SMALL FUNCTIONS, those we would put `inline`.**

<sup>a</sup>Contrast that with Java! A source of bad habit for the Java programmer who switches to C++...

In practice, we could get:

```
class mat {
 double x[M*N];
public:
 inline double& operator()(int i, int j) {
 return x[i+M*j];
 }
 inline double operator()(int i, int j) const {
 return x[i+M*j];
 }
};
```

Actually, the `inline` decorator is redundant here, as methods defined in the core of the class definition are `inline`!

## 12.5 Assertions

Let us recall the existence of function<sup>7</sup> `assert()` seen at 7.6. Do not hesitate to use it since it makes the code easier to understand (answers the question “what are the preconceptions at this stage of the program?”) and makes error diagnostic easier. Knowing that it costs nothing in Release mode (since not even compiled), do not restrict their usage. Here is how to make your accessors secure:

```
#include <cassert>
```

```
class mat {
 double x[M*N];
public:
 double& operator()(int i, int j) {
 assert(0<=i && i<M && 0<=j && j<N);
 }
};
```

<sup>7</sup>Actually, it is a macro, not a function. Hence it is not in namespace `std`.

```

 return x[i+M*j];
}
double operator()(int i,int j) const {
 assert(0<=i && i<M && 0<=j && j<N);
 return x[i+M*j];
}
};

```

## 12.6 Enumerated types

It is a good idea to use constants to make a program more readable:

```

const int north=0,east=1,south=2,west=3;
void advance(int direction);

```

but it is a bit clumsy to do so! That is, if you know about the existence of *enumerated types*:

```

enum Dir {north , east , south , west };
void avance(Dir direction);

```

It looks like a new data type is created, whereas in truth only integers are understood.<sup>8</sup> A last thing: we can force some values if needed, like this:

```





enum Code {C10=200,
 C11=231,
 C12=240,
 C13, // worth 241
 C14}; // worth 242

```

—

That is all for today, folks! We will continue our list of miscellaneous utilities in next chapter. It is time to meet again our famous reference card...

## 12.7 Reference card

| Reference Card (1/5)                                                                                                                                                                                              |                                                                                                                                                                                                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Loops</b></p> <ul style="list-style-type: none"> <li>do {<br/>...<br/>} while(!ok);</li> <li>int i=1;<br/>while(i&lt;=100) {<br/>...<br/>i=i+1;<br/>}</li> <li>for(int i=1;i&lt;=10;i++)<br/>...</li> </ul> | <ul style="list-style-type: none"> <li>for(int i=1,j=10;j&gt;i;<br/>i=i+2,j=j-3)<br/>...</li> <li>for (int i=...)<br/>for (int j=...) {<br/>//skip case i==j<br/>if (i==j)<br/>continue;<br/>...<br/>}</li> </ul> | <ul style="list-style-type: none"> <li>Debug: F5 </li> <li>Step over: F10 </li> <li>Step inside: F11 </li> <li>Indent: Ctrl+A, Ctrl+I</li> <li>Switch source/header: F4</li> <li>Switch decl./def.: F2</li> <li>Step out: Maj+F11 </li> <li>Gest. tâches: Ctrl+Maj+Ech</li> </ul> |
|                                                                                                                                                                                                                   | <b>Keys</b>                                                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

<sup>8</sup>Except in C++ 2011, there is a syntax to create really a new type and have a different underlying type.

| Reference Card (2/5)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Structures</b></p> <ul style="list-style-type: none"> <li>• struct Point {<br/>    double x,y;<br/>    Color c;<br/>};<br/>...<br/>Point a;<br/>a.x=2.3; a.y=3.4;<br/>a.c=RED;<br/>Point b={1,2.5,BLUE};</li> <li>• A structure is an objet fully public (→ see objects!)</li> </ul>                                                                                                                                                                                                                                             | <pre>float y=1.2f; unsigned int j=4; signed char d=-128; unsigned char d=25; complex&lt;double&gt;     z(2,3);</pre> <ul style="list-style-type: none"> <li>• Global variables:<br/>int n;<br/>const int m=12;<br/>void f() {<br/>    n=10; // OK<br/>    int i=m; // OK<br/>    ...}</li> <li>• Conversion:<br/>int i=int(x),j;<br/>float x=float(i)/j;</li> <li>• Stack/Heap</li> <li>• Enumerated type:<br/>enum Dir{N,E,S,W};<br/>void advance(Dir d);</li> </ul> | <ul style="list-style-type: none"> <li>• switch (i) {<br/>    case 1:<br/>        ...;<br/>        ...;<br/>        break;<br/>    case 2:<br/>    case 3:<br/>        ...;<br/>        break;<br/>    default:<br/>        ...;<br/>}</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <p><b>Variables</b></p> <ul style="list-style-type: none"> <li>• Definition:<br/>int i;<br/>int k,l,m;</li> <li>• Assignment:<br/>i=2;<br/>j=i;<br/>k=l=3;</li> <li>• Initialization:<br/>int n=5,o=n;</li> <li>• Constants:<br/>const int s=12;</li> <li>• Scope:<br/>int i;<br/>// i=j; forbidden!<br/>int j=2;<br/>i=j; // OK!<br/>if (j&gt;1) {<br/>    int k=3;<br/>    j=k; // OK!<br/>}<br/>//i=k; forbidden!</li> <li>• Types:<br/>int i=3;<br/>double x=12.3;<br/>char c='A';<br/>string s="hop";<br/>bool t=true;</li> </ul> | <p><b>Tests</b></p> <ul style="list-style-type: none"> <li>• Comparison:<br/>== != &lt; &gt; &lt;= &gt;=</li> <li>• Negation: !</li> <li>• Combinations: &amp;&amp;   </li> <li>• if (i==0) j=1;</li> <li>• if (i==0) j=1;<br/>    else j=2;</li> <li>• if (i==0) {<br/>    j=1;<br/>    k=2;<br/>}</li> <li>• bool t=(i==0);<br/>    if (t)<br/>        j=1;</li> </ul>                                                                                              | <p><b>Advice</b></p> <ul style="list-style-type: none"> <li>• Errors/warnings: click.</li> <li>• Indent!</li> <li>• Fix warnings.</li> <li>• Use the debugger.</li> <li>• Write functions.</li> <li>• Arrays: not to translate math formula!</li> <li>• Make structures.</li> <li>• Do separate source files</li> <li>• The .h must be enough for the user (who must not look into .cpp)</li> <li>• Don't abuse recursion.</li> <li>• Don't forget delete.</li> <li>• Compile regularly.</li> <li>• #include &lt;cassert&gt;<br/>    ...<br/>    assert(x!=0);<br/>    y=1/x;</li> <li>• Make objects.</li> <li>• Do not always make objects!</li> <li>• Think interface / implementation / usage.</li> </ul> |

| Reference Card (3/5)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Functions</b></p> <ul style="list-style-type: none"> <li>• <b>Definition:</b><br/> <pre>int plus(int a,int b){     int c=a+b;     return c; } void display(int a) {     cout &lt;&lt; a &lt;&lt; endl; }</pre> </li> <li>• <b>Declaration:</b><br/> <pre>int plus(int a,int b);</pre> </li> <li>• <b>Return:</b><br/> <pre>int sign(double x) {     if (x&lt;0)         return -1;     if (x&gt;0)         return 1;     return 0; } void display(int x,              int y) {     if (x&lt;0    y&lt;0)         return;     if (x&gt;=w    y&gt;=h)         return;     DrawPoint(x,y,RED); }</pre> </li> <li>• <b>Call:</b><br/> <pre>int f(int a) { ... } int g() { ... } ... int i=f(2),j=g();</pre> </li> <li>• <b>References:</b><br/> <pre>void swap(int&amp; a,           int&amp; b){     int tmp=a;     a=b;b=tmp; } ... int x=3,y=2; swap(x,y);</pre> </li> <li>• <b>Overload:</b><br/> <pre>int chance(int n); int chance(int a,            int b); double chance();</pre> </li> <li>• <b>Operators:</b><br/> <pre>vect operator+(     vect A,vect B) {     ... }</pre> </li> </ul> | <pre>... vect C=A+B;</pre> <ul style="list-style-type: none"> <li>• <b>Call stack</b></li> <li>• <b>Iterative/Recursive</b></li> <li>• <b>Constant references (avoid copy):</b><br/> <pre>void f(const obj&amp; x){     ... } void g(const obj&amp; x){     f(x); // OK }</pre> </li> <li>• <b>Default values:</b><br/> <pre>void f(int a,int b=0); void g() {     f(12); // f(12,0);     f(10,2); // f(10,2); } void f(int a,int b) {     // ... }</pre> </li> <li>• <b>Inline (fast call):</b><br/> <pre>inline double sqr(double x) {     return x*x; } ... double y=sqr(z-3);</pre> </li> <li>• <b>Return as reference:</b><br/> <pre>int i; // global var int&amp; f() {     return i; } ... f ()=3; // i=3!</pre> </li> </ul> <hr/> <p><b>Arrays</b></p> <ul style="list-style-type: none"> <li>• <b>Definition:</b><br/> <pre>- double x[5],y[5];   for(int i=0;i&lt;5;i++)       y[i]=2*x[i]; - const int n=5;   int i[n],j[2*n];</pre> </li> <li>• <b>Initialization:</b><br/> <pre>int t[4]={1,2,3,4}; string s[2]={"ab","c"};</pre> </li> <li>• <b>Assignment:</b><br/> <pre>int s[3]={1,2,3},t[3]; for (int i=0;i&lt;3;i++)     t[i]=s[i];</pre> </li> </ul> | <ul style="list-style-type: none"> <li>• <b>As parameter:</b><br/> <pre>- void init(int t[4]){     for(int i=0;i&lt;4;i++)         t[i]=0; } - void init(int t[],            int n) {     for(int i=0;i&lt;n;i++)         t[i]=0; }</pre> </li> <li>• <b>Variable size:</b><br/> <pre>int* t=new int[n]; ... delete[] t;</pre> </li> <li>• <b>As argument:</b><br/> <pre>- void f(int* t,int n)     { t[i]=... }  - void alloc(int*&amp; t){     t=new int[n]; }</pre> </li> <li>• <b>2D:</b><br/> <pre>int A[2][3]; A[i][j]=...; int A[2][3]=     {{1,2,3},{4,5,6}}; void f(int A[2][2]);</pre> </li> <li>• <b>2D in 1D:</b><br/> <pre>int A[2*3]; A[i+2*j]=...;</pre> </li> <li>• <b>Variable size:</b><br/> <pre>int *t,*s,n;</pre> </li> </ul> <hr/> <p><b>Separate compilation</b></p> <ul style="list-style-type: none"> <li>• <code>#include "vect.h", also in vect.cpp</code></li> <li>• <b>Functions:</b> declarations in .h, definitions in .cpp</li> <li>• <b>Types:</b> definitions in .h</li> <li>• <b>Declare in .h only useful functions</b></li> <li>• <code>#pragma once</code> at beginning of header file</li> <li>• <b>Do not cut too much...</b></li> </ul> |



## Reference Card (4/5)

## Objects

```

• struct obj {
 int x; // field
 int f(); // method
 int g(int y);
};
int obj::f() {
 int i=g(3); // my g
 int j=x+i; // my x
 return j;
}
...
int main() {
 obj a;
 a.x=3;
 int i=a.f();
}

• class obj {
 int x,y;
 void mine();
public:
 int z;
 void for_all();
 void another(obj A);
};
void obj::mine() {
 x=..; // OK
 ..=y; // OK
 z=..; // OK
}
void obj::for_all(){
 x=..; // OK
 mine(); // OK
}
void another(obj A) {
 x=A.x; // OK
 A.mine(); // OK
}
...
int main() {
 obj A,B;
 A.x=..; //NO
 A.z=..; //OK
 A.mine(); //NO
 A.for_all(); //OK
 A.another(B); //OK
}

• class obj {
 obj operator+(obj B);
};
...
int main() {
 obj A,B,C;
 C=A+B;
 // C=A.operator+(B)
}

• Constant methods:
void obj::f() const{
 ...
}

```

```

void g(const obj& x){
 x.f(); // OK
}

• Constructor:
class point {
 int x,y;
public:
 point(int X,int Y);
};
point::point(int X,
 int Y){
 x=X;
 y=Y;
}
...
point a(2,3);

• Empty constructor:
obj::obj() {
 ...
}
...
obj a;

• Temporary objects:
vec vec::operator+(
 vec b) {
 return vec(x+b.x,
 y+b.y);
}
...
c=vec(1,2)
+f(vec(2,3));

• Destructor:
obj::~~obj() {
 ...
}

• Copy constructor:
obj::obj(const obj& o)
{ ... }
Used by:
- obj b(a);
- obj b=a;
//Not obj b;b=a;
- function argument
- return value

• Assignment:
obj& obj::operator=(
 const obj&o){
 ...
 return *this;
}

• Objects with automatic dynamic allocation: Section 11.5

• Accessors:
class mat{

```

```

 double *x;
public:
 double& operator()
 (int i,int j){
 assert(i>=0 ...);
 return x[i+M*j];
 }
 double operator()
 (int i,int j) const{
 assert(i>=0 ...);
 return x[i+M*j];
 }
 ...

```

## Misc.

```

• i++;
 i--;
 i-=2;
 j+=3;

• j=i%n; // Modulo

• #include <cstdlib>
 ...
 i=rand() %n;
 x=rand() /
 double(RAND_MAX);

• #include <ctime>
 // Single call
 srand((unsigned int)
 time(0));

• #include <cmath>
 double sqrt(double x);
 double cos(double x);
 double sin(double x);
 double acos(double x);

• #include <string>
 using namespace std;
 string s="hop";
 char c=s[0];
 int l=s.size();
 if (s1==s1) ...
 if (s1!=s2) ...
 if (s1<s2) ...
 size_t i=s.find('h'),
 j=s.find('h',3);
 k=s.find("hop");
 l=s.find("hop",3);
 a="how";
 b="are you?";
 txt=a+" "+b;
 s1="one two three";
 s2=string(s1,4,3);
 getline(cin,s);
 getline(cin,s,':');
 const char *t=s.c_str();

```

| Reference Card (5/5)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>#include &lt;ctime&gt; s=double(clock())   /CLOCKS_PER_SEC; #include &lt;cmath&gt; double pi=M_PI;</pre> <p><b>Input/Output</b></p> <ul style="list-style-type: none"> <li>• #include &lt;iostream&gt;<br/>using namespace std;<br/>...<br/>cout &lt;&lt;"I="&lt;&lt;i&lt;&lt;endl;<br/>cin &gt;&gt; i &gt;&gt; j;</li> <li>• #include &lt;fstream&gt;<br/>using namespace std;<br/>ofstream f("hop.txt");<br/>f &lt;&lt; 1 &lt;&lt; ' ' &lt;&lt; 2.3;<br/>f.close();<br/>ifstream g("hop.txt");<br/>if (!g.is_open()) {<br/>    return 1;<br/>}<br/>int i;<br/>double x;<br/>g &gt;&gt; i &gt;&gt; x;<br/>g.close();</li> <li>• do {<br/>    ...<br/>} while (!(g.eof()));</li> <li>• ofstream f;<br/>f.open("hop.txt");</li> <li>• double x[10],y;<br/>ofstream f("hop.bin",<br/>    ios::binary);<br/>f.write((const char*)x,<br/>    10*sizeof(double));<br/>f.write((const char*)&amp;y,<br/>    sizeof(double));<br/>f.close();<br/>ifstream g("hop.bin",<br/>    ios::binary);<br/>g.read((char*)x,<br/>    10*sizeof(double));<br/>g.read((const char*)&amp;y,<br/>    sizeof(double));<br/>g.close();</li> <li>• string s;<br/>ifstream f(s.c_str());</li> <li>• #include &lt;sstream&gt;<br/>using namespace std;<br/>stringstream f;<br/>// String to int<br/>f &lt;&lt; s; f &gt;&gt; i;</li> </ul> | <pre>// int to string f.clear(); f &lt;&lt; i; f &gt;&gt; s; ⇔ s = std::to_string(i);</pre> <ul style="list-style-type: none"> <li>• ostream&amp; operator&lt;&lt;(<br/>    ostream&amp; f,<br/>    const point&amp;p){<br/>    f&lt;&lt;p.x&lt;&lt;' ' &lt;&lt; p.y;<br/>    return f;<br/>}<br/>istream&amp; operator&gt;&gt;(<br/>    istream&amp; f,point&amp; p){<br/>    f&gt;&gt;p.x&gt;&gt;p.y;<br/>    return f;<br/>}</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                | <ul style="list-style-type: none"> <li>• Point a;<br/>a={1,2}; // NO!</li> <li>• #include "tp.cpp"//NO</li> <li>• int f(int t[][]);//NO<br/>int t[2,3]; // NO!<br/>t[i,j]=...; // NO!</li> <li>• int* t;<br/>t[1]=...; // NO!</li> <li>• int* t=new int[2];<br/>int* s=new int[2];<br/>s=t; // lost s!<br/>delete[] t;<br/>delete[] s;//Crash!</li> <li>• int *t,s;// s is int<br/>// not int*<br/>t=new int[n];<br/>s=new int[n];// NO!</li> <li>• class vec {<br/>    int x,y;<br/>public:<br/>    ...<br/>};<br/>...<br/>vec a={2,3}; // NO</li> <li>• vec v=vec(1,2);//NO<br/>vec v(1,2); // Yes</li> <li>• obj* t=new obj[n];<br/>delete t;// missing []</li> <li>• //NO!<br/>void f(int a=2,int b);</li> <li>• void f(int a,int b=0);<br/>void f(int a);// NO!</li> <li>• <b>Not anything inline!</b></li> <li>• int f() {<br/>    ...<br/>}<br/>f()=3; // HORROR!</li> <li>• int&amp; f() {<br/>    int i;<br/>    return i;<br/>}<br/>f()=3; // NO!</li> </ul> |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | <p><b>Frequent errors</b></p> <ul style="list-style-type: none"> <li>• No definition of function inside a function!</li> <li>• int q=r=4; // NO!</li> <li>• if (i=2) // NO!<br/>if i==2 // NO!<br/>if (i==2) then // NO!</li> <li>• for(int i=0,i&lt;100,i++)<br/>    // NO!</li> <li>• int f() {...}<br/>int i=f; // NO!</li> <li>• double x=1/3; // NO!<br/>int i,j;<br/>x=i/j; // NO!<br/>x=double(i/j); //NO!</li> <li>• double x[10],y[10];<br/>for(int i=1;i&lt;=10;i++)<br/>    y[i]=2*x[i];//NO</li> <li>• int n=5;<br/>int t[n]; // NO</li> <li>• int f()[4] { // NO!<br/>int t[4];<br/>    ...<br/>return t; // NO!<br/>}<br/>int t[4]; t=f();</li> <li>• int s[3]={1,2,3},t[3];<br/>t=s; // NO!</li> <li>• int t[2];<br/>t={1,2}; // NO!</li> <li>• struct Point {<br/>    double x,y;<br/>} // NO!</li> </ul> | <p><b>Imagine++</b></p> <ul style="list-style-type: none"> <li>• See documentation...</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

# Chapter 13

## Parameterized functions and classes (templates)

We terminate our repository of miscellaneous utilities. Among them, data structures in the *STL* (Standard Template Library) will need first the notion of `template`. We will just introduce this powerful aspect of C++.

—

### 13.1 `template`

#### 13.1.1 Principle

Let us consider the classic function to swap two variables:

```
void swap(int& a, int& b) {
 int tmp;
 tmp=a;
 a=b;
 b=tmp;
}

...
int i, j;
...
swap(i, j);
```

If now we have to swap two variables of type `double`, we need to add an overload of function `swap()`, identical except concerning variable types. Fortunately, the C++ allows defining functions with *generic type*, a bit like a variable, that the compiler will *instantiate* at call site with a specific type. This *generic programming* is practiced with the definition of a "`template`":

```
// Swap two variables of any type T
template <typename T>
void swap(T& a, T& b) {
 T tmp;
 tmp=a;
```

```

 a=b;
 b=tmp;
}
...
int a=2,b=3;
double x=2.1,y=2.3;
swap(a,b); // instantiate T as int
swap(x,y); // instantiate T as double
...

```

Another example:

```

// Maximum of two variables , provided operator > exists
// for type T
template <typename T>
T maxi(T a,T b) {
 return (a>b)?a:b;
}

```

The declaration `typename T` specifies the generic type. We can have several:

```

// Find e1 in array tab1 and put
// in e2 the element of tab2 of same index
// Return false if not found
template <typename T1,typename T2>
bool find(T1 e1,T2& e2, const T1* tab1, const T2* tab2, int n) {
 for(int i=0; i<n; i++)
 if(tab1[i]==e1) {
 e2=tab2[i];
 return true;
 }
 return false;
}
...
string names[3]={ "jean", "pierre", "paul" };
int ages[3]={21,25,15};
...
string nm="pierre";
int ag;
if(find(nm,ag, names, ages, 3))
 cout << nm << "_is_" << ag << "_years_old" << endl;
...

