Introduction à la Programmation

—

TP #1

—

G1: monasse*(at)*imagine.enpc.fr G2: nicolas.audebert*(at)*onera.fr
G3: alexandre.boulch*(at)*onera.fr G4: maxime.ferrera*(at)*onera.fr
G5: laurent.bulteau*(at)*u-pem.fr G6: pierre-alain.langlois*(at)*eleves.enpc.fr

1 L'environnement de programmation

1.1 Bonjour, Monde !

1. *Connexion :*
   Se connecter sous Windows ou Linux.
2. *Projet :*
   Télécharger l'archive `Tp1_Initial.zip` sur la page du cours, la décompresser sur le bureau. Le fichier `CMakeLists.txt` décrit le projet. La ligne qui nous intéresse est la suivante :

   ```
   add_executable(Tp1 Tp1.cpp)
   ```

   Elle indique que le programme s'appellera `Tp1` (`Tp1.exe` sous Windows) et que le code source pour le construire est dans le fichier `Tp1.cpp`.
   (a) Visual (donc sous Windows) n'est pas capable d'interpréter le fichier `CMakeLists.txt`, il faut utiliser le programme `Cmake` au préalable. Lancer donc ce programme, et aller chercher comme répertoire "source code" `Tp1_Initial` par le bouton "`Browse source...`". C'est une bonne idée de séparer les fichiers générés des sources : sélectionner dans la ligne suivante un nouveau répertoire "Build". Cliquer "Generate" et sélectionner le bon générateur (Visual Studio 2008, sur les machines de l'École). Si tout s'est bien passé, on peut fermer `Cmake` et ouvrir le fichier "solution" (extension `.sln`) dans le répertoire Build, ce qui lance "`Visual Studio`"
   (b) Sous Linux, lancer Kdevelop, le menu "Open Project" et choisir le `CMakeLists.txt`. Les choix proposés par défaut (en particulier le mode "Debug") sont corrects. Kdevelop comprend le format `CMakeLists` et est capable de lancer `Cmake` lui-même.
   (c) QtCreator (Linux, Windows ou Mac) connaît également `Cmake`, donc procédez comme avec Kdevelop.
3. *Programmation :*
   (a) Rajouter `cout << "Hello"<< endl;` sur la ligne avant `return 0;`
4. *Génération :*
   (a) Dans la fenêtre "Solution explorer" de Visual Studio, rechercher et afficher le fichier `Tp1.cpp`.
   (b) "`Build/Build solution`", ou "`F7`" ou bouton correspondant.
   (c) Vérifier l'existence d'un fichier `Tp1` (`Tp1.exe` sous Windows) dans `Build/Tp1` (`Build/Tp1/Debug` sous Windows).
5. *Exécution :*
   (a) Lancer le programme (sous Visual) avec "`Debug/Start Without Debugging`" (ou "`Ctrl+F5`" ou bouton correspondant). Il faut d'abord lui préciser quel programme lancer (clic droit sur le projet Tp1 dans l'explorateur de solution, "Sélectionner comme projet de démarrage") Une fenêtre à fonds noir "console" s'ouvre, dans laquelle le programme s'exécute, et la fenêtre console reste ouverte jusqu'à l'appui d'une touche du clavier.
   (b) Vérifier qu'on a en fait créé un programme indépendant qu'on peut lancer dans une fenêtre de commande :
   — Essayer de le lancer depuis le gestionnaire de fichiers standard : le programme se referme tout de suite !
   — Dans les menus Windows : "`Démarrer/Exécuter`"
   — "`Ouvrir: cmd`"

— Taper "`D:`",[1] puis
"`cd \Documents and Settings\login\Bureau\Tp1\Tp1\Debug`"
(apprenez à profiter de la complétion automatique avec la touche `TAB`).
— Vérifier la présence de `Tp1.exe` avec la commande "`dir`".
— Taper "`Tp1`".

(c) Sous Kdevelop (Linux), il faut commencer par aller dans le menu "Configure launches", ajouter avec le bouton "+" et sélectionner l'exécutable. On peut alors le lancer avec le bouton "Execute".

Touche utile (Visual) : Ctrl+F5 =  = Start without debugging

6. *Fichiers :*
On a déjà suivi la création des fichiers principaux au fur et à mesure. Constater la présence de `Tp1.obj` qui est la compilation de `Tp1.cpp` (que le linker a ensuite utilisé pour créer Tp1.exe). Voir aussi la présence de nombreux fichiers de travail. Quelle est la taille du répertoire de Build de `Tp1` (clic droit + propriétés) ?

7. *Nettoyage :*
Supprimer les fichiers de travail et les résultats de la génération avec "`Build / Clean solution`" puis fermer Visual Studio. Quelle est la nouvelle taille du répertoire ?

8. *Compression :*
Sous Windows, en cliquant à droite sur le répertoire `Tp1`, fabriquer une archive comprimée `Tp1.zip` (ou `Tp1.7z` suivant la machine). Attention il faut quitter Visual Studio avant de comprimer. Il peut sinon y avoir une erreur ou certains fichiers trop importants peuvent subsister.

Il faut quitter Visual Studio avant de comprimer.

Une telle archive, comprenant tous les fichiers nécessaires à la compilation, sera le format utilisé pour déposer vos exercices et rendus de TP sur Educnet.

