Algorithmique et Programmation

—

Examen sur machine

—

G1: keriven*(at)*cermics.enpc.fr G2: Jean-Philippe.Pons*(at)*sophia.inria.fr
G3: charpiat*(at)*clipper.ens.fr G4: thomas.deneux*(at)*ens.fr
G5: pierre*(at)*senellart.com G6: gmellier*(at)*melix.org

—

09/01/04

—

1 Crible d'Ératosthène

Rappel : *un entier naturel est dit premier s'il n'est divisible que par 1 et lui-même, et qu'il est différent de 1.*

Le but de cet exercice est de dresser par ordre croissant la liste des nombres premiers. On utilisera pour cela le crible d'Ératosthène, qui repose essentiellement sur le fait que les diviseurs éventuels d'un entier sont plus petits que lui. La méthode consiste à parcourir dans l'ordre croissant la liste des nombres entiers candidats (par exemple initialement tous les nombres de 2 à 999, si l'on se restreint aux entiers inférieurs à 1000), et, à chaque nombre rencontré, à retirer de la liste des nombres candidats tous les multiples de celui-ci. Une fois la liste parcourue, il ne restera que les nombres premiers.

On recherche les nombres premiers inférieurs ou égaux à n, n étant un nombre entier supérieur à 2, intialement fixé à 100. Plutôt que de gérer une liste d'entiers et d'en enlever des nombres, on travaillera avec un tableau de n booléens avec la convention que la i^{e} case du tableau contiendra `true` si i est premier, et `false` sinon.

1. Partir d'un projet "console" (`Win 32 Basic Console`).
2. Dans la fonction `main`, créer le tableau de booléens et le remplir avec des `true`.
3. Créer une fonction `multiples` qui prend en argument le tableau, sa taille et un entier a, et qui tourne à `false` les éléments du tableau correspondants à des multiples de a (excepté a, bien sûr).
4. Utiliser la fonction `multiples` de façon appropriée dans la fonction `main`, dans une boucle parcourant le tableau, afin d'exécuter le crible d'Ératosthène. Il est inutile de considérer les multiples des nombres qui ne sont pas premiers (à ce propos ne pas oublier que 1 consitue un cas particulier).
5. Afficher le nombre de nombres premiers inférieurs ou égaux à 211, ainsi que les nombres premiers en question.
6. Vérifier que l'on peut s'arrêter à $\sqrt{n}$.

2 Calcul de π par la méthode de Monte Carlo

On désigne par *méthode de Monte Carlo* une méthode de résolution d'un problème mathématique à l'aide de suites de nombres aléatoires convergeant vers le résultat, en référence aux nombreux casinos monégasques. Le méthode de Monte Carlo classique pour calculer une valeur approchée de π consiste à tirer aléatoirement un grand nombre n de fois des points (x, y) du plan avec $-1 \leq x \leq 1$, $-1 \leq y \leq 1$ puis de déterminer quelle est la proportion p_n de ces points qui sont situés dans le disque unité. Comme la surface de ce dernier vaut π, on peut montrer que $lim_{n\to+\infty} p_n = \frac{\pi}{4}$. Le but de cet exercice est de programmer ce calcul et une visualisation graphique de celui-ci.

Fonctions utiles

– *Nombres pseudo-aléatoires*

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

```
void srand(unsigned int seed); // Initialise le gestionnaire de nombres
                               // pseudo-aléatoires

int rand(); // Retourne un nombre pseudo-aléatoire entre 0 et RAND_MAX, inclus
```

– *Temps*

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

long time(...); // time(0) renvoie le nombre de secondes écouées
                // depuis le 1er janvier 1970
```

– *WinLib*

```
#include <win>
using namespace Win;

void DrawPoint(int x,int y,const Color& col); // Affiche un pixel à
                                              // l'écran d'une certaine couleur