```

### 13.1.2 `template` and files

It is wise to remember that

**The compiler does not build a “magic” function that works on several types! Actually, it creates as many functions there are usages with different types (instantiations).**

For these reasons:

1. Writing `template` functions slows down the compilation.
2. We cannot put the declaration in a header and the definition in a `.cpp` file, since all users must know the definition. Hence, the rule is **to put everything in the header file**.<sup>1</sup>

### 13.1.3 Classes

It is also frequent that a class definition becomes all the more useful if it is generic. It is possible, but beware! In case of functions, the compiler understands from the arguments the used types. In the case of classes, the user needs to precise always the type with the syntax `obj<type>`:

```
// Pair of two variables of type T
template <typename T>
class pair {
 T x[2];
public:
 // constructors
 pair();
 pair(T A,T B);
 // accessors
 T operator()(int i) const;
 T& operator()(int i);
};

template <typename T>
pair<T>::pair() {
}

template <typename T>
pair<T>::pair(T A,T B) {
 x[0]=A; x[1]=B;
}

template <typename T>
T pair<T>::operator()(int i) const {
 assert(i==0 || i==1);
 return x[i];
}

template <typename T>
T& pair<T>::operator()(int i) {
 assert(i==0 || i==1);
 return x[i];
}
```

---

<sup>1</sup>This is annoying and goes against our principle of putting declarations in `.h` and to hide the definitions in `.cpp`. We had the same remark for `inline` functions.

```

}
...
pair<int> p(1,2), r;
int i=p(1);
pair<double> q;
q(1)=2.2;
...

```

When several types are generic, they are separated by a comma:

```

// Pair of two variables of different types
template <typename S, typename T>
class pair {
public:
 // All in public to simplify
 S x;
 T y;
 // constructors
 pair() {}
 pair(S X, T Y) { x=X; y=Y; }
};
...
pair<int, double> P(1, 2.3);
pair<string, int> Q;
Q.x="pierre";
Q.y=25;
...

```

At last, we can also put generic the choice of an integer (which must be known at compile time):

```

// n-uplet of variables of type T
// Beware: each nuplet<T,N> will be a different class
template <typename T, int N>
class nuplet {
 T x[N];
public:
 // accessors
 T operator()(int i) const {
 assert(i>=0 && i<N);
 return x[i];
 }
 T& operator()(int i) {
 assert(i>=0 && i<N);
 return x[i];
 }
};
...
nuplet<int,4> A;
A(1)=3;
nuplet<string,2> B;

```

```
B(1)= " pierre " ;
...
```

The functions must obviously adapt:

```
template <typename T, int N>
T sum(nuplet<T,N> u) {
 T s=u(0);
 for(int i=1; i<N; i++)
 s+=u(i);
 return s;
}
...
nuplet<double,3> C;
...
cout << sum(C) << endl;
...
```

Enthusiastic about such power, the programmer could be tempted to put `template` everywhere. Keep calm, think!

**Templates can be delicate to program, slow to compile, etc. Do not abuse them! Better to begin with classes and functions without template. We put them generic only when it appears we need to reuse the existing code with different types. And remember that the compiler creates a new class or a new function at each new instantiation.<sup>a</sup>**

<sup>a</sup>The nuplets above have nothing to do with variable size arrays. Everything happens as if we programmed several arrays of constant size, for different values of the size.

### 13.1.4 STL

Whereas programming `template` is delicate, using them is quite easy. The standard library of C++ offers some functions and classes using `template`. This set is commonly called STL (Standard Template Library). You can find its complete documentation on the web. We just show here some examples that could be used as starting point and make reading the documentation easier.

Some simple functions like `min` and `max` are provided in a generic manner:

```
int i=std::max(1,3);
double x=std::min(1.2,3.4);
```

Be cautious: a classic error is calling `max(1,2.3)`: the compiler understands it as the `max` of an `int` and a `double`, which results in a compilation error! We need to be more exact: `max(1.,2.3)`.

Complex numbers are also generic, keeping the real and imaginary parts template:

```
#include <complex>
using namespace std;
...
complex<double> z1(1.1,3.4), z2(1,0), z3;
z3=z1+z2;
cout << z3 << endl;
```

```
double a=z3.real(),b=z3.imag();
double m=abs(z3); // module
double th=arg(z3); // argument
```

Couples are also proposed by STL:

```
pair<int, string> P(2, "hop");
P.first=3;
P.second="hop";
```

At last, some data structures are included and can be used following a common scheme. See the example of lists:

```
#include <list>
using namespace std;
...
list<int> l; // l=[]
l.push_front(2); // l=[2]
l.push_front(3); // l=[3,2]
l.push_back(4); // l=[3,2,4]
l.push_front(5); // l=[5,3,2,4]
l.push_front(2); // l=[2,5,3,2,4]
```

To indicate a position in a list, we use an *iterator*. To indicate a position but for reading only, we use a *constant iterator*. The unary operator '\*' is used to access the element indexed by the iterator. Small difficulty, the type of these iterators is a bit awkward to type:<sup>2</sup>

```
list<int>::const_iterator it;
it=l.begin(); // index to beginning of list
cout << *it << endl; // prints 2
it=l.find(3); // Find location of first 3
if(it!=l.end())
 cout << "3_is_in_list" << endl;
list<int>::iterator it2;
it2=l.find(3); // Find location of first 3
*it=6; // now l=[2,5,6,2,4]
```

Iterators are also used to iterate (!) on lists:

```
// Enumerate and print list
template <typename T>
void print(const list<T>& l) {
 cout << "[";
 for(list<T>::const_iterator it=l.begin(); it!=l.end(); it++)
 cout << *it << ' ';
 cout << ']' << endl;
}

// Replace a by b in list
template <typename T>
```

---

<sup>2</sup>We have not seen how to define new types inside classes! That is what is done here...



```

void replace(list<T>& l, T a, T b) {
 for(list<T>::iterator it=l.begin(); it!=l.end(); it++)
 if(*it==a)
 *it=b;
}

...
print(l);
replace(l,2,1); // now l=[1,5,3,1,4]
...

```

Finally, we can call some algorithms such as sort on a list:

```

l.sort();
print(l);

```

In the same manner as lists, you will find in the STL:

- LIFO or stack.
- FIFO or queue.
- Collections without repetition set.
- Arrays of dynamic size vector.
- heap data structure.
- Tables or map (correspondence table key/value).
- A few others...

The remainder of the chapter groups several useful but not fundamental notions. They will at first help you more understanding already written code than using them in your own programs, at least until you get more practice.

## 13.2 Bitwise operators

Among classic errors, there is obviously the one consisting of replacing

```

if(i==0)
...

```

by

```

if(i=0) // NO!
...

```

which puts value 0 in i then considers 0 as a boolean, in this case [false](#).<sup>3</sup> Another frequent error is writing

```

if(i==0 & j==2) // NO!
...

```

---

<sup>3</sup>When converting a numeric value to boolean, any non-zero value is considered [true](#), whereas 0 is [false](#).

instead of

```
if (i==0 && j==2)
 ...
```

This does not result in a compilation error only because single `&` exists. It performs the bitwise operator "and" on integers. It is defined in this manner: writing `a&b` amounts to consider the binary expression of *a* and *b* then to perform an "and" bit by bit (with the rule `1&1` gives 1; `1&0`, `0&1` and `0&0` give 0). For instance: `13&10` is 8 since in binary `1101&1010` is `1000`.

There is also other bitwise manipulation operators:

| symbol                | usage                   | name         | result                                                                                    | example                    |
|-----------------------|-------------------------|--------------|-------------------------------------------------------------------------------------------|----------------------------|
| <code>&amp;</code>    | <code>a&amp;b</code>    | and          | <code>1&amp;1=1</code> , else 0                                                           | <code>13&amp;10=8</code>   |
| <code> </code>        | <code>a b</code>        | or           | <code>0 0=0</code> , else 1                                                               | <code>13 10=15</code>      |
| <code>^</code>        | <code>a^b</code>        | exclusive or | <code>1^0=0^1=1</code> , else 0                                                           | <code>13^10=7</code>       |
| <code>&gt;&gt;</code> | <code>a&gt;&gt;n</code> | right shift  | shift bits of a n times to the right at fills at left with 0 (the n right first are lost) | <code>13&gt;&gt;2=3</code> |
| <code>&lt;&lt;</code> | <code>a&lt;&lt;n</code> | left shift   | shift bits of a n times to the left and fills at right with 0                             | <code>5&lt;&lt;2=20</code> |
| <code>~</code>        | <code>~a</code>         | complement   | <code>~1=0</code> , <code>~0=1</code>                                                     | <code>~13=-14</code>       |

Remarks

- These instructions are very fast since simple for the preprocessor.
- The fact that `a^b` has meaning is also a source of bugs (**it is not the power function!**)
- The result of `~` depends on the type: if for example *i* is an unsigned integer on 8 bits worth 13, then `~i` is 242, since `~00001101` is `11110010`.

In practice, all that is not meant to decorate or look smart, but to manipulate numbers bit by bit. It happens that we use an `int` to memorize some properties and saving memory, by using the fact that property *n* is true if the *n*<sup>th</sup> bit of the integer is 1. A single 32-bit integer can then store up to 32 properties where it would have taken 32 variables of type `bool`. Here is how to use the operators above to manipulate said bits:

|                                      |                         |
|--------------------------------------|-------------------------|
| <code>i =(1&lt;&lt;n)</code>         | set to 1 the bit n of i |
| <code>i&amp;=~(1&lt;&lt;n)</code>    | set to 0 the bit n of i |
| <code>i^=(1&lt;&lt;n)</code>         | invert bit n of i       |
| <code>if (i&amp;(1&lt;&lt;n))</code> | true if bit n of i is 1 |

There are other frequent uses of bitwise operators, not for space saving but for speed:

|                           |                                                      |
|---------------------------|------------------------------------------------------|
| <code>(1&lt;&lt;n)</code> | is $2^n$ (instead of <code>int(pow(2,n))</code> )    |
| <code>(i&gt;&gt;1)</code> | computes $i/2$ quickly                               |
| <code>(i&gt;&gt;n)</code> | computes $i/2^n$ quickly                             |
| <code>(i&amp;255)</code>  | computes $i\%256$ quickly (idem for all powers of 2) |

## 13.3 Conditional values

It happens often that we have to choose between two values depending on the result of a test. A compact construct is then:

```
(test)? val1 : val2
```

that is worth `val1` if `test` is true and `val2` otherwise. Hence,

```
if (x>y)
 maxi=x;
else
 maxi=y;
```

can be replaced by:

```
maxi=(x>y)?x:y;
```

This convenient construct should not be abused, it would make the program unreadable!

## 13.4 Loops and break

We have already met at section 8.4 the instruction `continue` that *jumps to the end of loop and goes to next round*. Also useful, the command `break` *exits the loop, ignoring all the rest of the iteration*. Hence the code

```
bool stop=false;
for (int i=0; i<N && !stop; i++) {
 A;
 if (B)
 stop=true;
 else {
 C;
 if (D)
 stop=true;
 else {
 E;
 }
 }
}
```

is written more naturally:

```
for (int i=0; i<N; i++) {
 A;
 if (B)
 break;
 C;
 if (D)
 break;
 E;
}
```

Recurrent questions from beginners:

1. `break` does not exit an `if`!

```

if (...) {
 ...;
 if (...)
 break; // NO! Does not exit if! (but possibly
 // a for that would be around...)
 ...
}

```

2. `break` exits only the current loop, not outer loops:

```

1 for(int i=0; i<N; i++) {
2 ...
3 for(int j=0; j<M; j++) {
4 ...
5 if (...)
6 break; // ends loop in j and goes to
7 // line 10 (not line 12)
8 ...
9 }
10 ...
11 }
12 ...

```

3. `break` and `continue` work also with `while` and `do...while`.

## 13.5 Static variables

We are sometimes led to use a global variable to memorize permanently a value that interests a single function:

```

// Function random calling srand() directly at first call
bool first=true;
double random() {
 if(first) {
 first=false;
 srand((unsigned int)time(0));
 }
 return double(rand())/RAND_MAX;
}

```

The risk is that all the rest of the program sees this variable and mixes it with another global variable. It is possible to *hide this variable in the function* with the keyword `static` before the variable:

```

// Function random calling srand() directly at first call
// with global variable hidden inside

```

```
double random() {
 static bool first=true; // Do not forget static!
 if (first) {
 first=false;
 srand((unsigned int)time(0));
 }
 return double(rand())/RAND_MAX;
}
```

Remember that it is really a global variable and not a local one. A local variable would die at function exit, which would not be the desired effect in the example!

It is also possible to hide a global variable in a class, still with `static`. We won't show how, but it is not difficult.

## 13.6 `const` and arrays

We saw unwillingly `const char *` as argument of some functions (file opening for example). We thus have to explain it: *it does not mean a pointer to `char` that would be constant, but a pointer to `char` that are constant!* We have to remember that:

**Placed before an array, `const` means that the elements of the array cannot be modified.**

This possibility to say that an array cannot be modified is all the more important that an array is always passed by reference: without `const`, we could not ensure the preservation of the values:

```
void f(int t[4]) {
 ...
}

void g(const int t[4]) {
 ...
}

void h(const int* t, int n) {
 ...
}

...
int a[4];
f(a); // may modify a[]
g(a); // does not modify a[]
h(a,4); // does not modify a[]
...
```

## 13.7 Reference card

| Reference Card (1/6)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Loops</b></p> <ul style="list-style-type: none"> <li>do {<br/>    ...<br/>} while(!ok);</li> <li>int i=1;<br/>while(i&lt;=100) {<br/>    ...<br/>    i=i+1;<br/>}</li> <li>for(int i=1;i&lt;=10;i++)<br/>    ...</li> <li>for(int i=1,j=10;j&gt;i;<br/>    i=i+2,j=j-3)<br/>    ...</li> <li>for (int i=...)<br/>    for (int j=... ) {<br/>        //skip case i==j<br/>        if (i==j)<br/>            continue;<br/>        ...<br/>    }</li> <li>for (int i=... ) {<br/>    ...<br/>    if (t[i]==s){<br/>        // exit loop<br/>        break;<br/>    }<br/>    ...<br/>}</li> </ul>                                                                                                                                                              | <ul style="list-style-type: none"> <li>Negation: !</li> <li>Combinations: &amp;&amp;   </li> <li>if (i==0) j=1;</li> <li>if (i==0) j=1;<br/>    else j=2;</li> <li>if (i==0) {<br/>    j=1;<br/>    k=2;<br/>}</li> <li>bool t=(i==0);<br/>if (t)<br/>    j=1;</li> <li>switch (i) {<br/>    case 1:<br/>        ...;<br/>        ...;<br/>        break;<br/>    case 2:<br/>        ...;<br/>    case 3:<br/>        ...;<br/>        break;<br/>    default:<br/>        ...;<br/>}</li> <li>mx=(x&gt;y)?x:y;</li> </ul> | <pre> } void display(int x,              int y) {     if (x&lt;0    y&lt;0)         return;     if (x&gt;=w    y&gt;=h)         return;     DrawPoint(x,y,RED); } </pre> <ul style="list-style-type: none"> <li>Call:<br/>int f(int a) { ... }<br/>int g() { ... }<br/>...<br/>int i=f(2),j=g();</li> <li>References:<br/>void swap(int&amp; a,<br/>          int&amp; b){<br/>    int tmp=a;<br/>    a=b;b=tmp;<br/>}</li> <li>Overload:<br/>int chance(int n);<br/>int chance(int a,<br/>          int b);<br/>double chance();</li> <li>Operators:<br/>vect operator+(<br/>          vect A,vect B) {<br/>    ...<br/>}</li> <li>Call stack</li> <li>Iterative/Recursive</li> <li>Constant references (avoid copy):<br/>void f(const obj&amp; x) {<br/>    ...<br/>}</li> <li>void g(const obj&amp; x) {<br/>    f(x); // OK<br/>}</li> </ul> |
| <p><b>Keys</b></p> <ul style="list-style-type: none"> <li>Debug: F5 </li> <li>Step over: F10 </li> <li>Step inside: F11 </li> <li>Indent: Ctrl+A, Ctrl+I</li> <li>Switch source/header: F4</li> <li>Switch decl./def.: F2</li> <li>Step out: Maj+F11 </li> <li>Gest. tâches: Ctrl+Maj+Ech</li> </ul> <p><b>Tests</b></p> <ul style="list-style-type: none"> <li>Comparison:<br/>== != &lt; &gt; &lt;= &gt;=</li> </ul> | <p><b>Functions</b></p> <ul style="list-style-type: none"> <li>Definition:<br/>int plus(int a,int b){<br/>    int c=a+b;<br/>    return c;<br/>}</li> <li>void display(int a) {<br/>    cout &lt;&lt; a &lt;&lt; endl;<br/>}</li> <li>Declaration:<br/>int plus(int a,int b);</li> <li>Return:<br/>int sign(double x) {<br/>    if (x&lt;0)<br/>        return -1;<br/>    if (x&gt;0)<br/>        return 1;<br/>    return 0;</li> </ul>                                                                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

| Reference Card (2/6)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Default values:</b></p> <pre>void f(int a,int b=0); void g() {     f(12); // f(12,0);     f(10,2); // f(10,2); } void f(int a,int b) {     // ... }</pre> <p><b>Inline (fast call):</b></p> <pre>inline double     sqr(double x) {         return x*x;     }     ...     double y=sqr(z-3);</pre> <p><b>Return as reference:</b></p> <pre>int i; // global var int&amp; f() {     return i; } ... f()=3; // i=3!</pre> <hr/> <p><b>Structures</b></p> <ul style="list-style-type: none"> <li>• struct Point {             double x,y;             Color c;         };         ...         Point a;         a.x=2.3; a.y=3.4;         a.c=RED;         Point b={1,2.5,BLUE};</li> <li>• A structure is an objet fully public (→ see objects!)</li> </ul> <hr/> <p><b>Variables</b></p> <ul style="list-style-type: none"> <li>• Definition:             <pre>int i; int k,l,m;</pre> </li> <li>• Assignment:             <pre>i=2; j=i; k=l=3;</pre> </li> <li>• Initialization:             <pre>int n=5,o=n;</pre> </li> <li>• Constants:             <pre>const int s=12;</pre> </li> </ul> | <ul style="list-style-type: none"> <li>• <b>Scope:</b> <pre>int i; // i=j; forbidden! int j=2; i=j; // OK! if (j&gt;1) {     int k=3;     j=k; // OK! } //i=k; forbidden!</pre> </li> <li>• <b>Types:</b> <pre>int i=3; double x=12.3; char c='A'; string s="hop"; bool t=true; float y=1.2f; unsigned int j=4; signed char d=-128; unsigned char d=25; complex&lt;double&gt;     z(2,3);</pre> </li> <li>• <b>Global variables:</b> <pre>int n; const int m=12; void f() {     n=10; // OK     int i=m; // OK     ... }</pre> </li> <li>• <b>Conversion:</b> <pre>int i=int(x),j; float x=float(i)/j;</pre> </li> <li>• <b>Stack/Heap</b></li> <li>• <b>Enumerated type:</b> <pre>enum Dir{N,E,S,W}; void advance(Dir d);</pre> </li> <li>• <b>Staic variables:</b> <pre>int f() {     static bool once=true;     if (once) {         once=false;         ...     }     ... }</pre> </li> </ul> <hr/> <p><b>Objects</b></p> <ul style="list-style-type: none"> <li>• struct obj {             int x; // field</li> </ul> | <pre>int f(); // method int g(int y); }; int obj::f() {     int i=g(3); // my g     int j=x+i; // my x     return j; } ... int main() {     obj a;     a.x=3;     int i=a.f();</pre> <ul style="list-style-type: none"> <li>• class obj {             int x,y;             void mine();         public:             int z;             void for_all();             void another(obj A);         };         void obj::mine() {             x=..; // OK             ..=y; // OK             z=..; // OK         }         void obj::for_all(){             x=..; // OK             mine(); // OK         }         void another(obj A) {             x=A.x; // OK             A.mine(); // OK         }         ...         int main() {             obj A,B;             A.x=..; //NO             A.z=..; //OK             A.mine(); //NO             A.for_all(); //OK             A.another(B); //OK</li> <li>• class obj {             obj operator+(obj B);         };         ...         int main() {             obj A,B,C;             C=A+B;             // C=A.operator+(B)</li> </ul> |

| Reference Card (3/6)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Constant methods:</b></p> <pre>void obj::f() const{     ... } void g(const obj&amp; x){     x.f(); // OK }</pre> <p><b>Constructor:</b></p> <pre>class point {     int x,y; public:     point(int X,int Y); }; point::point(int X,              int Y){     x=X;     y=Y; } ... point a(2,3);</pre> <p><b>Empty constructor:</b></p> <pre>obj::obj() {     ... } ... obj a;</pre> <p><b>Temporary objects:</b></p> <pre>vec vec::operator+(     vec b) {     return vec(x+b.x,               y+b.y); } ... c=vec(1,2) +f(vec(2,3));</pre> <p><b>Destructor:</b></p> <pre>obj::~obj() {     ... }</pre> | <p><b>Copy constructor:</b></p> <pre>obj::obj(const obj&amp; o) { ... }</pre> <p><b>Used by:</b></p> <pre>- obj b(a); - obj b=a; //Not obj b;b=a; - function argument - return value</pre> <p><b>Assignment:</b></p> <pre>obj&amp; obj::operator=(     const obj&amp;o){     ...     return *this; }</pre> <p><b>Objects with automatic dynamic allocation: Section 11.5</b></p> <p><b>Accessors:</b></p> <pre>class mat {     double *x; public:     double&amp; operator()         (int i,int j){         assert(i&gt;=0 ...);         return x[i+M*j];     }     double operator()         (int i,int j)const{         assert(i&gt;=0 ...);         return x[i+M*j];     }     ... }</pre> <hr/> <p><b>Separate compilation</b></p> <ul style="list-style-type: none"> <li>• #include "vect.h", also in vect.cpp</li> <li>• Functions: declarations in .h, definitions in .cpp</li> </ul> | <ul style="list-style-type: none"> <li>• Types: definitions in .h</li> <li>• Declare in .h only useful functions</li> <li>• #pragma once at beginning of header file</li> <li>• Do not cut too much...</li> </ul> <hr/> <p><b>STL</b></p> <ul style="list-style-type: none"> <li>• min,max,...</li> <li>• complex&lt;double&gt; z;</li> <li>• pair&lt;int,string&gt; p;             <pre>p.first=2; p.second="hop";</pre> </li> <li>• #include&lt;list&gt;             <pre>using namespace std; ... list&lt;int&gt; l; l.push_front(1); ...</pre> </li> <li>• if(l.find(3)!=l.end())             <pre>...</pre> </li> <li>• list&lt;int&gt;::             <pre>const_iterator it; for (it=l.begin();     it!=l.end();it++)     s+= *it;</pre> </li> <li>• list&lt;int&gt;::iterator it             <pre>for (it=l.begin();     it!=l.end();it++)     if (*it==2)         *it=4;</pre> </li> <li>• stack, queue, heap, map, set, vector...</li> </ul> |



| Reference Card (4/6)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Template</b></p> <ul style="list-style-type: none"> <li>• <b>Functions:</b><br/> <pre>// To put in // file that uses // or in .h template &lt;typename T&gt; T maxi(T a,T b) {     ... } ... // Type is found // alone! maxi(1,2); //int maxi(.2,.3); //double maxi("a","c");//string</pre> </li> <li>• <b>Objects:</b><br/> <pre>template &lt;typename T&gt; class pair {     T x[2]; public:     pair() {}     pair(T a,T b) {         x[0]=a;x[1]=b;     }     T add() const; }; ... template &lt;typename T&gt; T pair&lt;T&gt;::add()const{     return x[0]+x[1]; } ... // Type must be // precised! pair&lt;int&gt; a(1,2); int s=a.add(); pair&lt;double&gt; b; ...</pre> </li> <li>• <b>Multiple:</b><br/> <pre>template &lt;typename T,           typename S&gt; class hop {     ... }; ... hop&lt;int,string&gt; A; ...</pre> </li> <li>• <b>Integers:</b><br/> <pre>template &lt;int N&gt; class hop {</pre> </li> </ul> | <pre>.. }; ... hop&lt;3&gt; A; ...</pre> <hr/> <p><b>Input/Output</b></p> <ul style="list-style-type: none"> <li>• <pre>#include &lt;iostream&gt; using namespace std; ... cout &lt;&lt;"I="&lt;&lt;i&lt;&lt;endl; cin &gt;&gt; i &gt;&gt; j;</pre> </li> <li>• <pre>#include &lt;fstream&gt; using namespace std; ofstream f("hop.txt"); f &lt;&lt; 1 &lt;&lt; ' ' &lt;&lt; 2.3; f.close(); ifstream g("hop.txt"); if (!g.is_open()) {     return 1; } int i; double x; g &gt;&gt; i &gt;&gt; x; g.close();</pre> </li> <li>• <pre>do {     ... } while (!(g.eof()));</pre> </li> <li>• <pre>ofstream f; f.open("hop.txt");</pre> </li> <li>• <pre>double x[10],y; ofstream f("hop.bin",           ios::binary); f.write((const char*)x,         10*sizeof(double)); f.write((const char*)&amp;y,         sizeof(double)); f.close(); ifstream g("hop.bin",           ios::binary); g.read((char*)x,         10*sizeof(double)); g.read((const char*)&amp;y,         sizeof(double)); g.close();</pre> </li> <li>• <pre>string s; ifstream f(s.c_str());</pre> </li> <li>• <pre>#include &lt;sstream&gt; using namespace std;</pre> </li> </ul> | <pre>stringstream f; // String to int f &lt;&lt; s; f &gt;&gt; i; // int to string f.clear(); f &lt;&lt; i; f &gt;&gt; s; ⇔ s = std::to_string(i);</pre> <hr/> <ul style="list-style-type: none"> <li>• <pre>ostream&amp; operator&lt;&lt;(     ostream&amp; f,     const point&amp;p){     f&lt;&lt;p.x&lt;&lt;' '&lt;&lt;p.y;     return f; } istream&amp; operator&gt;&gt;(     istream&amp; f,point&amp; p){     f&gt;&gt;p.x&gt;&gt;p.y;     return f; }</pre> </li> </ul> <hr/> <p><b>Advice</b></p> <ul style="list-style-type: none"> <li>• Errors/warnings: click.</li> <li>• Indent!</li> <li>• Fix warnings.</li> <li>• Use the debugger.</li> <li>• Write functions.</li> <li>• Arrays: not to translate math formula!</li> <li>• Make structures.</li> <li>• Do separate source files</li> <li>• The .h must be enough for the user (who must not look into .cpp)</li> <li>• Don't abuse recursion.</li> <li>• Don't forget delete.</li> <li>• Compile regularly.</li> <li>• <pre>#include &lt;cassert&gt; ... assert(x!=0); y=1/x;</pre></li> <li>• Make objects.</li> <li>• Do not always make objects!</li> <li>• Think interface / implementation / usage.</li> </ul> |

| Reference Card (5/6)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Misc.</b></p> <ul style="list-style-type: none"> <li>• <code>i++;</code><br/><code>i--;</code><br/><code>i-=2;</code><br/><code>j+=3;</code></li> <li>• <code>j=i%n;</code> // Modulo</li> <li>• <code>#include &lt;cstdlib&gt;</code><br/>...<br/><code>i=rand()%n;</code><br/><code>x=rand()/</code><br/>    <code>double(RAND_MAX);</code></li> <li>• <code>#include &lt;ctime&gt;</code><br/>// Single call<br/><code>srand((unsigned int)</code><br/>    <code>time(0));</code></li> <li>• <code>#include &lt;cmath&gt;</code><br/><code>double sqrt(double x);</code><br/><code>double cos(double x);</code><br/><code>double sin(double x);</code><br/><code>double acos(double x);</code></li> <li>• <code>#include &lt;string&gt;</code><br/>using namespace std;<br/><code>string s="hop";</code><br/><code>char c=s[0];</code><br/><code>int l=s.size();</code><br/><code>if (s1==s1) ...</code><br/><code>if (s1!=s2) ...</code><br/><code>if (s1&lt;s2) ...</code><br/><code>size_t i=s.find('h'),</code><br/>    <code>j=s.find('h',3);</code><br/>    <code>k=s.find("hop");</code><br/>    <code>l=s.find("hop",3);</code><br/><code>a="how";</code><br/><code>b="are you?";</code><br/><code>txt=a+" "+b;</code><br/><code>s1="one two three";</code><br/><code>s2=string(s1,4,3);</code><br/><code>getline(cin,s);</code><br/><code>getline(cin,s,':');</code><br/><code>const char *t=s.c_str();</code></li> <li>• <code>#include &lt;ctime&gt;</code><br/><code>s=double(clock())</code><br/>    <code>/CLOCKS_PER_SEC;</code></li> <li>• <code>#include &lt;cmath&gt;</code><br/><code>double pi=M_PI;</code></li> <li>• <b>Bitwise operators</b><br/>    <b>and:</b>           <code>a&amp;b</code><br/>    <b>or:</b>             <code>a b</code><br/>    <b>xor:</b>            <code>a^b</code><br/>    <b>right shift:</b>   <code>a&gt;&gt;n</code><br/>    <b>left shift:</b>    <code>a&lt;&lt;n</code><br/>    <b>complement:</b>   <code>~a</code><br/>    <b>examples:</b></li> </ul> | <pre> set(i,1):   i =(1&lt;&lt;n) reset(i,1): i&amp;=~(1&lt;&lt;n) test(i,1):  if (i&amp;(1&lt;&lt;n)) flip(i,1):  i^=(1&lt;&lt;n) </pre> <hr/> <p><b>Frequent errors</b></p> <ul style="list-style-type: none"> <li>• <b>No definition of function inside a function!</b></li> <li>• <code>int q=r=4;</code> // NO!</li> <li>• <code>if (i=2)</code> // NO!<br/><code>if i==2</code> // NO!<br/><code>if (i==2) then</code> // NO!</li> <li>• <code>for(int i=0,i&lt;100,i++)</code><br/>    // NO!</li> <li>• <code>int f() {...}</code><br/><code>int i=f;</code> // NO!</li> <li>• <code>double x=1/3;</code> // NO!<br/><code>int i,j;</code><br/><code>x=i/j;</code> // NO!<br/><code>x=double(i/j);</code> //NO!</li> <li>• <code>double x[10],y[10];</code><br/><code>for(int i=1;i&lt;=10;i++)</code><br/>    <code>y[i]=2*x[i];</code>//NO</li> <li>• <code>int n=5;</code><br/><code>int t[n];</code> // NO</li> <li>• <code>int f()[4] { // NO!</code><br/>    <code>int t[4];</code><br/>    ...<br/>    <code>return t;</code> // NO!</li> <li>• <code>int t[4]; t=f();</code></li> <li>• <code>int s[3]={1,2,3},t[3];</code><br/><code>t=s;</code> // NO!</li> <li>• <code>int t[2];</code><br/><code>t={1,2};</code> // NO!</li> <li>• <code>struct Point {</code><br/>    <code>double x,y;</code><br/>} // NO!</li> <li>• <code>Point a;</code><br/><code>a={1,2};</code> // NO!</li> <li>• <code>#include "tp.cpp"</code>//NO</li> <li>• <code>int f(int t[][]);</code>//NO<br/><code>int t[2,3];</code> // NO!<br/><code>t[i,j]=...;</code> // NO!</li> <li>• <code>int* t;</code><br/><code>t[1]=...;</code> // NO!</li> <li>• <code>int* t=new int[2];</code><br/><code>int* s=new int[2];</code><br/><code>s=t;</code> // lost s!<br/><code>delete[] t;</code><br/><code>delete[] s;</code>//Crash!</li> </ul> | <ul style="list-style-type: none"> <li>• <code>int *t,s;</code> // s is int<br/>    // not int*<br/><code>t=new int[n];</code><br/><code>s=new int[n];</code>// NO!</li> <li>• <code>class vec {</code><br/>    <code>int x,y;</code><br/>public:<br/>    ...<br/>};<br/>...<br/>    <code>vec a={2,3};</code> // NO</li> <li>• <code>vec v=vec(1,2);</code>//NO<br/><code>vec v(1,2);</code> // Yes</li> <li>• <code>obj* t=new obj[n];</code><br/><code>delete t;</code>// missing []</li> <li>• //NO!<br/><code>void f(int a=2,int b);</code></li> <li>• <code>void f(int a,int b=0);</code><br/><code>void f(int a);</code>// NO!</li> <li>• <b>Not anything inline!</b></li> <li>• <code>int f() {</code><br/>    ...<br/>}<br/><code>f()=3;</code> // HORROR!</li> <li>• <code>int&amp; f() {</code><br/>    <code>int i;</code><br/>    <code>return i;</code><br/>}<br/><code>f()=3;</code> // NO!</li> <li>• <code>if (i&gt;0 &amp; i&lt;n)</code> // NO<br/><code>if (i&lt;0   i&gt;n)</code> // NO</li> <li>• <code>if (...) {</code><br/>    ...<br/>    <code>if (...)</code><br/>        <code>break;</code> // No,<br/>        // loops only<br/>}</li> <li>• <code>for (i ...)</code><br/>    <code>for (j ...) {</code><br/>        ...<br/>        <code>if (...)</code><br/>            <code>break;</code>//NO, exit<br/>        // loop j only</li> <li>• <code>int i;</code><br/><code>double x;</code><br/><code>j=max(i,0);</code>//KO<br/><code>y=max(x,0);</code>//NO<br/>    // 0.0 and not 0: max<br/>    // is an STL template</li> </ul> <hr/> <p><b>Imagine++</b></p> <ul style="list-style-type: none"> <li>• See documentation...</li> </ul> |

| Reference Card (6/6)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                         |                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Arrays</b></p> <ul style="list-style-type: none"> <li>• <b>Definition:</b> <pre>- double x[5],y[5];   for(int i=0;i&lt;5;i++)     y[i]=2*x[i];</pre> </li> <li>- const int n=5;       <pre>int i[n],j[2*n];</pre> </li> <li>• <b>Initialization:</b> <pre>int t[4]={1,2,3,4}; string s[2]={"ab","c"};</pre> </li> <li>• <b>Assignment:</b> <pre>int s[3]={1,2,3},t[3]; for (int i=0;i&lt;3;i++)   t[i]=s[i];</pre> </li> <li>• <b>As parameter:</b> <pre>- void init(int t[4]){   for(int i=0;i&lt;4;i++)</pre> </li> </ul> | <pre>t[i]=0; }</pre> <ul style="list-style-type: none"> <li>- void init(int t[],       <pre>int n) {   for(int i=0;i&lt;n;i++)     t[i]=0; }</pre> </li> <li>• <b>Variable size:</b> <pre>int* t=new int[n]; ... delete[] t;</pre> </li> <li>• <b>As argument:</b> <pre>- void f(int* t,int n)   { t[i]=... }</pre> <pre>- void alloc(int*&amp; t){   t=new int[n]; }</pre> </li> </ul> | <ul style="list-style-type: none"> <li>• <b>2D:</b> <pre>int A[2][3]; A[i][j]=...; int A[2][3]=   {{1,2,3},{4,5,6}}; void f(int A[2][2]);</pre> </li> <li>• <b>2D in 1D:</b> <pre>int A[2*3]; A[i+2*j]=...;</pre> </li> <li>• <b>Variable size:</b> <pre>int *t,*s,n;</pre> </li> <li>• <b>As argument (end):</b> <pre>void f(const int* t,       int n) {   ...   s+=t[i]; // OK   ...   t[i]=...; // NO! }</pre> </li> </ul> |



# Appendix A

## Practicals

Note: corrections are available on the course web page after each practical session.

### A.1 Programming environment

#### A.1.1 Hello, World!

1. *Connection:*


Log in on your machine.

2. *Project:*

Download archive `Tp1_Initial.zip` from the course web page, decompress it. The file `CMakeLists.txt` describes the project. The line of interest to us is the following:

```
add_executable(Tp1 Tp1.cpp)
```

It indicates that our program will be called `Tp1` (`Tp1.exe` under Windows) and that the source code to build it is the file `Tp1.cpp`.

`Qt Creator`  is the programming environment that we will use, it has the advantage to work on all main platforms (Linux, Windows or Mac) and to control directly `CMake`.<sup>1</sup> To start, launch `Qt Creator` and choose “Open Project” to fetch the file `CMakeLists.txt`. Look at the window bottom tab `6 General Messages`,<sup>2</sup> that should print a lot of messages: it is the result of the file read and its interpretation by `CMake`.

3. *Programming:* insert a comment line with your name at the top of file `Tp1.cpp`. Comments are introduced with `//` and go until the line end. They are ignored by the compiler, but useful for the programmer. Insert then the command


```
cout << "Hello" << endl;
```

on a line before `return 0;` in file `Tp1.cpp`.

---

<sup>1</sup>Most other environments, such as Visual Studio (Windows) and Xcode (Mac), do not know `CMake`, and we must use beforehand `CMake` to convert into a native project for the environment.

<sup>2</sup>If you do not see it, you can display it with the small arrow at the right of the tabs

4. *Generation*: Push the hammer button  at bottom left to generate the program. Look in tab 4 “Compile Output”, it says it created the file `Tp1.cpp.o`. Check with the file explorer (outside Qt Creator) that the file is indeed present in the “build” folder. Observe that the program `Tp1` is also there.
5. *Execution*:
  - (a) Launch the program with the green arrow in Qt Creator, observe the result in tab 3 “Application Output”.
  - (b) Check that we created an independent program that we can launch from the command line:
    - Find the build folder created by QtCreator (see Section [A.1.6](#)).
    - Open a terminal (under Windows: powershell)
    - Go to the build folder: `cd C:\...\build-Tp1...` (learn to take profit of automatic completion with tabulation key `TAB`, just above Caps lock). Under Linux and Mac, the path is different, and replace all `\` with `/` to delimit folders. You can also copy-paste the path from QtCreator.
    - Check the presence of `Tp1.exe` or `Tp1` with the command `ls`.
    - Type `./Tp1`. Under Windows, you have to adjust the search path for shared libraries (Section [A.1.6](#)).
6. *Compress*:

With a right mouse click on folder `Tp1`, build a compressed archive `Tp1.zip` (or `Tp1.7z` if proposed). Such an archive, comprising all files necessary to compilation, will be the format used to upload your exercises and practical session returns under Educnet.

Note that with CMake we have two folders:

- The *source* folder containing files `Tp1.cpp` and `CMakeLists.txt`;
- The *build* folder, with name chosen by Qt Creator.

The most important one is the first, since the second can always be generated again with CMake. Send your instructor only the source folder, the build will be done on his machine. Your *build* folder is useless on another machine if its system is not the same. When you have finished a project, do not hesitate to clean up by removing your *build* folder, but keep cautiously your *source* folder, it is the result of your hard work. Though CMake allows using the same folder for both, it is better to avoid it, to separate clearly sources and automatically generated files. To understand a little how these tools coordinate, you can read Appendix C of the lecture notes.

The precious folder containing your work is the *source* folder, `Tp1_Initial`. The disposable one is the *build* folder, `build-Tp1_Initial-...`

7. Check that everything could be restarted from scratch: quit Qt Creator, erase your source and build folders, extract the source folder from your archive; if a file `CMakeLists.txt.user` remains, it was created by Qt Creator, remove it before launching Qt Creator and opening the project.

## A.1.2 First errors

You will make many,<sup>3</sup> it is perfectly normal. Let's get used to it right now.

### 1. *Modify the program:*

Modify the program by changing for example the displayed sentence.

- (a) Test a new generation/execution. Check that Qt Creator saves the file automatically (or proposes to save it) before generating.

**Launching directly an execution saves and builds automatically.**

Knowing that, the hammer tool may seem useless. However, it is good when coding to check simply compilation very frequently, even before taking care of execution because there is nothing to observe yet.

### 2. *Compilation errors*

Provoke, observe and learn to recognize some compilation errors:

- (a) `includ` instead of `include`
- (b) `iostrem` instead of `iostream`
- (c) Forget the `;` after `std`
- (d) `inte` instead of `int`
- (e) `cou` instead of `cout`
- (f) Forget the double quotes `"` ending the string `"Hello_..."`
- (g) Add a line `i=3;` before `return`.

Talking about errors, it is useful to discover that:

**Double-clicking on an error message (Tab 1 "Issues") shows in the editor the code line where it occurs.**

Note that tab 1 is only an automatic and clickable extract of tab 4, the full error message is found in tab 4. Sometimes, the automatic extraction may have defects and the error message in tab 1 is not clear since incomplete.

### 3. *Linker errors*

It is a bit early to have the linker fail,<sup>4</sup> but it will happen at some point. It is still necessary to know the difference between linker and compiler errors. In general, the linker will notify an error if it does not find a function or some variables because some object file or a library is missing. It is also an error if it finds twice the same function...

<sup>3</sup>Even the expert programmer will make some.

<sup>4</sup>It will happen more often when we separate our source in several files.

- (a) Add a line `f(2);` before `return` and build. It is for now a *compilation* error (the compiler does not know what `f` designates).
- (b) Fix the compilation error by adding the line (for the moment "magic") `void f(int i);` before the line with `main`. Build the program: the *linker* complains about the absence of the function `f()` used by `main()`.
- (c) Remove now the call to `f` in `main()`. Everything works again. Rename `main` as `main2`. The absence of function `main` will be noticed by the linker, and it will refuse to build the program.

#### 4. Indents:

With all these changes, the code may not correctly "indented" anymore. It is still essential for a good understanding and to spot some parenthesis error, curly brace, etc. The menu `Edit/Advanced` includes an option to reindent automatically.

**To spot errors, always indent. Ctrl+I = indent the selected zone. Ctrl+A, Ctrl+I = indent all file.**

#### 5. Warnings of the compiler

Modifying the `main()`, provoke some warnings.<sup>5</sup>

- (a) `int i;`  
`i=2.5;`  
`cout << i << endl;`  
Execute to see the result.
- (b) `int i;`  
`i=4;`  
`if(i=3) cout << "hello" << endl;`  
Launch!
- (c) Add `exit;` as first instruction of `main`. Trying to call a function without arguments can happen (and does nothing)! Launch to see. Correct by putting `exit(0);` (The function `exit()` ends the program at once!)

Some warnings are not not meaningful (false positives) and are not programmer errors. The problem with not correcting them is that they can drown the true positives, you should thus try to fix *all* of them.

**It is advised not to leave warnings in your code! Correct them as they appear. A compiler option can even propose to consider them as errors and prevent the compilation to go through!**

### A.1.3 Debugger


**Knowing how to use the debugger is fundamental. It should be the first reflex in front of an incorrect program. It is a true investigative tool, simpler and more powerful than stuffing the code with additional instructions to spy its run.**




<sup>5</sup>Warnings are not errors: they may be present to warn the programmer about possible mistakes, but the sense for the compiler is clear. Hence, depending on compiler options, the warnings may not appear.