void DrawCircle(int xc,int yc,int r,const Color& col); // Affiche un cercle
                                                       // centré en (xc,yc),
                                                       // de rayon r
                                                       // et de couleur col
```

1. Ajouter un nouveau projet WinLib nommé `MonteCarlo` à la solution et le définir cmme projet de démarrage.
2. Définir dans `main.cpp` une constante globale `taille_fenetre` et l'utiliser dans l'appel à `OpenWindow`. Dans tout ce qui suit, la fenêtre sera carrée, représentant l'ensemble des points dont les coordonnées sont entre -1 et 1.
3. Ajouter au projet deux fichiers `Point.h` et `Point.cpp` dans lesquels vous déclarerez et définirez à l'endroit approprié :
   (a) une structure PointPlan représentant des points du plan (il faut en particulier pouvoir avoir des coordonnées entre -1 et 1)
   (b) une fonction GenerePoint retournant un PointPlan dont les coordonnées sont tirées aléatoirement entre -1 et 1. *Ne pas oublier qu'en C++, la division de deux entiers est la division entière !*
   (c) une fonction AffichePoint, prenant en argument la taille de la fenêtre d'affichage, une couleur et un PointPlan, et qui affiche ce dernier à l'écran dans la couleur précisée (*penser à renormaliser !*).
4. Appeler la procédure d'initialisation du gestionnaire de nombre pseudo-aléatoires au début de `main` avec `srand(unsigned int(time(0)));`. Penser aux `#include` et `using namespace` correspondants.
5. Dessiner à l'écran le cercle unitaire au début de `main.cpp`.
6. Définir une constante globale `nb_iterations` dans `main.cpp` représentant le nombre d'itérations effectuées. Construire une boucle effectuant `nb_iterations` fois les opérations suivantes :
   (a) Générer un point pseudo-aléatoire
   (b) Incrémenter une variable `compteur` si ce point est dans le disque unitaire (c'est-à-dire si $x^2+y^2 \leq 1$). On rajoutera une fonction retournant le carré de la norme d'un point.
   (c) Afficher ce point à l'écran, en bleu s'il est dans le disque, en rouge sinon.
7. Afficher à la fin du programme la valeur de π déduite.
8. Varier `nb_iterations` afin d'avoir un bon compromis temps d'affichage / précision de la valeur calculée. Que constate-t-on ? Pourquoi ? (Répondre en commentaire du programme)

3 Serpent

Le but de cet exercice est de programmer un serpent qui se déplace sur un terrain rectangulaire en évitant de se mordre lui-même. Pour cela structures et tableaux seront utiles. Ne pas oublier de compiler et tester à chaque étape... NB : certaines questions sont indépendantes.

1. *Structure :* Partir d'un projet WinLib. Créer une structure `point` mémorisant deux coordonnées entières `x` et `y`.

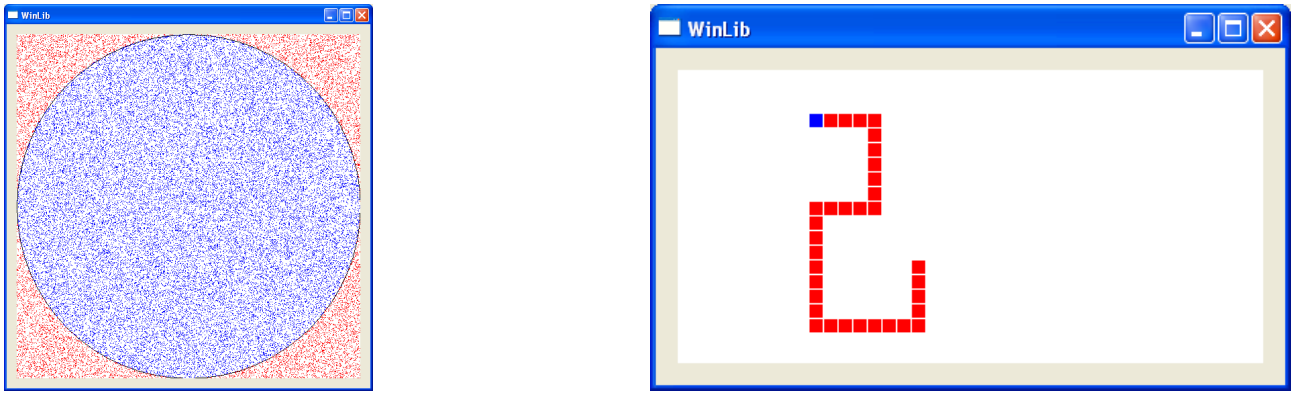

FIG. 1 – A gauche : calcul de π. A droite : un serpent.

2. *Dessin :* On va dessiner un serpent de déplaçant sur un terrain de dimensions `(w,h)`. Pour bien le visualiser, on l'affichera dans un fenêtre de dimensions `(w*z,h*z)` (Prendre par exemple `(w,h,z)=(40,30,10)`).
   (a) Programmer une fonction `void dessine(point p,int zoom,Color c)` utilisant la fonction `FillRect(x,y,w,h,c)` pour dessiner un carré plein de coin supérieur gauche `(p.x*zoom,p.y*zoom)`, de côté `zoom-1` et de couleur `c`.
   (b) Ecrire une fonction `void dessine(point s[],int n,int zoom)` dessinant un serpent constitué des points `s[0]` à `s[n-1]`, `s[0]` étant la tête. On pourra tracer le corps du serpent en rouge et sa tête en bleu (figure 1).