1. Type the following function `main`.

```
int main()
{
 int i, j, k;
 i=2;
 j=3*i;
 if (j==5)
 k=3;
 else
 k=2;
 return 0;
}
```

2. To use the debugger with Qt Creator, we need to compile in Debug mode. The mode is chosen by clicking on the small terminal  on top of the green start triangle. Observe that the variable `CMAKE_BUILD_TYPE` takes then value `Debug` in tab "Projects".
3. Launch the program "under the debugger control" with the green triangle button with superimposed bug. The program executes but nothing seems to happen.
4. Put a "breakpoint" by clicking on the left margin at the level of line `i=2`; then relaunch the debugger.
5. Advance in "Step over" with key `F10` or the corresponding button and observe the values of variables (in the special window on the top right or by placing the mouse cursor without click on the variable in source code).
6. Place a second breakpoint and note that `F5` continues the program until the next reached breakpoint.
7. Add `i=max(j,k)`; before the `return`. Use "Step into" or `F11` or the corresponding button when the cursor is on this line. Note the difference with `F10`: `max` is a function call and we indicate the debugger we want to follow what happens inside.
8. At last, see how machine code looks like: execute until some breakpoint then activate menu `Debug/Operate by Instruction`. We can also in the same menu see the registers of the microprocessor. It can happen we fall in the window "machine code" unwillingly when we debug a program for which we do not have the source file anymore. This display is actually very useful to the programmer to check what the compiler produced and check it optimizes well.
9. The registers are visible by going in the menu `Window then Views`. See Figure A.1.

|                     |            |   |                                                                                     |   |                    |
|---------------------|------------|---|-------------------------------------------------------------------------------------|---|--------------------|
|                     | <b>F5</b>  | = |  | = | <b>Debug</b>       |
| <b>Useful keys:</b> | <b>F10</b> | = |  | = | <b>Step over</b>   |
|                     | <b>F11</b> | = |  | = | <b>Step inside</b> |

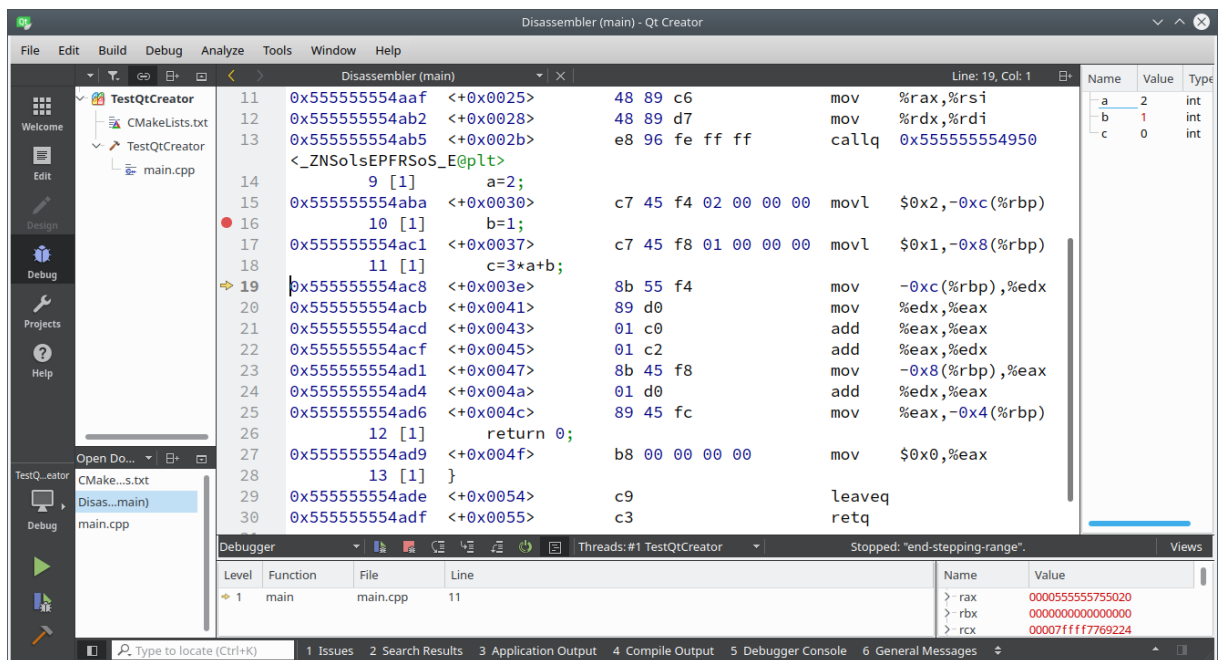


Figure A.1: Qt Creator showing machine code and registers.

### A.1.4 If there is time left

Download the supplementary program `Tp1_Final.zip` on the course web page (<http://imagine.enpc.fr/~monasse/Info>) play with it... and complete it with the right rebounds!

### A.1.5 Install Imagine++ at home

Go to <http://imagine.enpc.fr/~monasse/Imagine++>. Install on your laptop computer and try the supplementary program with Qt Creator.

### A.1.6 Launching the program from the command line

1. Find the build folder through button "Projects" (Figure A.2).
2. For Windows users: to open the PowerShell at the correct location, go there in the Explorer and shift-click on the blank background, Figure A.3.
3. For Windows users: the program depends on shared libraries (files with extension `dll`) whose location is not remembered by the program. As they are not in the build folder, you need to give the search path, see Figure A.4.

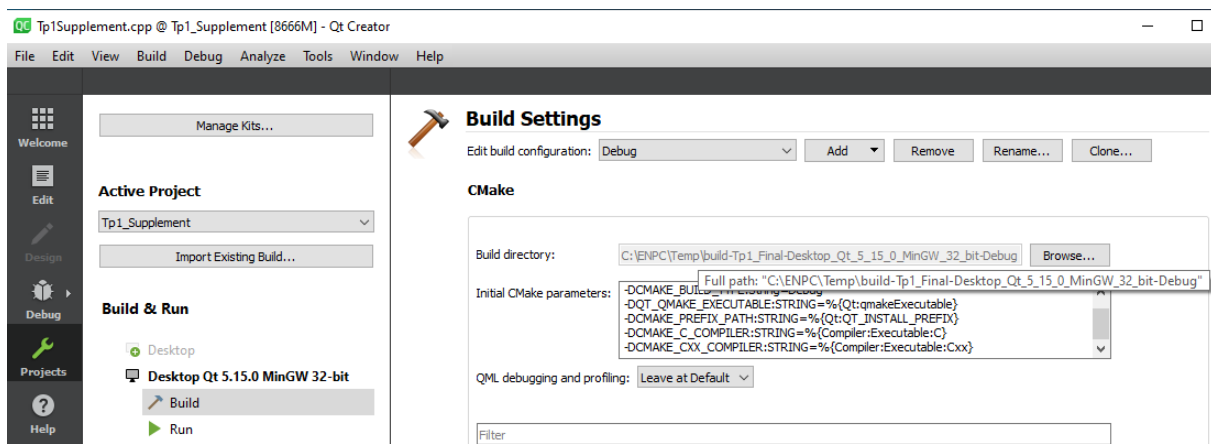


Figure A.2: Finding the build folder in QtCreator

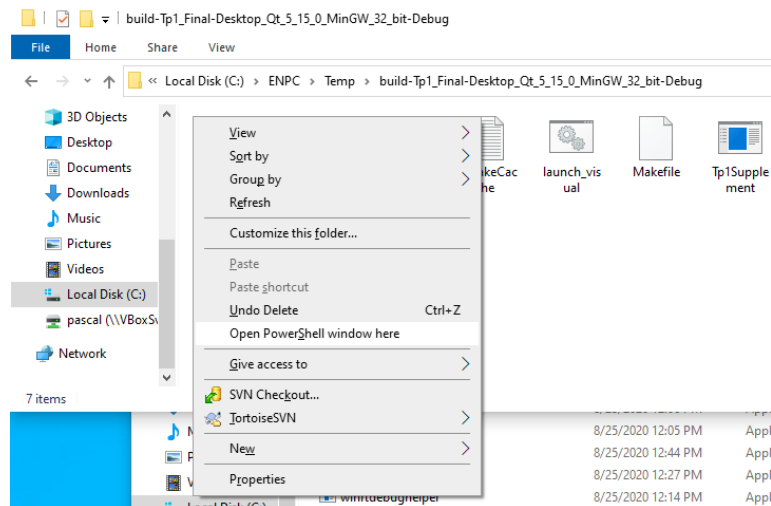


Figure A.3: Opening the PowerShell directly from the build folder (Windows).

## A.2 Variables, loops, conditions, functions

### A.2.1 First program with functions

#### 1. Get the example program:

Download archive `Tp2.zip` on webpage, extract the files and open the project by opening the `CMakeLists.txt` of `Tp2_Initial` from Qt Creator. Check project `Hop` whose source looks like:

```

Windows PowerShell
PS C:\ENPC\Temp\build-Tp1_Final-Desktop_Qt_5_15_0_MinGW_32_bit-Debug> $env:Path+='C:\ENPC\ProgramFiles\Qt\Tools\mingw810
32\bin\';
PS C:\ENPC\Temp\build-Tp1_Final-Desktop_Qt_5_15_0_MinGW_32_bit-Debug> $env:Path+='C:\ENPC\ProgramFiles\Qt\5.15.0\mingw81
32\bin\';
PS C:\ENPC\Temp\build-Tp1_Final-Desktop_Qt_5_15_0_MinGW_32_bit-Debug> .\Tp1Supplement.exe_

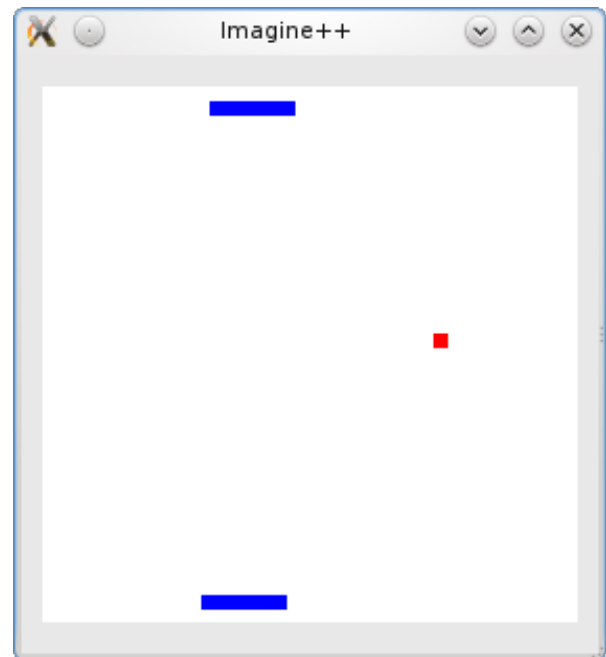
```

Figure A.4: Opening the PowerShell directly from the build folder (Windows). Adapt the paths to your local installation, and do not forget the semicolon ‘;’ at each line. The second path is needed for Tp1Supplement as it has graphical output.

```

1 #include <iostream>
2 using namespace std;
3
4 int plus(int a, int b) {
5 int c;
6 c=a+b;
7 return c;
8 }
9
10 void triple1(int a) {
11 a=a*3;
12 }
13
14 void triple2(int& a) {
15 a=a*3;
16 }
17
18 int main() {
19 int i, j=2, k;
20 i=3;
21 k=plus(i, j);
22 triple1(i);
23 triple2(i);
24 return 0;
25 }

```



Mini tennis...

## 2. Debugger:

Run the program step by step and study how variables change. You must use the program in Debug mode.

## A.2.2 First program with Imagine++

In this practical session and the following, we will use the graphics library of Imagine++ (see Appendix of lecture notes). It allows to manage easily the windowing system, drawings, and input/output of keyboard and mouse.

### 1. Starter program:

Input your name in the placeholder of file `tennis.cpp`. You must do that at each practical. Study the code of project Tennis:

```

1 #include <Imagine/Graphics.h>
2 using namespace Imagine;
3 ...
4 int main() {
5 // Open window
6 openWindow(256,256);
7 // Position and speed of ball
8 int xb=128,
9 yb=20,
10 ub=2,
11 vb=3;
12 // Main loop
13 while(true) {
14 // Display ball
15 fillRect(xb-3,yb-3,7,7,RED);
16 // Temporisation
17 milliSleep(20);
18 // Erase ball
19 fillRect(xb-3,yb-3,7,7,WHITE);
20 // Rebound
21 if(xb+ub>253)
22 ub=-ub;
23 // update ball position
24 xb+=ub;
25 yb+=vb;
26 }
27 endGraphics();
28 return 0;
29 }

```

**Do not care how function keyboard() works (it is a bit technical)**

Build then execute the program.<sup>6</sup> What happens?

2. *Online help.* At any moment, the key F1 allows accessing the documentation of the keyword under cursor. Test this with keywords `if`, `while` and `return`.

**Useful key: F1 = Access documentation**

3. *Understand how the program works:*  
Identify the main loop of the program. It proceeds so:

- (a) Display ball
- (b) Temporisation of a few milliseconds so that the ball does not move too fast

---

<sup>6</sup>Since the project has two executable programs, make sure the project Tennis is selected with the Project button on top of the start green triangle

- (c) Erase ball
- (d) Handle rebounds by updating speed
- (e) Update ball coordinates.

The line with instruction `while` may signal a warning. Can you understand why? What is the point of the condition by instruction `if`?

4. *Managing all rebounds:*

Complete the program so that all rebounds are handled. For example, inverse the horizontal component `ub` of the speed vector when the balls is going to meet the left or right border of the window.

5. *Global variables:*

Double the window height. Modify the ball size. This requires hanging the code at several locations. Instead of placing numerical variables “in the rock”, it is better to define variables for them. To make it clear, use constant global variables. For that, insert right after the two include lines the following code:

```
const int width = 256; // Window width
const int height = 256; // Window height
const int ball_size = 3; // Radius of ball
```

and reformulate numeric values in the program with these variables. The keyword `const` indicates that the variables cannot be modified after their initialization. Try to add the line `width=300;` at the start of function `main` and note that it makes a compilation error.

6. *Usage of functions:*

The ball is drawn twice in the loop, the first time in red and the second one in white to erase it. Here, drawing the ball requires a single line but could be much more if it had a more complex shape. Thus, to structure the program and make the code more readable while minimizing duplication, group the display of the ball and its erasure in a function `DrawBall` defined before the function `main`:

```
void DrawBall(int x, int y, Color col) {
 ...
}
```

In the same manner, define a function

```
void MoveBall(int &x, int &y, int &u, int &v)
```

to handle rebounds and movement of the ball.

### A.2.3 Tennis

We are going to make the program more fun by adding two rackets moving horizontally at top and bottom of the screen, and controlled by the keyboard.

1. *Display of rackets:*

Add in function `main` variables `xr1,yr1,xr2,yr2` dedicated to the position of both rackets. Then define a function `DrawRacket` taking example from `DrawBall`. Put the calls to these functions at appropriate locations in the loop.

2. *Keyboard handling:*

Management of the keyboard is ready for use in function `keyboard` whose content we will not comment. This function allows knowing directly if one of the keys of interest (s and d for the first racket, k and l for the second) are pressed or not. This function, `keyboard(int& sens1, int& sens2)`, returns in `sens1` and `sens2`, the values 0, -1 or 1 (0: no move, -1: to the left, 1: to the right).

3. *Racket motion:*

Code the motion of a racket in a function

```
void MoveRacket(int &x, int sens)
```

then call this function in the main loop for each racket. Naturally, make sure the rackets do not move outside the window.

4. *Rebounds on rackets:*

Take inspiration from the management of rebounds of the ball. Here, we have to check not only whether the ball will reach the top or bottom of the screen, but also if it is close enough in abscissa to the corresponding racket.

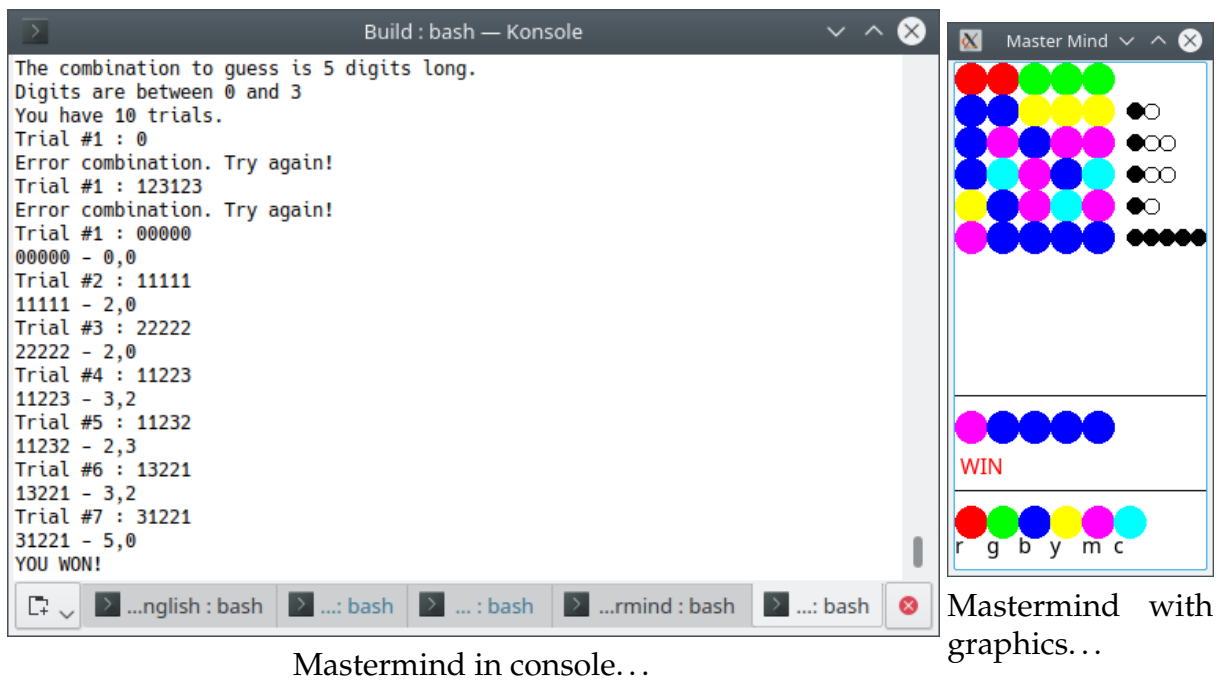
5. *Count and display score:*

Modify the function `MoveBall` so as to keep track of the score of the players and display it in the terminal.

6. *For the ones that have finished:*

During a rebound on the racket, modify the inclination of the trajectory of the ball as a function of the speed of the racket and of the location of the rebound.

You should have obtained a program looking like the one in the figure.



Mastermind in console...

Mastermind with graphics...

Figure A.5: The two forms of our game.

## A.3 Arrays

In this practical, we will program a game of Mastermind, where the player has to guess a combination generated randomly by the computer. The user has a predetermined number of trials. For each combination trial, the computer answers with two hints: the number of correctly placed pieces and the number of pieces of the right color but at a wrong position.

### A.3.1 Text Mastermind

1. *Get the initial code:*

Download the archive `Tp3_Initial.zip` from the course webpage, decompress it and open the project. Study the project `MasterMind`. Input your name in the placeholder of file `mastermind.cpp`.

2. *Represent a combination:*

We will take here a combination of 5 pieces of 4 different colors. The color of a piece will be represented by an integer in the range from 0 to 3. For a combination of 5 pieces, we will therefore use an array of 5 integers.

```
int combin[5]; // array of 5 integers
```

3. *Display a combination:*

Code a function displaying a combination on the screen. The simplest manner it to display the different numbers of the combinations on the same line one after the other.

4. *Generate a random combination:*

At the start of the game, the computer must generate randomly a combination



for the user to guess. We will use for that the functions declared in file `cstdlib`, notably the function `rand()` allowing to generate a random number between 0 and `RAND_MAX` (a constant, which may vary depending on the compiler). To get a number between 0 and `n`, we can proceed like this:

```
int x = rand()%n;
```

So that the sequence of generated numbers is not the same at each run of the program, it is necessary to initialize the generator with a variable seed. The simplest way is to use the current time. The function `time()` declared in file `ctime` gives access to it.

Finally, our function allows generating a combination:

```
#include <cstdlib>
#include <ctime>
using namespace std;

void generateCombination(int combin[5]) {
 for(int i=0; i<5; ++i)
 combin[i] = rand()%4; // calls to generator
}

srand((unsigned int)time(0)); // initialization
generateCombination(combin);
```

#### 5. Change the game complexity:

You will soon become experts in Mastermind and will want to increase the difficulty. It is enough to extend the length of the combination, or change the number of possible colors. It is quite easy if you thought of using a global constant for each quantity. If not, it is not too late to remedy it. Define for example:

```
const int nbcases = 5; // length of combination
const int nbcol = 4; // number of colors
```

Review the code you have written using these constants. It is very important to store constant parameters in variables, it saves a lot of effort when we need to change them.

#### 6. Input combination from the keyboard:

The following function, that we will admit for now, get a *character string* (type `string`) from the keyboard and fills the array `combi[]` with the `nbcases` first characters of the string.

```
void getCombination(int trial, int combi[nbcases]) {
 cout << "Trial#_" << trial << "_:_";
 string s;
 cin >> s;
 for(int i=0; i<nbcases; i++)
 combi[i]=s[i]-'0';
}
```

Beware, the `cin` does not work from the integrated terminal in Qt Creator (that is why the window is called *Application Output*, it does not support Input), we

need to use an external terminal. Go into mode Projects, section Run and select the button “Run in Terminal”.

In the framework of our Mastermind, we are going to modify this function so that it controls the input string is of the correct size and that its digits are between 0 and `nbcol-1`. The trial will be asked again until a valid combination is submitted. We will use the function `s.size()` that returns the size of a string `s` (the syntax of this function call<sup>7</sup> will be explained much later in the course...)

A small explanation on the line `s[i] - '0'`. Characters (of type `char`) are actually numeric values, their ASCII code. `'0'` gives the ASCII code of character zero. We can check it writing for example

```
cout << (int)'0' << endl;
```

(We need to cast to an integer, otherwise we ask to display a `char`, and then the computer prints the character associated to the ASCII code, 0!) This values is 48. It happens that digits from 0 to 9 have consecutive ASCII codes: `'0'`=48, `'1'`=49, `'9'`=57. The type string of variable `s` behaves like an array of `char`. Hence, `s[i] - '0'` is the integer 0 if `s[i]` is character 0, `'1' - '0'`=1 if it is the character 1, etc.

#### 7. *Combination analysis:*

We should now program a function comparing a given combination to the one to guess. It should return two values: the number of pieces of correct color at correct position, and, for the remaining pieces, the number of correct color but at a wrong position.

For example, if the secret combination is 02113:

00000 : 1 piece well placed (0xxxx), 0 piece wrongly placed (xxxxx)

20000 : 0 well placed (xxxxx), 2 wrongly placed (20xxx)

13133 : 2 well placed (xx1x3), 1 wrongly placed (1xxxx)

13113 : 3 well placed (xx113), 0 wrongly placed (xxxxx)

12113 : 4 well placed (x2113), 0 wrongly placed (xxxxx)

...

To begin and to be able to test right away the game, program first a function returning only the number of well placed pieces.

8. *Game loop:* We have now at disposal all the necessary foundation,<sup>8</sup> we just have to assemble them to have a mastermind game. Think also to add the detection of victory (all pieces are well placed), and of failure (number of trials have been reached, in which case the solution must be printed).
9. *Complete version:* Complete the function of combination comparison so that it returns also the number of wrongly placed pieces.

### A.3.2 Mastermind in graphics mode

The mastermind game we built has a rudimentary user interface. We will fix it by adding a graphical user interface (GUI).

<sup>7</sup>You could have expected something like `size(s)`.

<sup>8</sup>even if the function of combination analysis is not yet complete

1. *Study of initial project:*

Switch to project `MastermindGraphique`<sup>9</sup>

Graphic functions are already defined. They work following a principle of division of the window in lines. The function:

```
void displayCombination(int n, int combi[nbcases]);
```

prints the combination *combi* at line *n*. At the start, we leave at the top of the window as many free lines as the number of trials in order to display the game. We display at the bottom a mini user guide that sums up the correspondence between keys and colors.

2. *Graphic Mastermind:*

Reinsert in the project the functions of random generation of a combination and of comparison of two combinations. Reprogram the main loop of the game using the graphic display.

3. *Last improvement:*

We wish to be able to erase a color after its input in case of error. Study the functions

```
int keyboard(); and void getCombination(int,int []);
```

The first one already takes care of the backspace key, but not the second one that considers it as a wrong key. Modify the latter in consequence.

## 4. In both projects, make the program display from the start the supposedly secret combination, so that your instructor can check your program more easily.

---

<sup>9</sup>Under Qt Creator, select it with the icon representing a screen on the left of the window.



## A.4 Structures, Separate Files

*Warning:* In this practical, we will have bodies evolving under gravitation and undergoing elastic shocks. It is a long practical that will take several sessions. Actually, the second part is a reorganization of the first one. In section [A.4.4](#) some functions to use are presented, and in [A.4.5](#) their physics justification.

### A.4.1 Part 1: Single file

#### Translation motion

1. *To begin, study the project:*  
Download `Tp4_Initial.zip`, decompress it and launch Qt Creator. Input your name in the placeholder of file `gravitation1.cpp`. Browse the project, look out for global variables and the function `main` (do not waste time with the contents of functions already defined but not used yet). The program lets a point  $(x, y)$  evolve according to a constant translation  $(v_x, v_y)$ , and displays regularly a disk centered at this point. For that, to erase it, we store the position of the disk at the last display (in `ox` and `oy`); besides, two instructions beginning with `noRefresh` nest the graphics instructions to have a smoother display without flicker.
2. *Use a structure:*  
Modify the program so as to use a structure `Ball` keeping all information about the disk (position, speed, radius, color).
3. *Display:*  
Implement (and use!) a function `void DisplayBall(Ball D)` printing the disk `D`, and another `void EraseBall(Ball D)` that erases it.
4. *Make the disk move properly:*  
To have the position of the disk evolve, replace the corresponding instructions already present in the `main` by a call to a function modifying the coordinates of a `Ball`, by adding the speed of the `Ball` multiplied by a certain time step defined as a global variable ( $dt = 1$  for now).

#### Gravitation

5. *Evolution with acceleration:*  
Create (and use) a function that modifies the speed of a `Ball` so that it undergoes a constant acceleration  $a_x = 0$  and  $a_y = 0.0005$ . Hint: proceed as before, that is add  $0.0005 * dt$  to  $v_y$ .
6. *Add a sun:*  
We wish not to have a uniform gravitation. Add a field storing the mass to the structure `Ball`. Create a sun (of type `Ball`), yellow, fixed (null speed) at the window center, of mass 10 and radius 4 pixels (the mass of the moving planet is 1). Display it.

7. *Gravitational acceleration:*

Create (and use instead of the uniform gravitation) a function that takes as argument the planet and the sun, and that makes the speed of the planet evolve. Reminder in physics: remember since Newton that the acceleration is  $-G m_S/r^3 \vec{r}$ , here  $G = 1$  (meaning our mass is not expressed in kg but in another unit, which does not matter). You may need the square root function `double sqrt(double x)`. Do not forget the factor  $dt$ ... Let it run and observe. Initialize the position of the planet at  $x = \text{width}/3$ ,  $y = \text{height}/3$ ,  $v_x = v_y = 0$ . Note that the expression of acceleration becomes huge when  $r$  is near 0; we will therefore apply no acceleration when  $r$  is too small, such as  $r < r_1 + r_2$  with  $r_1$  and  $r_2$  the radii of the planet and the sun.

8. *Elliptic orbit:*

In a separate simulation (that is, another initialization and motion loop) start from the same position but put  $v_x = 0.3$ ,  $v_y = 0$ . You should observe an elliptic orbit of the planet. You can play with the initial speed to observe the different behaviors.

9. *Make the asteroid bounce:*

Have the asteroid undergo elastic shocks each time it goes too close to the sun, so that it never goes inside it (function `shockSimple`). To know if two spherical bodies will undergo a shock, use the function `Collision`. *Do not modify* the given functions `shock`, `shockSimple` and `collision`. Instead, you will create new ones taking as argument `Ball` (with the same name, it is fine as long as the function arguments are different, it is called function overload), they call the corresponding initial function.

10. *Finish first program:*

Handle the bounce in both loops. For the one with null initial speed, the planet should go out of the window soon after the bounce. Stop the loop when it happens, and let the program wait for a mouse click before proceeding with the second loop. The user must be warned by a message that a mouse click is expected.

## A.4.2 Reorganization, suns galore

11. *Several files*

We will have our project manage a second program. For that, copy/paste the file `gravitation1.cpp` to `gravitation2.cpp` in the same folder. Add into the file `CMakeLists.txt` the two lines adapted to the program `Gravitation2`, based on `gravitation2.cpp`. Make sure to run `CMake` (that should happen automatically if you modified the file from Qt Creator, otherwise look for the option in the Build menu). Notice that a new project tree for `Gravitation2` appears. From now on, we will work on this file. In the "Project" button above the "Run" button (green arrow), select `Gravitation2`, so that it is this program that is run.

## Tools functions

### 12. *A file of definitions...*

Add a new source file named `tools.cpp` to the project. Put there the initial functions of TP4 (shock, shockSimple and collision), removing them from `gravitation2.cpp`.

Do not forget to modify the arguments of `add_executable` in the `CMakeLists.txt` to add `tools.cpp` (source files are separated just by a blank space). Qt Creator should note the change when you save the file and should relaunch CMake, as you should observe in the tab "General Messages". If not, relaunch CMake through the menus.

If something goes wrong, which can happen if you apply changes to `CMakeLists.txt` and Qt Creator gets confused, save all files, quit Qt Creator, remove the file `CMakeLists.txt.user` and relaunch Qt Creator. That should fix the problem.

### 13. *...and a declaration file*

Add a header file `tools.h`. Insert the protection against double inclusion with `#pragma` once, instead of the other preprocessor instructions (beginning in `#`) put there by Qt Creator. Place the declarations of functions of `tools.cpp`, so as the definition of `dt`, removing it from `main`. Add at the beginning of `tools.cpp` and `gravitation2.cpp` the inclusion directive

```
#include "tools.h"
```

## Vectors

### 14. *Structure Vector:*

Create in a new file `vector.h` a structure representing a plane vector, with two members of type `double`. Do not forget the mechanism against double inclusion. Declare (but do not define yet) the operators and functions that follow:

```
Vector operator+(Vector a, Vector b); // Sum
Vector operator-(Vector a, Vector b); // Difference
double norm2(Vector a); // Euclidean norm
Vector operator*(Vector a, double lambda); // Mult. scalar
Vector operator*(double lambda, Vector a); // Mult. scalar
```

### 15. *Functions and operators with Vector:*

Create a new file `vector.cpp`. Put the `#include` for the header file and define the operators declared there (Reminder: `sqrt` is provided by the header file `<cmath>`). Once a version of `operator*` is defined, the second one can use the first in its definition.<sup>10</sup>

### 16. *Vector speed and vector position:*

Replace in `gravitation2.cpp` the speeds and positions by structures of type `Vector` (also in the definition of structure `Ball`). Use as much as possible the operators and functions defined in `vector.h`.

<sup>10</sup>It is elegant to do like that, it makes sure the two calls are consistent.

### Putting Ball apart

17. *Structure Ball:*

Move the definition of structure `Ball` in its own header file `ball.h`. Since `Ball` uses types `Vector` and `Color`, you need to add the lines:

```
#include <Imagine/Graphics.h>
using namespace Imagine;
#include "vector.h"
```

18. *Associated functions:*

Move all functions taking some `Ball` as argument to a new file `Ball.cpp`. At this point, in `gravitation2.cpp` the only function left should be `main`. Declare in `ball.h` the functions defined in `ball.cpp`. Add the `#include` necessary in this file and in `gravitation2.cpp` and perform necessary adaptations (for instance, if some functions use `width` or `height`, since these constants are defined only in `gravitation2.cpp`, they need to take them as argument...)

### Back to physics

19. *Random initialization:*

Create (and use instead of initial conditions given for the sun) a function initializing a `Ball`, with position in the window, null speed, radius between 5 and 15, and mass being radius divided by 20 (beware not to use Euclidean division). You will use the `Imagine++` random functions of Section [A.4.4](#).

20. *Suns galore...*

Place 10 suns randomly (and take them into account in the computation of the motion for the asteroid...).

21. *Tune the time step of the computation:*

To avoid errors due to discretization of time, decrease the time step  $dt$ , such as 0.01 (or even 0.001 if the machine is powerful enough). Tune the display frequency accordingly (inversely proportional to  $dt$ ). Launch several times the program.

22. *Everything moving, everything bouncing:*

Have everything move, including suns. Use for elastic shocks the function `shock` (that makes the two bodies bounce).

### A.4.3 A third program: Duel

23. *Add a new program:*

Create a file `duel.cpp`, initially a copy of `gravitation2.cpp`. Modify then the file `CMakeLists.txt` to append two lines indicating that the executable `Duel` depends on `duel.cpp` and uses the `Graphics` library of `Imagine++`.



24. *Creating a library:*

The program will be different but will reuse exactly the same files for tools, vectors and balls. Thus we make them as a library, as indicated in the CMakeLists.txt:

```
add_library(Gravitation file1.cpp file2.cpp file3.cpp ...)
ImagineUseModules(Gravitation Graphics)
```

replacing `filei.cpp` with the common files. Then we can remove these files from the `add_executable` line of `Gravitation2`, and instead add the line

```
target_link_libraries(Gravitation2 Gravitation)
```

At this point, you can add the three lines for program `Duel`, which uses also the library `Gravitation`.

25. *Your turn!*

Transform the project `Duel`, with the help of the functions already defined, in a shooting game with two players. Each player has a fixed position, and various suns are placed randomly on screen. Each player in turn can launch a `Ball` with the chosen initial speed, the ball undergoing the gravitation of the different suns and disappearing after 250 display time steps. The winner is the first one that launches the ball to the other one... Practical advice: place the players symmetrically with respect to the center, at mid-height leaving a margin of one eighth of the width; use the function `getMouse` to know the position of the mouse at a click; deduce the chosen speed by subtracting from these coordinates the ones of the ball center to launch and multiplying by a factor 0.00025.

26. *Correct initialization:*

Modify the function putting suns so that they do not intersect initially, and that they are at least at distance 100 pixels from the players.

27. *Trace of the shoot:*

Instead of simply erasing the previous position of the ball, we also redraw it with half radius.

### A.4.4 Help

Given functions:

```
void initRandom();
```

(in `Imagine++`) is to be executed before the first call to `random` (and only once).

```
double intRandom(int a, int b);
```

(in `Imagine++`) returns an `int` drawn randomly between `a` and `b`.

```
double doubleRandom();
```

(in `Imagine++`) returns a `double` drawn randomly between 0 and 1.

```
void shockSimple(double x, double y, double& vx, double& vy, double m,
 double x2, double y2, double vx2, double vy2);
```



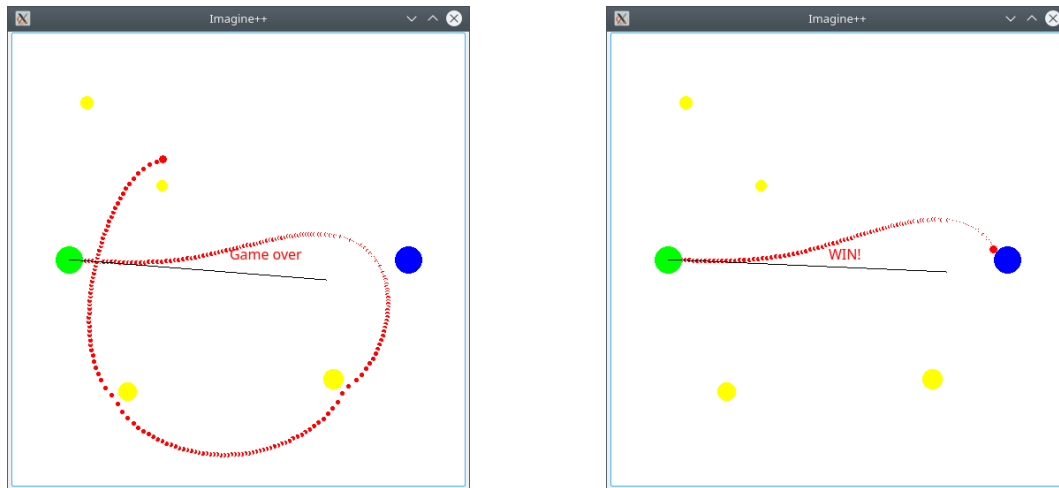


Figure A.6: Duel: on the left, the green player failed hitting the blue one (black line showing the shooting direction and impulse). On the right, a slight correction led to a win...

makes the first particle bounce, with coordinates  $(x, y)$ , speed  $(v_x, v_y)$  and mass  $m$ , on the second one, with coordinates  $(x_2, y_2)$  and speed  $(v_{x2}, v_{y2})$ , without moving the first one.

```
void shock(double x, double y, double& vx, double& vy, double m,
 double x2, double y2, double& vx2, double& vy2, double m2);
```

makes *both* particles bounce.

```
bool collision(double x1, double y1, double vx1, double vy1, double r1,
 double x2, double y2, double vx2, double vy2, double r2);
```

returns **true** if the body at  $(x_1, y_1)$ , with speed  $(v_{x1}, v_{y1})$  and with radius  $r_1$  is about to collide with the body at  $(x_2, y_2)$ , with speed  $(v_{x2}, v_{y2})$  and radius  $r_2$ , and **false** otherwise.

## A.4.5 Physics

Remark: this section is here only to explain the contents of the function already programmed. You can skip it if not interested.

### Acceleration

The sum of forces applying on a body  $A$  is equal to the product of its mass by the acceleration of its center gravity.

$$\sum_i \vec{F}_{i/A} = m_A \vec{a}_{G(A)}$$

### Universal gravitation

Given two bodies  $A$  and  $B$ ,  $A$  undergoes an attraction force from  $B$ :

$$\vec{F}_{B/A} = -Gm_A m_B \frac{1}{d_{A,B}^2} \vec{u}_{B \rightarrow A}.$$

### Elastic shocks

Let  $A$  and  $B$  two particles involed in a collision. Knowing all parameters before the shock, how to determine their values after? Actually, only the speed of the particles need to be updated, since at the shock moment, no position is changed.

During a shock said *elastic*, three quantitates are preserved:

1. momentum  $\vec{P} = m_A \vec{v}_A + m_B \vec{v}_B$
2. cinetic moment  $M = m_A \vec{r}_A \times \vec{v}_A + m_B \vec{r}_B \times \vec{v}_B$  (a real number in the case of plane motion).
3. cinetic energy  $E_c = \frac{1}{2} m_A v_A^2 + \frac{1}{2} m_B v_B^2$ .

Therefore, we get 4 equations for 4 unknowns.

### Shock resolution

We move into the corrdinate frame of the center of mass. We get at each time:

1.  $\vec{P} = 0$  (by definition of this frame), hence  $m_A \vec{v}_A = -m_B \vec{v}_B$ .
2.  $M = (\vec{r}_A - \vec{r}_B) \times m_A \vec{v}_A$ , hence, noting  $\Delta \vec{r} = \vec{r}_A - \vec{r}_B$ ,  $M = \Delta \vec{r} \times m_A \vec{v}_A$ .
3.  $2E_c = m_A(1 + \frac{m_A}{m_B})v_A^2$ .

The constancy of  $E_c$  indicates that in this frame, the norm of speed is preserved, and the constancy of the cinetic moment that the speeds vary parallely to  $\Delta \vec{r}$ . Apart from the initial speeds, there is a unique possibility left: multiply by  $-1$  the component of  $\vec{v}_i$  along  $\Delta \vec{r}$ . That leads to a simple algorithm for the shock.

### Deciding if a shock is going to happen

We won't be happy, along the step by step evolution of the coordinates of the disks, while deciding if a shock will happen between  $t$  and  $t + dt$ , with just estimating the distance between the two candidates to collision at time  $t$ , not even by taking in consideration this distance at  $t + dt$ , because, if the speed is too high, one disk may already have gone through the other and gone out at  $t + dt$ ... The best solution is to explicitly compute the minimum distance between the disks along the time, between  $t$  and  $t + dt$ .

Let  $N(u) = (\vec{r}_A(u) - \vec{r}_B(u))^2$  be the square of the distance. We get:

$$N(u) = (\vec{r}_A(t) - \vec{r}_B(t) + (u - t)(\vec{v}_A(t) - \vec{v}_B(t)))^2$$

This gives, with supplementary notations:

$$N(u) = \Delta \vec{r}(t)^2 + 2(u - t)\Delta \vec{r}(t) \cdot \Delta \vec{v}(t) + (u - t)^2 \Delta \vec{v}(t)^2$$

The norm, always positive, is minimal at point  $u$  so that  $\partial_u N(u) = 0$ , hence:

$$(t_m - t) = -\frac{\Delta \vec{r}(t) \cdot \Delta \vec{v}(t)}{\Delta \vec{v}(t)^2}$$

Thus:

1. if  $t_m < t$ , the minimum is reached at  $t$ ,
2. if  $t < t_m < t + dt$ , the minimum is reached at  $t_m$ ;
3. otherwise,  $t + dt < t_m$ , the minimum is reached at  $t + dt$ .

That gives easily and explicitly the smallest distance between the bodies between times instants  $t$  and  $t + dt$ .

## A.5 Images



Figure A.7: Two images and various processing on the second one (negative, blur, relief, deformation, contrast and edges).

In this practical, we will play with bidimensional arrays (but stored in 1D arrays), first static then dynamic. To change from matrices, however fascinating they can be, we will work with images (figure A.7).

### A.5.1 Allocation

1. *Get the project:*  
Download the file `Tp6_Initial.zip`, decompress it and launch your favorite development environment, Qt Creator.
2. *Fill the memory:*  
Nothing to do with what we will do after, but it is nice to have seen it at least once... Do, in an infinite loop, allocations of 1000000 integers without deallocation, each followed by a pause of 0.5 seconds, and look at the process size growing. (Use `Ctrl+Shift+Echap` to get the task manager under Windows, use command `top` in a terminal in Linux or MacOS).<sup>11</sup>

### A.5.2 Static arrays

3. *Gray levels:*  
A black and white image is represented by an array of pixels of constant dimensions  $W=300$  and  $H=200$ . Each pixel  $(i, j)$  is a `byte` (integers between 0 and

<sup>11</sup>Your program should crash at a certain point, when no more memory is available. If not, try in mode Release to have a true handling of the heap (The mode Debug may behave differently...)

255) with value 0 for black and 255 for white. The origin of coordinates is the top left,  $i$  is horizontal and  $j$  is vertical. In a mono-dimensional array of byte  $t$  of size  $W \times H$  store the pixel  $(i, j)$  in  $t[i+W*j]$ :

- Create a black image and display it with `putGreyImage(0, 0, t, W, H)`.
- Same for a white image.
- Same for a gradation from black to white (be careful with risks of Euclidean division with integers, conversion to `double` may be necessary).
- Same with  $t(i, j) = 128 + 127 \sin(4\pi i/W) \sin(4\pi j/H)$  (see figure A.7). Use

```
#include <cmath>
```

to get mathematical functions and constants: `M_PI` vaut  $\pi$ .<sup>12</sup>

#### 4. Colors:

Display with `putColorImage(0, 0, r, g, b, W, H)`, an image in color stored in three arrays  $r$ ,  $g$  and  $b$  (red, green, blue). Use function `click()` to wait a user mouse click after each new display.