   (c) Initialiser un serpent dans un tableau de dimension constante `n=10` et l'afficher.
3. *Divers :* programmer quatre fonctions utiles pour la suite :
   (a) Une fonction `bool operator==(point a,point b)` retournant vrai si les points `a` et `b` sont égaux, ce qui permettra d'écrire un test comme `if (p1==p2)`.
   (b) Une fonction `bool cherche(point s[],int n,point p)` retournant vrai si le point `p` se trouve dans le tableau `s` de taille `n`.
   (c) Une fonction `void decale(point s[],int n,point p)` qui décale le tableau `s` vers le haut (i.e. `s[i]=s[i-1]`) et range `p` dans `s[0]`. Cette fonction sera utile pour la suite !
   (d) Une fonction `point operator+(point a,point b)` calculant la somme de deux points (considérés comme des vecteurs).
4. *Avancer d'une case :* Le serpent peut avancer dans une des quatre directions. On mémorise la direction de son déplacement par un entier de 0 à 3, avec la convention `(0,1,2,3)=(est,sud,ouest,nord)`.
   (a) Créer et initialiser une variable globale `const point dir[4]` telle que `dir[d]` correspond à un déplacement dans la direction `d`. Ainsi, `d[0]={1,0}`, `d[1]={0,1}`, etc.
   (b) Ecrire et tester une fonction `void avance(point s[],int n,int d,int zoom)` utilisant ce qui précède pour :
   – Avancer le serpent `s,n` dans la direction `d`.
   – Afficher sa nouvelle position sans tout re-dessiner (effacer la queue et dessiner la tête).
5. *Avancer :* Ecrire les quatre fonctions suivantes :
   – `bool sort(point a,int w,int h)` qui retourne vrai si le point `a` est en dehors du terrain.
   – `void init_rand()` qui initialise le générateur aléatoire.
   – `void change_dir(int& d)` qui change aléatoirement `d` en `d-1` une fois sur 20, en `d+1` une fois sur 20 et ne modifie pas `d` les 18 autres fois, ce qui, compte tenu de la convention adoptée pour les directions, revient à tourner à droite ou à gauche de temps en temps. Attention à bien conserver `d` entre 0 et 3.
   – `bool ok_dir(point s[],int n,int d,int w,int h)` qui teste si le déplacement du serpent dans la direction `d` ne va pas le faire sortir.
   puis les utiliser pour faire avancer le serpent dans une boucle de 500 pas de temps, en vérifiant qu'il ne sort pas du terrain. Penser à rajouter un `MilliSleep()` si le déplacement est trop rapide (sur certains PC de l'ENPC, la WinLib ne sera pas à jour et l'affichage sera lent et saccadé... Ignorer.)
6. *Ne pas se manger :* modifier `ok_dir()` pour que le serpent ne se rentre pas dans lui-même.
7. *Escargot :* le serpent peut se retrouver coincé s'il s'enroule sur lui-même. Programmer une fonction `bool coince(point s[],int n,int w,int h)` qui teste si aucune direction n'est possible et modifier la boucle principale pour terminer le programme dans ce cas.

8. *Grandir* (question finale difficile) :
   - Changer le programme pour que le tableau `s` soit alloué dans le tas (`n` pourra alors être variable).
   - Un pas de temps sur 20, appeler, à la place de `avance()`, une fonction `void allonge()` qui avance dans le direction `d` **sans supprimer la queue**. Du coup, `s` doit être réalloué et `n` augmente de 1. Ne pas oublier de désallouer l'ancienne valeur de `s`. Bien gérer l'affichage, etc. Cette fonction sera déclarée `void allonge(point*& s,int& n,int d,int zoom)`.