### A.5.3 Dynamic arrays

#### 5. Dimensions from the keyboard:

Modify the program so that  $W$  and  $H$  are not constant anymore but values input from the user with the keyboard. Do not forget deallocation.

### A.5.4 Load a file

#### 6. Color image:

The call `loadColorImage(srcPath("ppd.jpg"), r, g, b, W, H);` loads the file "ppd.jpg" that is located in the source folder (`srcPath`), allocates by itself the arrays  $r$ ,  $g$ ,  $b$ , fills them with image pixel values, and assigns also  $W$  and  $H$ .<sup>13</sup> Warning: do not forget to deallocate arrays  $r$ ,  $g$ ,  $b$  with `delete[]` after use.

- Load this image and display it. Do not forget deallocation, again.

#### 7. Black and white image:

The function `loadGreyImage(srcPath("ppd.jpg"), t, W, H)` does the same but converts the image to grayscale. Display this image...

<sup>12</sup>To be exact, `M_PI` is not standard, some compilers may not define it. Don't worry, your compiler has it.

<sup>13</sup>The size of the image is stored in a header of the file, and read by the function. This is fortunate, otherwise the program would have to know in advance the size of the images it uses.

### A.5.5 Functions

8. *Split the work:*

We will only work with the graylevel image from now on. Write functions to allocate, destroy, display and load images:

```
byte* allocImage(int W, int H);
void deallocImage(byte *I);
void displayImage(byte* I, int W, int H);
byte* loadImage(const char* name, int &W, int &H);
```

9. *Files:*

Create files `image.cpp` and `image.h` for the functions above...

### A.5.6 Structure

10. *Principle:*

Modify the preceding program to use a structure:

```
struct Image {
 byte* t;
 int w,h;
};
```

`AllocImage()` and `LoadImage()` can return some `Image`.

11. *Independence:*

To avoid having to remember how pixels are stored, add:

```
byte get(Image I, int i, int j);
void set(Image I, int i, int j, byte g);
```

12. *Processing:*

Add in `main.cpp` different functions to process images

```
Image negative(Image I);
Image blur(Image I);
Image relief(Image I);
Image edges(Image I, double threshold);
Image deform(Image I);
```

and use them:

- (a) `negative`: invert black and white by an affine transform.
- (b) `blur`: each pixel becomes the average of itself and its 8 neighbors. Beware of pixels at image border that have fewer than 8 neighbors (leave them unchanged and use instruction `continue!`). In general, when you do image processing, you must always be careful not to get outside the image.
- (c) `relief`: the derivative along a diagonal gives the impression of cast shadows from an oblique lighting.

- Approximate this derivative by finite difference: it is proportional to  $I(i + 1, j + 1) - I(i - 1, j - 1)$ .
  - Rescale the gray levels by an affine function to get back in range 0 to 255.
- (d) edges: compute by finite differences the horizontal  $d_x = (I(i + 1, j) - I(i - 1, j))/2$  and the vertical  $d_y$  derivatives, then the norm of gradient  $|\nabla I| = \sqrt{d_x^2 + d_y^2}$  and display in white the points where this is above some threshold.
- (e) deform: Build a new image with the principle  $J(i, j) = I(f(i, j))$  with  $f$  smartly chosen. You can use a sine to go from 0 to  $W-1$  and from 0 to  $H-1$  in a nonlinear fashion.<sup>14</sup>

### A.5.7 Final clean-up

13. Clean up everything: it is likely you have put in comments the code for first questions in order to save time during further development. Put them back un-commented, and check that everything still compiles and runs as expected.
14. If you have time left, you can try:
- Make a reduced image.
  - Instead of negative, you can change the contrast, for example by multiplying (by a small factor  $\lambda$ ) the difference between a pixel and the average of its neighbors (this is the negative of Laplacian):<sup>15</sup>

$$J(i, j) = I(i, j) + \lambda \left( I(i, j) - \frac{I(i - 1, j) + I(i + 1, j) + I(i, j - 1) + I(i, j + 1)}{4} \right)$$

<sup>14</sup>Whatever your choice, if function  $f$  goes outside the image  $I$ , just replace with a white pixel.

<sup>15</sup>Saturate values that go out of range 0 to 255, otherwise it will perform a modulo and you will get for example white pixels appearing in a dark region.

## A.6 First objects and fractals

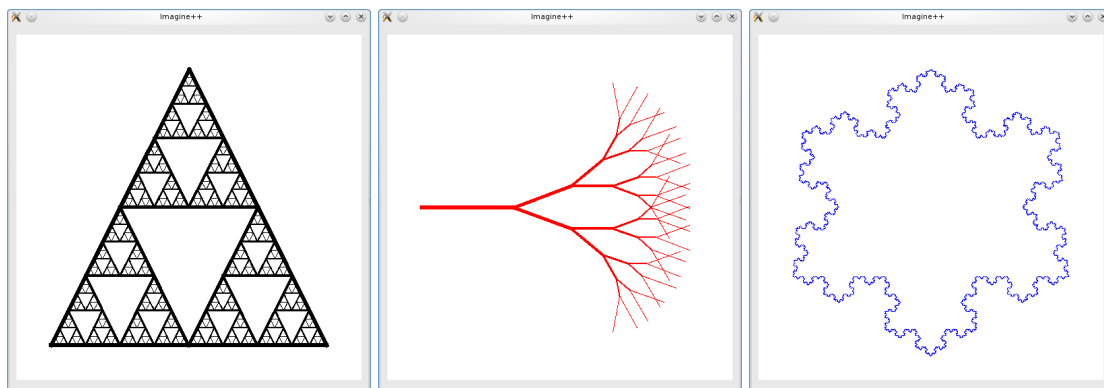


Figure A.8: Fractals...

In this practical, we will try some modest object oriented programming. We will transform a structure for a vector in a class and use it to draw fractals (figure A.8).

### A.6.1 Sierpinski triangle

1. *Get the project:*

Download the file `Tp7_Initial.zip`, decompress it and launch your IDE. Study the structure `Vector` defined in files `Vector.cpp` and `Vector.h`.

2. *Interface with Imagine++:*

The structure `Vector` does not provide any graphic display. Add in `main.cpp` functions `drawLine` and `drawTriangle` taking some `Vector` as arguments (here, a vector is used to note a plane point). Just call the regular function from `Imagine++`

```
void drawLine(int x1, int y1, int x2, int y2,
 Color c, int pen_w)
```

The last argument controls the thickness of the pencil.

3. *Sierpinski Triangle:*

This is the figure chosen by ENPC for its logo. Figure A.9 illustrates its construction.

Write a recursive function to draw the Sierpinski triangle. This functions takes as arguments the three points of the current triangle and the thickness of the pencil. The three subtriangles are drawn with a pencil less thick. **Do not forget the break condition for the recursion!**

Use this function in `main` by giving it an initial equilateral triangle of thickness 6 pixels.



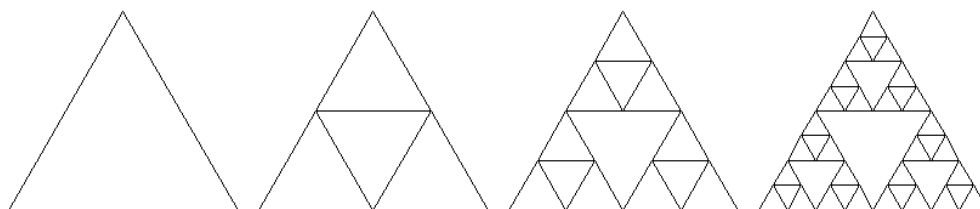


Figure A.9: Construction of Sierpinski triangle.

### A.6.2 A class rather than a structure

#### 4. *Class vector:*

Transform the structure `Vector` into a class. Incorporate all functions and operators. Put in public only what is necessary, do the required modifications in `main.cpp`.

#### 5. *Accessors for members:*

Add accessors for reading and writing members, and use them systematically in the main program. The idea is to hide from the user of the class `Vector` the details of its implementation.

#### 6. *Recursive drawing of a tree:*

We are now going to draw a tree. For that, we start from a trunk and replace the second half of each branch by two branches of same length with an angle of 20 degrees with the originating branch. Figure A.10 illustrates the result for different depths of recursion.

Write a recursive function to draw such a curve. You will use the method

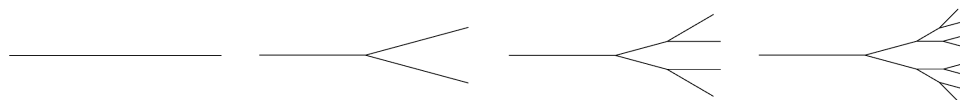


Figure A.10: Tree construction.

`rotate` of class `Vector`.

It is natural to think of erasing the half of the preceding segment to draw the new branches, but it is not a great idea: the segments are discretized (in pixels), and a discretized subsegment may not go exactly on the same pixels as the master segment, hence some residual stains after erasing. The same remark will apply to the Koch curve below.

### A.6.3 Change the implementation

#### 7. *Second implementation:*

Modify the implementation of class `Vector` by replacing the members `double x,y`; by an array `double coord[2]`; What modifications should be applied in `main.cpp`?

8. *Vectors of higher dimension:*

The advantage of the latest implementation is that it can be generalized to vectors of higher dimension. Put a global constant `DIM` of value 2 at the top of file `vector.h` and make the class `Vector` independent of the dimension.

NB: The method `rotate` and the accessors we defined cannot be generalized directly to higher dimension. Leave them unchanged but reserve their use to dimension 2 by putting an `assert...`

### A.6.4 The snowflake

9. *Koch curve:*

This fractal curve can be built by starting from a segment and replacing the second third of each segment by two segments so as to get an equilateral triangle.

Write a recursive function to draw this curve.



Figure A.11: Construction of the Koch curve.

10. *Snowflake:*

It is obtained by building three Koch curves from each of the sides of an equilateral triangle.

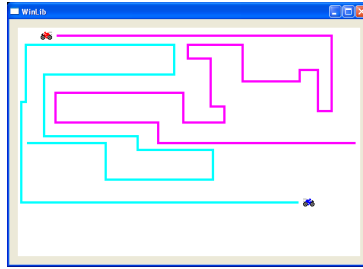


Figure A.12: Game of Tron.

## A.7 Tron

In this practical, we are going to program the game of TRON. It is a game with 2 players in which each controls a mobile that moves with constant speed and leaves behind a trace. The first player that crashes on its trace or the adversary's trace (or exits the window) loses the game. This practical is rather ambitious and is a mini-project. It will take us several sessions.

### A.7.1 Snake

We will proceed in two steps. First program a game of Snake with a single player. In this game, the single player controls a snake that grows little by little (of one element each  $x$  round, with the convention that the total length is bounded by a constant  $n_{max}$  elements).

The starting project has two files, `utils.h` and `utils.cpp`, that contain a structure point (that we may enrich with useful methods later) and a function meant to get the keys pressed by the players.

We will build an object `Snake` with adequate methods, and a function `game_1p` exploiting the capacities of the `Snake` to reproduce the desired behavior. First, we won't care about collisions (with the border or with itself), and only do that in a second time. Your work is split into 6 steps.

1. *(on paper)* Define an interface for class `Snake` (list all necessary functionalities).
2. *(on paper)* Think about implementation of the class `Snake`: how to store the data? How to program the different methods? (read first some remarks from next paragraph).
3. In a file `snake.h`, write the declaration of the class `Snake`: its members, its methods, what is public, what is not.
4. *Submit the result of your thoughts to the instructor to validate your choices.*
5. Implement the class `Snake` (that is, program the methods that you declared).
6. Program the function `game_1p` using a `Snake`.

Remark: the file `utils.h` defines:

1. 4 integers `left`, `down`, `up`, `right` such that:
  - (a) the function  $x \rightarrow (x + 1)\%4$  transforms left to down, down to right, right to up and up to left; it corresponds to a quarter turn in trigonometric sense.
  - (b) the function  $x \rightarrow (x + 3)\%4$  makes the same but in clockwise direction.<sup>16</sup>
2. an array of 4 points `dir` such that, after definition of a function computing the sum of two points, the function  $p \rightarrow p + \text{dir}[d]$  returns:
  - (a) for  $d = \text{left}$  the point corresponding to a shift of  $p$  of 1 unit left.
  - (b) for  $d = \text{up}$  the point corresponding to a shift of  $p$  of 1 unit up.
  - (c) for  $d = \text{right}$  the point corresponding to a shift of  $p$  of 1 unit right.
  - (d) for  $d = \text{down}$  the point corresponding to a shift of  $p$  of 1 unit down.

## A.7.2 Tron

From the game of Snake, it will be easy to implement the game of Tron.

1. Two players.

Inspired by function `game_1p`, create a function `game_2p` with two players. We will use for this player the keys S, X, D and F.<sup>17</sup> The function `keyboard()` will return the integers `int('S')`, `int('X')`, `int('D')` and `int('F')`. Remark: we treat only one key per round, hence a single call to function `keyboard()` per round, otherwise the snakes are in concurrency for the keyboard.

2. Ultimate tuning.

- (a) Handle the collision of the snakes.
- (b) The principle of Tron is that the trace of mobiles remains. To implement that, we just need to extend the snake at each round.

## A.7.3 Graphics

Small bonus to make our game more attractive: we will see how to handle graphics instead of the uniform rectangles we had until now. The objective is to replace the heading square by an image that we will move at each step.

We are going to use the `NativeBitmap` of `Imagine++`, that are images that are faster to draw on screen than regular images. To put an image in a `NativeBitmap` we proceed like follows:

```
// Integers passed by reference when loading the image
// so as to store the width and height of the image
int w,h;
// Image loading
```

<sup>16</sup>In mathematics,  $(x + 3)\%4 = (x - 1)\%4$ , but for C++  $-1\%4 = -1$ , which is wrong for our purpose.

<sup>17</sup>The keys A, Z, Q and W are not in the right configuration for a Qwerty keyboard, thus avoid them.

```

byte* rgb;
loadColorImage(srcPath("image_name.bmp"), rgb, w, h);
// Declaration of a NativeBitmap
NativeBitmap my_native_bitmap(w, h);
// Put the image in the NativeBitmap
my_native_bitmap.setColorImage(0, 0, rgb, w, h);
delete [] rgb; // We don't need the image anymore

```

The display of a `NativeBitmap` on screen can be done using the function of `Imagine++`:

```
void putNativeBitmap(int x, int y, NativeBitmap nb)
```

1. Replace in the snake the display of the head by a bitmap. You can use the images `moto_blue.bmp` and `moto_red.bmp` in the archive.
2. Use the image `explosion.bmp` to show the crash of a player.

### A.7.4 Make the code great

To have a clean code of which you may be proud, heed the following guidelines:

- The class `Snake` has one/several constructor(s).
- The class `Snake` has a destructor only if necessary. Hint: did some dynamic allocation occur?
- No useless copy of a `Snake` when passing as argument of a function. Hint: are objects of type `Snake` passed by reference? See the course about copy constructor to understand why it matters.
- Only methods used by the exterior are public. All others must be private.
- The methods that do not change the `Snake` are `const`.
- The indentation is correct. Do we really have to repeat this obvious guideline?
- The bitmaps are read with `srcPath` and the program stops if one image file is not found.



## Appendix B

# Imagine++

Imagine++ is a set of libraries (sometimes called a *framework*) allowing to do simply graphics and linear algebra. They rely for that on projects Qt (for 2D and 3D graphics) and Eigen (linear algebra). These provide a wealth of possibilities much greater than Imagine++, but are much more complex, especially for the beginner.

Imagine++ is free and open-source software, so that you can use and distribute it as you wish (for free!), but if you distribute it modified, you must offer the sources of your modifications under identical terms. A complete documentation is available on its Web page, including installation and usage.

To use a module (that is a library), such as `Images`, a source file must include it

```
#include <Imagine/Images.h>
```

for a correct compilation, and your file `CMakeLists.txt` must use

```
ImagineUseModules(MyProgram Images)
```

for the link to succeed.

Everything is in a namespace `Imagine`, so that if you want to avoid prefixing all by `Imagine::` you must use in your code

```
using namespace Imagine;
```

### B.1 Common

The module `Common` defines among others the class `Color`, a mix of red, green and blue (though used primarily in `Graphics`). The quantity of each is given by an integer between 0 and 255 (type `unsigned char`):

```
Color black(0,0,0);
Color white(255,255,255);
Color red(255,0,0);
```

A few constants of this type are already defined: `BLACK`, `WHITE`, `RED`, `GREEN`, `BLUE`, `CYAN`, `MAGENTA`, `YELLOW`, `ORANGE`, `PURPLE`.

The type `byte` (synonym of `unsigned char`) is defined to store an integer value between 0 and 255. Components of `Color` are of this type and can be accessed in several manners:

```
cout << "Blue_component_of_color_white:" << white[2]
 << '=' << white.b();
```

Quite convenient, `srcPath` prefixes its character string argument with the complete path to the folder containing the source file. The same for an argument of type `string` is `stringSrcPath`:

```
const char* file = srcPath("my_file.txt");
string s = "my_file.txt";
s = stringSrcPath(s);
```

In other words, the file will be found whatever the location of the executable program.

The template class `FArray` is used for arrays of small size, which must be known at compile time (static array). For arrays whose size is unknown at compilation time (dynamic allocation), use `Array`. For matrices and vectors, use `FMatrix` and `FVector`, using static allocation as indicated by the prefix `F` (*fixed*). Equivalent dynamic ones are present in `LinAlg`.

Also convenient, the call to function `intRandom(5,10)` returns a random integer in range  $[5, 10]$ . To obtain at each run of the program a new sequence of random draws, use `initRandom()`, a function normally called only once, typically at the beginning of the main function. `doubleRandom()` and `gaussianRandom()` return a uniform in  $[0, 1]$  or normal Gaussian-distributed number.

## B.2 Graphics

The module `Graphics` provides 2D and 3D graphics. The 2D coordinates are in pixel unit, the  $x$ -axis goes to the right and  $y$  to the bottom (beware, it is not the usual mathematical orientation!). Point  $(0,0)$  is at upper-left corner of the window (for drawing) or of the screen (for `openWindow`).

```
openWindow(500,500); // Size of window
drawRect(10,10, 480,480,RED); // Top-left (10,10), size 480x480
drawLine(10,10, 490,490,BLUE); // Diagonal
Window w = openWindow(100,100); // Other window
setActiveWindow(w); // Select for next drawings
drawString(10,10, "Some_text", MAGENTA); // Putting text
endGraphics(); // Wait for a mouse click then close windows
```

The last function is normally called just before getting out of `main` function so that the user can see the graphics as long as desired before the program stops.

If we have many drawings at once and we want to display only the final result (to avoid flicker), we frame the drawing code by:

```
noRefreshBegin();
...
noRefreshEnd();
```

To have an animation, it is useful to have a little pause between images to tune the rhythm:

```
milliSleep(50); // Time in milliseconds
```



Do not put such an instruction between `noRefreshBegin` and `noRefreshEnd`, because nothing would display during that period.

We can load an image in memory (`loadGreyImage`, `loadColorImage`) or save (`saveGreyImage`, `saveColorImage`) it in a file. The former functions allocate memory that should be freed (with `delete []`) after usage.

```
byte* g;
int width, height;
if (! loadGreyImage(srcPath("image.jpg"), g, width, height)) {
 cerr << "Error_opening_file_"
 << srcPath("image.jpg") << endl;
 exit(1);
}
// Draw with top-left corner at (0,0):
putGreyImage(0, 0, g, width, height);
delete [] g; // Do not forget!
```

Note the `srcPath`, defined in `Common`, indicating to look for the file name in the folder containing the source file.

To avoid managing by hand the memory used by images, there is a class dedicated to this:

## B.3 Images

The module `Images` manages loading, manipulating and saving images.

```
Image<byte> im; // Image in gray levels
if (! load(im, srcPath("image_file.png"))) {
 cerr << "Error_opening_file_"
 << srcPath("image_file.png") << endl;
 exit(1);
}
display(im); // Draw in active window
im(0,0)=128; // Put top-left pixel in midlevel gray
save(im, "image_file2.png"); // Save in file
```

Beware: copy and assignment (`operator=`) are cheap and only perform a link between images (shallow copy). The consequence is that modifying one impacts the other:

```
Image<Color> im1(100,100);
Image<Color> im2 = im1; // Copy constructor
im1(10,10) = CYAN;
assert(im2(10,10) == CYAN); // im2 is impacted also!
```

To do a real copy (deep copy) rather than a link, we use:

```
im2 = im1.clone();
im1(10,10) = CYAN; // No effect on im2
```

The consequence is that if an image is passed as argument of a function, since the copy constructor is called, everything happens as if the image were passed by reference:

```
void f(Image<Color> im) { // Value passing
 im(10,10) = CYAN;
}
f(im1); // Still modifies pixel (10,10)
```

A classical error is trying to read or write at coordinates beyond the array boundaries, typically an error in a loop index.

## B.4 LinAlg

The module `LinAlg` proposes linear algebra with matrices and vectors.

```
Matrix<float> I(2,2); // Size 2x2
I.fill(0.0f); // Null matrix
I(0,0)=I(1,1)=1.0f; // Identity matrix
cout << "det(I)=" << det(I) << endl; // Determinant
```

Operators for sum (matrix+matrix and vector+vector), subtraction (matrix-matrix and vector-vector) and multiplication (matrix\*matrix and matrix\*vector) are of course defined.

As for images, beware not to go out the bounds while accessing coefficients of matrix and vector!

A very useful function is `linSolve` to solve a system of linear equations (by least squares in case there are more equations than unknowns). Several standard matrix factorizations are also implemented.

## B.5 Installation

Let's look at the file `CMakeLists.txt` of a program using `Imagine++`.

```
cmake_minimum_required(VERSION 2.6)
project(Ball)
find_package(Imagine REQUIRED)
```

```
add_executable(Ball Ball.cpp)
ImagineUseModules(Ball Graphics)
```

In order for the `find_package` to work, it may be necessary to define the system variable `Imagine_DIR`.<sup>1</sup> This command will load the file<sup>2</sup>

```
/usr/share/Imagine++/CMake/ImagineConfig.cmake
```

that contains `CMake` commands, including the definition of `ImagineUseModules`. The latter performs two things:

- Indicate to the compiler<sup>3</sup> where to look for header files with `#include`, such as in the folder

<sup>1</sup>It is the case when `Imagine++` is not installed in the standard installation folders, which depend on the platform.

<sup>2</sup>This example is based on Linux, where `/usr/share` is a standard installation folder.

<sup>3</sup>More precisely, the preprocessor, which is launched just before actual compilation.

```
/usr/share/Imagine++/include
```

Therefore, the instruction

```
#include "Imagine/Graphics.h"
```

will include the file

```
/usr/share/Imagine++/include/Imagine/Graphics.h
```

(this calls the standard CMake function `include_directories`).

- Indicate to the linker to use the library `libImageGraphics.a` after compilation. This is located in folder

```
/usr/share/Imagine++/lib
```

(this calls the standard CMake function `target_link_libraries`).



# Appendix C

## Compiler, CMake, Linker, Qt... Help!

*All right, all this seems quite complex, and indeed it is not that simple. However, if you want to understand the tools you use, we will see that we can get around them, despite the fright of being overwhelmed by so many tools. Reading this appendix is not required to use them, but it should be helpful to help understanding how they are organized.*

—

### C.1 Compilation

#### C.1.1 Compiler and linker

Let's begin with the basic principle: to go from *source code*, that is files with extension `.h` and `.cpp`,<sup>1</sup> to an *executable program*, we need to launch the *build*, often called abusively compilation, whereas the true compilation is only a part of the process.<sup>2</sup> It is composed of two phases, compilation (the job of the compiler) followed by the link (the job of the *linker*). The linker's role is to gather the products of the compilation phase, the *object files* (with extension `.o` or `.obj`), with possible *libraries*, that are in essence only archive-like files (like a zip) of object files. This is summed up by the scheme of Figure C.1.

Several things are worth noticing:

- The compiler is launched *independently* on each source `.cpp`, to the point that the compilations could be launched in parallel. An important consequence is that the modification of a `.cpp` does not require recompilation of the others, hence a time saving in the build.
- The *header files* (in `.h`) are *not* passed directly to the compiler, but they are indirectly by the `.cpp` that `#include` them. Besides, since several `.cpp` can `#include "file1.h"`, this file will go several times under the compiler. It is thus essential that the compiler does not generate *symbols* from `file1.h`, such as global variables (which should be avoided as much as possible anyway) or function

---

<sup>1</sup>Sometimes extension `.h` is replaced by `.hpp` to avoid confusion with language C header files. We can also meet the conventions `.hxx` and `.cxx`, since the `+` in file name could be badly managed by certain file systems, and after all `x` is just a rotated `+`.

<sup>2</sup>This is a synecdoque. Doing engineering may not hinder the taste for literary figures!

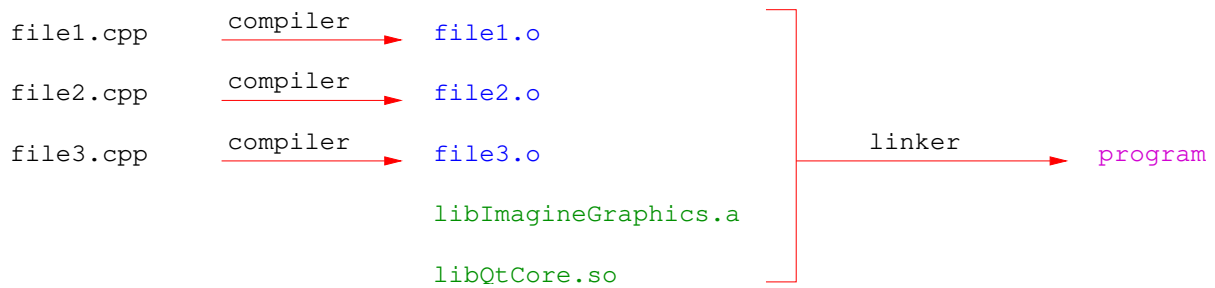


Figure C.1: Phases of compilation and link. File extensions here are for Unix systems (Linux and Mac), under Windows object files end with `.obj`, libraries with `.lib` and the program with `.exe`.

definitions, but only declarations (of classes, functions, constants).<sup>3</sup> Otherwise, at link time, there will be an error: `Multiply defined symbol...`

- The role of the linker is to check that each used function or class method<sup>4</sup> is defined one and only one time, that is, in a unique `.o` or library. Moreover, it checks that the entry function of the program, the `main`, is present (exactly once).
- Libraries can be present in two forms: static (extension `.a`) and dynamic (extension `.so`, as `shared object`). At link time, used symbols of static libraries are included in the resulting program, whereas those of dynamic libraries are not, the linker just checking their existence. The advantage of the latter is that the program is lighter in memory, that several running programs using the same library require only a single load in memory of the library, and that the correction of security default or a bug in a dynamic library does not require a recompilation of all programs using it but only to redistribute it to users (provided these bugs are corrected without changing arguments of functions, etc). The counterpart is that the executable program cannot be launched directly since it is incomplete, it has to find the dynamic libraries it depends on.

Actually, under Windows, the dynamic libraries are separated in two: the `.lib` is used at link time (can be considered as a kind of header for the library) whereas the part `.dll` is used when the program is run. The program cannot be run if it does not find the needed `.dll`.<sup>5</sup> One can list the dynamic libraries a program depends on with the tools `ldd` (Linux), `otool` (Mac) and `depend.exe` (Windows).

<sup>3</sup>Exceptions are functions `inline`, class methods defined directly in the class, that are also `inline`, and functions or methods `template`, that are very particular since they cannot be compiled as long as the template parameters are not known, we talk about *instantiation* of templates.

<sup>4</sup>A function or class method just *declared* but not *called* does not require a *definition* but is simply considered as superfluous by the linker.

<sup>5</sup>It searches in the folder containing the program and in the list of folders indicated by the environment variable `PATH`. Under Linux and Mac, the equivalent variable to search `.so` is named `LD_LIBRARY_PATH`.

## C.2 With command line

Here is an example<sup>6</sup> of command-line invocation of the compiler on a file `test.cpp`, resulting in object file `test.o`:

```
/usr/bin/c++ -DQT_CORE_LIB -DQT_GUI_LIB -DQT_NO_DEBUG
-DQT_OPENGL_LIB -DQT_WIDGETS_LIB -O3 -DNDEBUG
-I/home/pascal/Ponts/svn/ImagineQt/Imagine/Common/src
-I/home/pascal/Ponts/svn/ImagineQt/Imagine/Graphics/src
-I/usr/include/qt5 -I/usr/include/qt5/QtCore
-I/usr/lib/x86_64-linux-gnu/qt5/mkspecs/linux-g++-64
-I/usr/include/qt5/QtWidgets -I/usr/include/qt5/QtGui
-I/usr/include/qt5/QtOpenGL
-DSRCDIR="/home/pascal/Ponts/svn/ImagineQt/Imagine/Graphics/test"
-o CMakeFiles/ImagineGraphicsTest.dir/test.o
-c /home/pascal/Ponts/svn/ImagineQt/Imagine/Graphics/test/test.cpp
```

The compiler program is called `c++`, the options `-D` do as if `test.cpp` had a corresponding `#define`. The option `-O3` indicates optimization of level 3 (the highest one). The options `-I` indicate the folders where the `#include` must look for header files if they are not in current folder. Now, the link of this single object file with the libraries it needs:

```
/usr/bin/c++ CMakeFiles/ImagineGraphicsTest.dir/test.o
-o ImagineGraphicsTest ../src/libImagineGraphics.a
/usr/lib/x86_64-linux-gnu/libQt5Core.so.5.2.1
/usr/lib/x86_64-linux-gnu/libQt5Widgets.so.5.2.1
/usr/lib/x86_64-linux-gnu/libQt5Gui.so.5.2.1
/usr/lib/x86_64-linux-gnu/libQt5OpenGL.so.5.2.1
-lGLU -lGL -lSM -lICE -lX11 -lXext
```

The resulting program is called `ImagineGraphicsTest`, using the static library `libImagineGraphics.a` and dynamic Qt libraries, so as the system libraries, namely `libGLU.so`, `libGL.so`, etc. The latter are installed at locations known from the linker, so it does not need indication of where to find them, though it could have been precised with options `-L`, in the same manner as `-I` for the compiler. Actually, the same program `c++` understands from its arguments that it has to call the linker and not the compiler.

We see that to use a library in our program, we need to indicate (1) where the compiler will find its header files `.h`, (2) where the linker will find the libraries.

## C.3 Make and CMake

We see that the command lines above are not too complex but fastidious, and repeating these steps for each build would get tiresome. At this point the build automaton `make` appears. It reads a file named `Makefile` that describes the dependencies between files.<sup>7</sup> A `Makefile` for the program above would indicate that the executable program `ImagineGraphicsTest` depends on `test.o`, which itself depends on `test.cpp`,

<sup>6</sup>Under Linux. This can be entered into a terminal window.

<sup>7</sup>Among others, that a `.cpp` file depend on the header files `.h` it includes, so that if one these headers was modified, the file must be recompiled.

but it does not stop here, since `test.cpp` performs some `#include` of header files `.h`, that themselves could depend on other `.h` through `#include`, etc.

The commands of generation of `test.o` and of `ImagineGraphicsTest` are those of compilation and link of the example above. The program `make` relies on the modification time of files to know what is up to date and needs no specific action, and what is not and requires application of the build rules: if file *A*, on which file *B* depends, was modified more recently than *B*, the rule of build of *B* is applied. In this manner the unique command `make` can perform both compilation and link. Running `make` a second time just after has no effect, since everything is up to date based on their modification time. It is the least effort principle and is reasonable.

To find all dependencies of a file `.cpp`, we need to look at its `#include` and follow them transitively. The compiler (actually, the preliminary phase of the compiler, the preprocessor, to which all instructions with `#` are addressed) is actually able to do it and writing dependencies by hand in a `Makefile` is long and probably difficult to maintain correct along development of the source code. To complicate things, the `make` command of Linux and Mac is *GNU make* whereas the one of Windows that comes with Visual Studio is *nmake*. They are not completely compatible, and since anyway the compilers are different and do not use the same options, we would need two versions of `Makefile` for a portable program. To avoid all this complexity, a more modern tool is *CMake*, a kind of meta-make. Its advantages are:

- A simpler format, uniform for all OS, of meta-`Makefile` called `CMakeLists.txt`.
- Automatic computation of dependencies.
- Built-in possibility (and it is strongly advised) to put the generated files in a different folder (*build directory*) than the sources (*source directory*), so that a cleaning procedure is very simple: remove the build directory.

The `CMakeLists.txt` for the preceding example is elementary:

```
cmake_minimum_required(VERSION 3.4)
find_package(Imagine REQUIRED COMPONENTS Graphics)
add_executable(ImagineGraphicsTest test.cpp)
ImagineUseModules(ImagineGraphicsTest Imagine::Graphics)
```

The first line is boilerplate. The second line indicates we need `Imagine++`, looking for a file named `ImagineConfig.cmake`. `Imagine++` is composed of several components, the `Graphics` one is the most frequently used. The third line indicates that the built program<sup>8</sup> is called `ImagineGraphicsTest(.exe under Windows)` and that it depends only on `test.cpp`. If there were several `.cpp`, their name would be included separated by blanks. We can also add `.h`, even though they are not compiled directly: they will be just visible in the part *headers* when we ask `CMake` to create a project (see *IDE* below). The last line indicates that our program needs the library `Graphics` of `Imagine++`, which indicates the path of headers and of the library. The fact that this library depends on `Qt` is automatically handled. A typical example of usage of `CMake` in command line under Linux or Mac:

```
mkdir Build ← Create build directory
cd Build ← Go in the directory
```

<sup>8</sup>to create a library (dynamic by default) instead, it is enough to replace by `add_library`.



`cmake ..` ← Indicates the source directory as the parent (`..`)

`make` ← Now the classic make command is run

Launching `cmake` creates a file `CMakeCache.txt` in the build directory that contains a list of variables for CMake. The most important is `CMAKE_BUILD_TYPE` that indicates the degree of optimization to apply. The maximum one is `Release`, whereas mode `Debug` is for a program we want to follow line by line. We can modify one of the variables<sup>9</sup> and run again `cmake`. This generates the file `Makefile` for us, which is then the configuration of `make`.

It is convenient that a further modification to `CMakeLists.txt`, for example to add source files at line `add_executable`, does not require to launch again explicitly `cmake`, since the generated `Makefile` keeps the information that `CMakeCache.txt` is a dependency of `CMakeLists.txt`. Hence the fact of launching `make` directly is enough to launch in turn `cmake` before continuing with normal build.

Finally, the power of CMake is that it knows all classic IDE (Visual Studio, Eclipse, Code::Blocks, etc) and is able to generate projects for these (cf Figure C.2). Absent from this list are `Kdevelop` (Linux) and `Qt Creator` (all platforms). Indeed, they generate themselves projects of type CMake, and we can launch `cmake` from their interface.

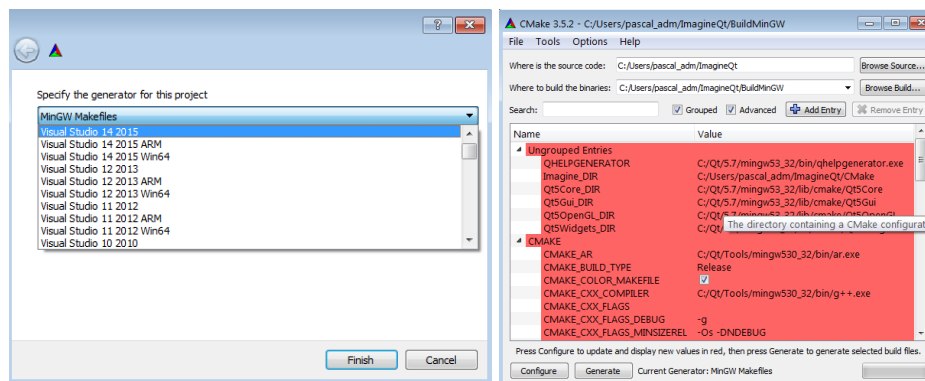


Figure C.2: Creation of a project with choice of IDE and interaction with variables of CMake stored in `CMakeCache.txt` thanks to `cmake-gui`. Note that CMake handles the different versions of Visual Studio, in 32 and 64 bits. The compiler MinGW being installed (it comes with the Qt installer under Windows), it can also create Makefiles as under Linux and Mac, except that GNU make that interprets them is called `mingw32-make` and not `make`.

The interest of all this is that the user can choose the IDE at will (or none if preferred) and that a single configuration file `CMakeLists.txt` ensures portability of build on all platforms. Then, it is the programmer's responsibility to respect the C++ norm for a maximum of portability.<sup>10</sup>

## C.4 Usage of an IDE

The comfort provided by an IDE is notable: integrated editor, easy code navigation, integrated build, debugger, version control (especially `git`), memory analyzer (`vagrind`

<sup>9</sup>directly in a text editor, or with tools `ccmake` or `cmake-gui` that offer a graphic interface

<sup>10</sup>Some compilers have *extensions* by default, features that are not portable because not part of the standard but deemed convenient.

under Linux and Mac)... One can also modify the `CMakeLists.txt` from the integrated editor and not launch `cmake` after: CMake is smart enough to create projects that know when it is necessary to run the `cmake` command. Once the project created, we can do everything from the IDE.

However, in order for all these tools to work in harmony, things must be correctly configured. For instance, Qt Creator must know where to find the program `cmake`, so as the compiler, the debugger, etc. At last, we have to be careful to keep CMake as build system, and refuse, when a new source file is created, the proposition of the IDE to integrate it in the project. This must be done at the level of CMake, by modifying `CMakeLists.txt` accordingly.

For Imagine++, the command `find_package(Imagine REQUIRED)` must be able to find the file `ImagineConfig.cmake`. If it is not in a standard location, it needs help in the form of the installation folder in a variable `Imagine_DIR`.

When the IDE seems to be malfunctioning, it is wise to close the project and restart it from scratch: erase generated files of the IDE, and reopen it. That can happen when major changes are done in the project and the IDE is unable to keep track.

At last, note that Qt *is not* an IDE but a set of multi-platform C++ libraries (sometimes called a *framework*), used by Imagine++ (which is also a framework). It happens that the IDE Qt Creator<sup>11</sup> uses also the Qt libraries, and if it has been installed together with the Qt libraries, it knows where to find them, which is helpful for an Imagine++ project.

## C.5 Configuration of Qt Creator

To integrate all the different developer tools, the IDE needs to be configured. The procedure is specific to each one, we develop here the one of Qt Creator. The entry point is accessible through a menu, “Edit/Preferences...” (Windows), “Tools/Options...” (Linux) or “Qt Creator/Preferences...” (Mac).

### C.5.1 CMake and Qt

Since we use CMake, Qt Creator must know where it is installed (Figure C.3). Notice that here CMake was found under the Qt installation folder: some installers of Qt provide a version of CMake. The Qt libraries information come from a program called `qmake`, which Qt Creator needs to be aware of. Notice there is also a “Compilers” tab, where the different compilers found on the system are registered.

### C.5.2 Kit

A collection of tools to use is registered in a kit. It specifies the version of CMake, compilers and Qt version to use. Figure C.4 shows one kit. Several kits may be installed, each time a new project is built the different kits to use are proposed to the user. Pay attention to the “CMake Generator”: by default, it is frequently set as “Ninja”, which is an alternative to “make”, but is not generally installed, provoking a failure in the

---

<sup>11</sup>A very popular IDE nowadays is Visual Studio Code (VS Code). It can perfectly be used for Imagine++, but it requires some configuration: where to find Qt libraries, installing the extension to handle `cmake`, etc.

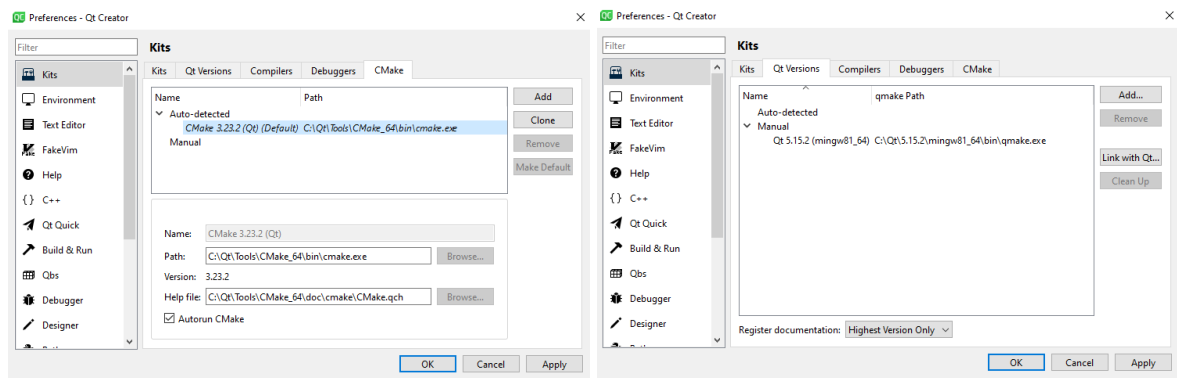


Figure C.3: CMake and Qt configuration in Qt Creator

build phase. This must be corrected by configuring the kit to use a “Makefile” (of type MinGW for Windows, Unix for Linux and Mac) and an extra generator.

### C.5.3 Project

For our purposes, a project is a directory with a file `CMakeLists.txt` and accompanying source and header files. Qt has no specific “project” format, it reuses the one of another IDE (CodeBlocks). This is a file with extension `.cbp` in the build directory, which we do not have to care about.

The first time a project is opened (through opening its file `CMakeLists.txt`), it creates a file `CMakeLists.txt.user` in the *source* directory. Among others, it specifies the kit that was used. The next time the project is opened, the presence of this file makes Qt Creator skip the proposition of a kit and the check that the compiler is working (which may be a bit slow under Windows). However, if the kit had some problems and we changed it, this `.user` file is out of sync. To reinit, it is generally a good idea to remove the file before opening the project.

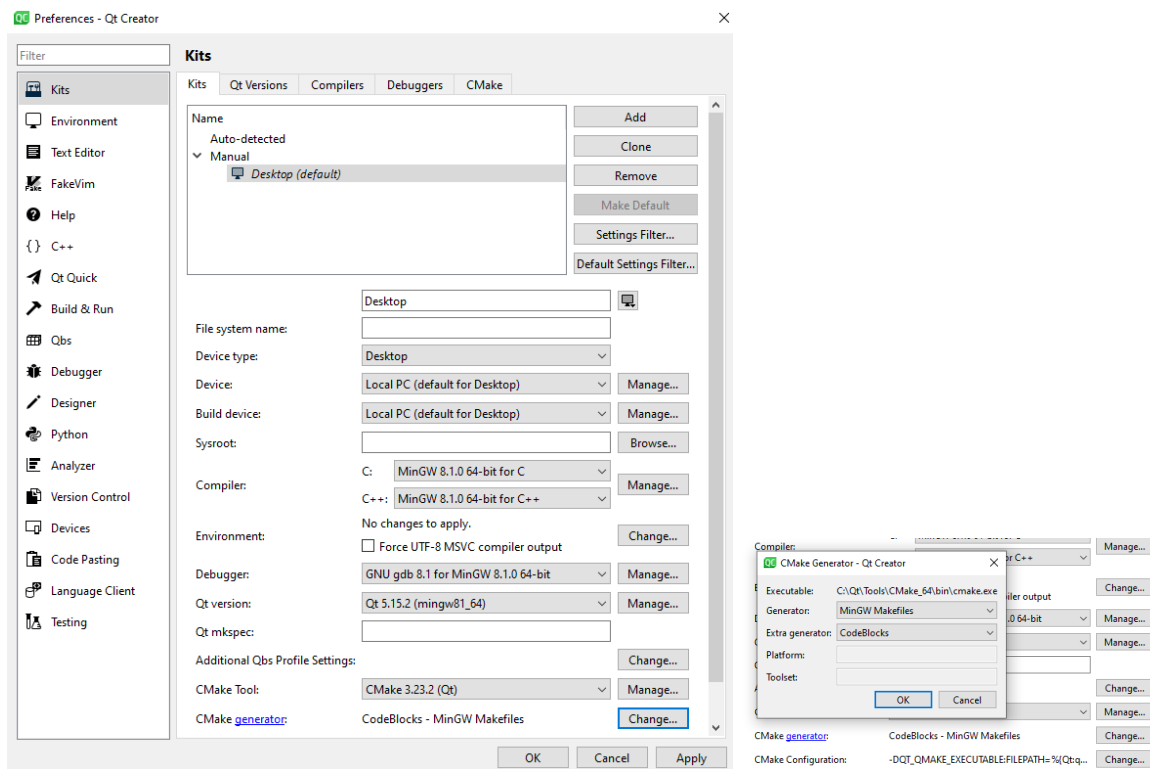


Figure C.4: A kit. Notice the CMake Generator, here the one for Windows. On Linux and Mac, the Generator must be “Unix Makefiles” and the Extra generator is the same.

# Appendix D

## Final reference card

| Reference Card (1/6)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Variables</b></p> <ul style="list-style-type: none"> <li>• <b>Definition:</b><br/> <pre>int i; int k,l,m;</pre> </li> <li>• <b>Assignment:</b><br/> <pre>i=2; j=i; k=l=3;</pre> </li> <li>• <b>Initialization:</b><br/> <pre>int n=5,o=n;</pre> </li> <li>• <b>Constants:</b><br/> <pre>const int s=12;</pre> </li> <li>• <b>Scope:</b><br/> <pre>int i; // i=j; forbidden! int j=2; i=j; // OK! if (j&gt;1) {     int k=3;     j=k; // OK! } //i=k; forbidden!</pre> </li> <li>• <b>Types:</b><br/> <pre>int i=3; double x=12.3; char c='A'; string s="hop"; bool t=true; float y=1.2f; unsigned int j=4; signed char d=-128; unsigned char d=25; complex&lt;double&gt;     z(2,3);</pre> </li> </ul> | <ul style="list-style-type: none"> <li>• <b>Global variables:</b><br/> <pre>int n; const int m=12; void f() {     n=10; // OK     int i=m; // OK     ... }</pre> </li> <li>• <b>Conversion:</b><br/> <pre>int i=int(x),j; float x=float(i)/j;</pre> </li> <li>• <b>Stack/Heap</b></li> <li>• <b>Enumerated type:</b><br/> <pre>enum Dir{N,E,S,W}; void advance(Dir d);</pre> </li> <li>• <b>Staic variables:</b><br/> <pre>int f() {     static bool once=true;     if (once) {         once=false;         ...     }     ... }</pre> </li> </ul> <hr/> <p><b>Tests</b></p> <ul style="list-style-type: none"> <li>• <b>Comparison:</b><br/> <pre>== != &lt; &gt; &lt;= &gt;=</pre> </li> <li>• <b>Negation: !</b></li> <li>• <b>Combinations: &amp;&amp;   </b></li> <li>• <code>if (i==0) j=1;</code></li> <li>• <code>if (i==0) j=1; else j=2;</code></li> </ul> | <ul style="list-style-type: none"> <li>• <code>if (i==0) { j=1; k=2; }</code></li> <li>• <code>bool t=(i==0); if (t) j=1;</code></li> <li>• <code>switch (i) { case 1: ...; break; case 2: ...; case 3: ...; break; default: ...; }</code></li> <li>• <code>mx=(x&gt;y)?x:y;</code></li> </ul> <hr/> <p><b>Loops</b></p> <ul style="list-style-type: none"> <li>• <code>do { ... } while(!ok);</code></li> <li>• <code>int i=1; while(i&lt;=100) { ... i=i+1; }</code></li> <li>• <code>for(int i=1;i&lt;=10;i++) ...</code></li> <li>• <code>for(int i=1, j=10; j&gt;i; i=i+2, j=j-3) ...</code></li> </ul> |

## Reference Card (2/6)

## Loops

- ```
for (int i=...)
  for (int j=...) {
    //skip case i==j
    if (i==j)
      continue;
    ...
  }
```
- ```
for (int i=...) {
 ...
 if (t[i]==s){
 // exit loop
 break;
 }
 ...
}
```

## Functions

- **Definition:**

```
int plus(int a,int b){
 int c=a+b;
 return c;
}
void display(int a) {
 cout << a << endl;
}
```
- **Declaration:**

```
int plus(int a,int b);
```
- **Return:**

```
int sign(double x) {
 if (x<0)
 return -1;
 if (x>0)
 return 1;
 return 0;
}
void display(int x,
 int y) {
 if (x<0 || y<0)
 return;
 if (x>=w || y>=h)
 return;
 DrawPoint(x,y,RED);
}
```
- **Call:**

```
int f(int a) { ... }
int g() { ... }
...
int i=f(2),j=g();
```
- **References:**

```
void swap(int& a,
 int& b){
```

- ```
int tmp=a;
a=b;b=tmp;
}
...
int x=3,y=2;
swap(x,y);
```
- **Overload:**

```
int chance(int n);
int chance(int a,
           int b);
double chance();
```
 - **Operators:**

```
vect operator+(
  vect A,vect B) {
  ...
  ...
  vect C=A+B;
```
 - **Call stack**
 - **Iterative/Recursive**
 - **Constant references (avoid copy):**

```
void f(const obj& x){
  ...
}
void g(const obj& x){
  f(x); // OK
}
```
 - **Default values:**

```
void f(int a,int b=0);
void g() {
  f(12); // f(12,0);
  f(10,2); // f(10,2);
}
void f(int a,int b) {
  // ...
}
```
 - **Inline (fast call):**

```
inline double
  sqr(double x) {
  return x*x;
}
...
double y=sqr(z-3);
```
 - **Return as reference:**

```
int i; // global var
int& f() {
  return i;
}
...
f()=3; // i=3!
```

Arrays

- **Definition:**

```
- double x[5],y[5];
  for(int i=0;i<5;i++)
    y[i]=2*x[i];
```
- ```
- const int n=5;
 int i[n],j[2*n];
```
- **Initialization:**

```
int t[4]={1,2,3,4};
string s[2]={"ab","c"};
```
- **Assignment:**

```
int s[3]={1,2,3},t[3];
for (int i=0;i<3;i++)
 t[i]=s[i];
```
- **As parameter:**

```
- void init(int t[4]){
 for(int i=0;i<4;i++)
 t[i]=0;
}
- void init(int t[],
 int n) {
 for(int i=0;i<n;i++)
 t[i]=0;
}
```
- **Variable size:**

```
int* t=new int[n];
...
delete[] t;
```
- **As argument:**

```
- void f(int* t,int n)
 { t[i]=... }
- void alloc(int*& t){
 t=new int[n];
}
```
- **2D:**

```
int A[2][3];
A[i][j]=...;
int A[2][3]=
 {{1,2,3},{4,5,6}};
void f(int A[2][2]);
```
- **2D in 1D:**

```
int A[2*3];
A[i+2*j]=...;
```
- **Variable size:**

```
int *t,*s,n;
```

| Reference Card (3/6)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Arrays</b></p> <ul style="list-style-type: none"> <li>• As argument (end):<br/> <pre>void f(const int* t,       int n) {     ...     s+=t[i]; // OK     ...     t[i]=...; // NO! }</pre> </li> </ul> <hr/> <p><b>Structures</b></p> <ul style="list-style-type: none"> <li>• struct Point {<br/> <pre>double x,y; Color c; }; ... Point a; a.x=2.3; a.y=3.4; a.c=RED; Point b={1,2.5,BLUE};</pre> </li> <li>• A structure is an objet fully public (→ see objects!)</li> </ul> <hr/> <p><b>Objects</b></p> <ul style="list-style-type: none"> <li>• struct obj {<br/> <pre>int x; // field int f(); // method int g(int y); }; int obj::f() {     int i=g(3); // my g     int j=x+i; // my x     return j; } ... int main() {     obj a;     a.x=3;     int i=a.f(); } </pre> </li> <li>• class obj {<br/> <pre>int x,y; void mine(); public: int z; void for_all(); void another(obj A); };</pre> </li> </ul> | <pre>void obj::mine() {     x=..; // OK     ..=y; // OK     z=..; // OK } void obj::for_all(){     x=..; // OK     mine(); // OK } void another(obj A) {     x=A.x; // OK     A.mine(); // OK } ... int main() {     obj A,B;     A.x=..; //NO     A.z=..; //OK     A.mine(); //NO     A.for_all(); //OK     A.another(B); //OK } </pre> <ul style="list-style-type: none"> <li>• class obj {<br/> <pre>obj operator+(obj B); }; ... int main() {     obj A,B,C;     C=A+B;     // C=A.operator+(B) } </pre> </li> <li>• Constant methods:<br/> <pre>void obj::f() const{     ... } void g(const obj&amp; x){     x.f(); // OK } </pre> </li> <li>• Constructor:<br/> <pre>class point {     int x,y; public:     point(int X,int Y); }; point::point(int X,              int Y){     x=X;     y=Y; } </pre> </li> </ul> | <pre>... point a(2,3); </pre> <ul style="list-style-type: none"> <li>• Empty constructor:<br/> <pre>obj::obj() {     ... } ... obj a;</pre> </li> <li>• Temporary objects:<br/> <pre>vec vec::operator+(     vec b) {     return vec(x+b.x,               y+b.y); } ... c=vec(1,2) +f(vec(2,3));</pre> </li> <li>• Accessors:<br/> <pre>class mat {     double *x; public:     double&amp; operator()         (int i,int j){         assert(i&gt;=0 ...);         return x[i+M*j];     }     double operator()         (int i,int j)const{         assert(i&gt;=0 ...);         return x[i+M*j];     }     ... } </pre> </li> </ul> <hr/> <p><b>Separate compilation</b></p> <ul style="list-style-type: none"> <li>• #include "vect.h", also in vect.cpp</li> <li>• Functions: declarations in .h, definitions in .cpp</li> <li>• Types: definitions in .h</li> <li>• Declare in .h only useful functions</li> <li>• #pragma once at beginning of header file</li> <li>• Do not cut too much...</li> </ul> |

## Reference Card (4/6)

## STL

- `min, max, ...`
- `complex<double> z;`
- `pair<int, string> p;`  
`p.first=2;`  
`p.second="hop";`
- `#include<list>`  
`using namespace std;`  
`...`  
`list<int> l;`  
`l.push_front(1);`  
`...`
- `if(l.find(3) != l.end())`  
`...`
- `list<int>::`  
`const_iterator it;`  
`for (it=l.begin();`  
`it != l.end(); it++)`  
`s += *it;`
- `list<int>::iterator it`  
`for (it=l.begin();`  
`it != l.end(); it++)`  
`if (*it == 2)`  
`*it = 4;`
- `stack, queue, heap,`  
`map, set, vector...`

## Input/Output

- `#include <iostream>`  
`using namespace std;`  
`...`  
`cout << "I=" << i << endl;`  
`cin >> i >> j;`
- `#include <fstream>`  
`using namespace std;`  
`ofstream f("hop.txt");`  
`f << 1 << ' ' << 2.3;`  
`f.close();`  
`ifstream g("hop.txt");`  
`if (!g.is_open()) {`  
`return 1;`  
`}`  
`int i;`  
`double x;`  
`g >> i >> x;`  
`g.close();`
- `do {`  
`...`  
`} while (!g.eof());`
- `ofstream f;`  
`f.open("hop.txt");`

- `double x[10], y;`  
`ofstream f("hop.bin",`  
`ios::binary);`  
`f.write((const char*)x,`  
`10*sizeof(double));`  
`f.write((const char*)&y,`  
`sizeof(double));`  
`f.close();`  
`ifstream g("hop.bin",`  
`ios::binary);`  
`g.read((char*)x,`  
`10*sizeof(double));`  
`g.read((const char*)&y,`  
`sizeof(double));`  
`g.close();`
- `string s;`  
`ifstream f(s.c_str());`
- `#include <sstream>`  
`using namespace std;`  
`stringstream f;`  
`// String to int`  
`f << s; f >> i;`  
`// int to string`  
`f.clear();`  
`f << i; f >> s; ⇔`  
`s = std::to_string(i);`
- `ostream& operator<<(<`  
`ostream& f,`  
`const point&p){`  
`f << p.x << ' ' << p.y;`  
`return f;`  
`}`  
`istream& operator>>(<`  
`istream& f, point& p){`  
`f >> p.x >> p.y;`  
`return f;`  
`}`

## Template





- **Functions:**  
`// To put in`  
`// file that uses`  
`// or in .h`  
`template <typename T>`  
`T maxi(T a, T b) {`  
`...`  
`}`  
`...`  
`// Type is found`  
`// alone!`  
`maxi(1, 2); //int`  
`maxi(.2, .3); //double`  
`maxi("a", "c"); //string`

- **Objects:**  
`template <typename T>`  
`class pair {`  
`T x[2];`  
`public:`  
`pair() {}`  
`pair(T a, T b) {`  
`x[0]=a; x[1]=b;`  
`}`  
`T add() const;`  
`};`  
`...`  
`template <typename T>`  
`T pair<T>::add() const {`  
`return x[0]+x[1];`  
`}`  
`...`  
`// Type must be`  
`// precised!`  
`pair<int> a(1, 2);`  
`int s=a.add();`  
`pair<double> b;`  
`...`
- **Multiple:**  
`template <typename T,`  
`typename S>`  
`class hop {`  
`...`  
`};`  
`...`  
`hop<int, string> A;`  
`...`
- **Integers:**  
`template <int N>`  
`class hop {`  
`..`  
`};`  
`...`  
`hop<3> A;`  
`...`

## Advice

- **Errors/warnings: click.**
- **Indent!**
- **Fix warnings.**
- **Use the debugger.**
- **Write functions.**
- **Arrays: not to translate math formula!**
- **Make structures.**



| Reference Card (5/6)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Advice</b></p> <ul style="list-style-type: none"> <li>• Do separate source files</li> <li>• The .h must be enough for the user (who must not look into .cpp)</li> <li>• Don't abuse recursion.</li> <li>• Don't forget delete.</li> <li>• Compile regularly.</li> <li>• <code>#include &lt;cassert&gt;</code><br/>...<br/><code>assert(x!=0);</code><br/><code>y=1/x;</code></li> <li>• Make objects.</li> <li>• Do not always make objects!</li> <li>• Think interface / implementation / usage.</li> </ul> <hr/> <p><b>Keys</b></p> <ul style="list-style-type: none"> <li>• Debug: F5 </li> <li>• Step over: F10 </li> <li>• Step inside: F11 </li> <li>• Indent: Ctrl+A, Ctrl+I</li> <li>• Switch source/header: F4</li> <li>• Switch decl./def.: F2</li> </ul> | <ul style="list-style-type: none"> <li>• Step out: Maj+F11 </li> <li>• Gest. tâches: Ctrl+Maj+Ech</li> </ul> <hr/> <p><b>Misc.</b></p> <ul style="list-style-type: none"> <li>• <code>i++;</code><br/><code>i--;</code><br/><code>i-=2;</code><br/><code>j+=3;</code></li> <li>• <code>j=i%n; // Modulo</code></li> <li>• <code>#include &lt;cstdlib&gt;</code><br/>...<br/><code>i=rand() %n;</code><br/><code>x=rand() /</code><br/><code>double(RAND_MAX);</code></li> <li>• <code>#include &lt;ctime&gt;</code><br/><code>// Single call</code><br/><code>srand((unsigned int)</code><br/><code>time(0));</code></li> <li>• <code>#include &lt;cmath&gt;</code><br/><code>double sqrt(double x);</code><br/><code>double cos(double x);</code><br/><code>double sin(double x);</code><br/><code>double acos(double x);</code></li> <li>• <code>#include &lt;string&gt;</code><br/><code>using namespace std;</code><br/><code>string s="hop";</code><br/><code>char c=s[0];</code><br/><code>int l=s.size();</code><br/><code>if (s1==s1) ...</code></li> </ul> | <pre> if (s1!=s2) ... if (s1&lt;s2) ... size_t i=s.find('h'),       j=s.find('h',3);       k=s.find("hop");       l=s.find("hop",3); a="how"; b="are you?"; txt=a+" "+b; s1="one two three"; s2=string(s1,4,3); getline(cin,s); getline(cin,s,':'); const char *t=s.c_str(); </pre> <ul style="list-style-type: none"> <li>• <code>#include &lt;ctime&gt;</code><br/><code>s=double(clock())</code><br/><code>/CLOCKS_PER_SEC;</code></li> <li>• <code>#include &lt;cmath&gt;</code><br/><code>double pi=M_PI;</code></li> <li>• <b>Bitwise operators</b><br/>and: <code>a&amp;b</code><br/>or: <code>a b</code><br/>xor: <code>a^b</code><br/>right shift: <code>a&gt;&gt;n</code><br/>left shift: <code>a&lt;&lt;n</code><br/>complement: <code>~a</code></li> <li>• <b>examples:</b><br/><code>set(i,1): i =(1&lt;&lt;n)</code><br/><code>reset(i,1): i&amp;=~(1&lt;&lt;n)</code><br/><code>test(i,1): if (i&amp;(1&lt;&lt;n))</code><br/><code>flip(i,1): i^=(1&lt;&lt;n)</code></li> </ul> |

## Reference Card (5/6)

## Frequent errors

- No definition of function inside a function!
- `int q=r=4; // NO!`
- `if (i=2) // NO!`  
`if i==2 // NO!`  
`if (i==2) then // NO!`
- `for(int i=0,i<100,i++)`  
`// NO!`
- `int f() {...}`  
`int i=f; // NO!`
- `double x=1/3; // NO!`  
`int i,j;`  
`x=i/j; // NO!`  
`x=double(i/j); //NO!`
- `double x[10],y[10];`  
`for(int i=1;i<=10;i++)`  
`y[i]=2*x[i];//NO`
- `int n=5;`  
`int t[n]; // NO`
- `int f()[4] { // NO!`  
`int t[4];`  
`...`  
`return t; // NO!`  
`}`  
`int t[4]; t=f();`
- `int s[3]={1,2,3},t[3];`  
`t=s; // NO!`
- `int t[2];`  
`t={1,2}; // NO!`
- `struct Point {`  
`double x,y;`  
`} // NO!`
- `Point a;`  
`a={1,2}; // NO!`
- `#include "tp.cpp"//NO`
- `int f(int t[][]);//NO`  
`int t[2,3]; // NO!`  
`t[i,j]=...; // NO!`
- `int* t;`  
`t[1]=...; // NO!`
- `int* t=new int[2];`  
`int* s=new int[2];`  
`s=t; // lost s!`  
`delete[] t;`  
`delete[] s;//Crash!`
- `int *t,s;// s is int`  
`// not int*`  
`t=new int[n];`  
`s=new int[n];// NO!`
- `class vec {`  
`int x,y;`  
`public:`  
`...`  
`};`  
`...`  
`vec a={2,3}; // NO`
- `vec v=vec(1,2);//NO`  
`vec v(1,2); // Yes`
- `//NO!`  
`void f(int a=2,int b);`
- `void f(int a,int b=0);`  
`void f(int a);// NO!`
- Not anything inline!
- `int f() {`  
`...`  
`}`  
`f()=3; // HORROR!`
- `int& f() {`  
`int i;`  
`return i;`  
`}`  
`f()=3; // NO!`
- `if (i>0 & i<n) // NO`  
`if (i<0 | i>n) // NO`
- `if (...) {`  
`...`  
`if (...)`  
`break; // No,`  
`// loops only`  
`}`
- `for (i ...)`  
`for (j ...) {`  
`...`  
`if (...)`  
`break;//NO, exit`  
`// loop j only`  
`}`
- `int i;`  
`double x;`  
`j=max(i,0);//KO`  
`y=max(x,0);//NO`  
`// 0.0 and not 0: max`  
`// is an STL template`

## Imagine++

- See documentation...