Notez bien qu'avec `Cmake` nous avons deux dossiers :
— Le dossier *source* contenant les fichiers `Tp1.cpp` et `CMakeLists.txt`;
— Le dossier *build* que vous avez choisi au démarrage de `Cmake`.
Le plus important est le premier, puisque le deuxième peut toujours être régénéré avec `Cmake`. N'envoyez à votre enseignant que le répertoire source, il recompilera lui-même. Votre *build* lui est probablement inutile car il n'utilise pas le même système que vous. Ainsi, quand vous avez terminé un projet ou un TP, n'hésitez pas à nettoyer en supprimant votre dossier *build*, mais gardez précieusement votre dossier *source*, c'est celui-ci qui représente le résultat de votre travail. Bien que `Cmake` autorise d'utiliser un même répertoire pour les deux, c'est à éviter, pour bien séparer les sources et les fichiers générés automatiquement. Pour comprendre un peu comment tous ces outils se coordonnent, on peut se reporter à l'annexe C du polycopié.

1.2 Premières erreurs

1. *Modifier le programme :*
Modifier le programme en changeant par exemple la phrase affichée.

(a) Tester une nouvelle génération/exécution. Vérifier que Visual Studio sauve le fichier automatiquement avant de générer.

(b) Modifier à nouveau le programme. Tester directement une exécution. Visual Studio demande automatiquement une génération !

Lancer directement une exécution sauve et génère automatiquement. Attention toutefois de ne pas confirmer l'exécution si la génération s'est mal passée.

2. *Erreurs de compilation*
Provoquer, constater et apprendre à reconnaître quelques erreurs de compilation :

(a) `includ` au lieu de include

(b) `iostrem` au lieu de iostream

(c) Oublier le `;` après std

(d) `inte` au lieu de int

(e) `cou` au lieu de cout

(f) Oublier les guillemets " fermant la chaîne "Hello ... "

(g) Rajouter une ligne `i=3;` avant le return.

A ce propos, il est utile de découvrir que :

1. Sur certaines machines, il faut en fait aller sur C :, vérifiez en regardant où est installé votre projet

Double-cliquer sur un message d'erreur positionne l'éditeur sur l'erreur.

3. *Erreur de linker*
   Il est un peu tôt pour réussir à mettre le linker en erreur. Il est pourtant indispensable de savoir différencier ses messages de ceux du compilateur. En général, le linker indiquera une erreur s'il ne trouve pas une fonction ou des variables parce qu'il manque un fichier objet ou une bibliothèque. C'est aussi une erreur s'il trouve deux fois la même fonction...
   (a) Rajouter une ligne `f(2);` avant le return et faire `Ctrl+F7`. C'est pour l'instant une erreur de compilation.
   (b) Corriger l'erreur de compilation en rajoutant une ligne (pour l'instant *"magique"*)
   `void f(int i);` avant la ligne avec main. Compiler sans linker : il n'y a plus d'erreur. Générer le programme : le linker constate l'absence d'une fonction `f()` utilisée par la fonction main() qu'il ne trouve nulle part.

4. *Indentations :*
   Avec toutes ces modifications, le programme ne doit plus être correctement "indenté". C'est pourtant essentiel pour une bonne compréhension et repérer d'éventuelle erreur de parenthèses, accolades, etc. Le menu `Edit/Advanced` fournit de quoi bien indenter.

Pour repérer des erreurs, toujours bien indenter.
Ctrl+K,Ctrl+F (Visual) = Ctrl+I (QtCreator) = indenter la zone sélectionnée.
Ctrl+A,Ctrl+K,Ctrl+F (Visual) = Ctrl+A,Ctrl+I (QtCreator) = tout indenter.

5. *Warnings du compilateur*
   En modifiant le main(), provoquer les warnings suivants : [2]
   (a)
   ```
   int i;
   i=2.5;
   cout << i << endl;
   ```
   Exécuter pour voir le résultat.
   (b)
   ```
   int i;
   i=4;
   if (i=3) cout << "salut"<< endl;
   ```
   Exécuter !
   (c)
   ```
   int i,j;
   j=i;
   ```
   Exécuter (répondre "abandonner" !)
   (d) Provoquer le warning inverse : variable déclarée mais non utilisée.
   (e) Ajouter `exit;` comme première instruction de `main`. Appeler une fonction en oubliant les arguments arrive souvent ! Exécuter pour voir. Corriger en mettant exit (0);. Il y a maintenant un autre warning. Pourquoi ? (La fonction exit () quitte le programme en urgence !)

Il est très formellement déconseillé de laisser passer des warnings ! Il faut les corriger au fur et à mesure. Une option du compilateur propose même de les considérer comme des erreurs !

1.3 Debugger

Savoir utiliser le debuggeur est essentiel. Il doit s'agir du premier réflexe en présence d'un programme incorrect. C'est un véritable moyen d'investigation, plus simple et plus puissant que de truffer son programme d'instructions supplémentaires destinées à espionner son déroulement.

1. Taper le `main` suivant.

```
int main()
{
    int i,j,k;
    i=2;
    j=3*i;
    if (j==5)
        k=3;
    else
```

2. Certains de ces warnings ne se manifestent qu'en niveau d'exigence le plus élevé : pour le mettre en place, clic droit sur le projet dans la fenêtre de gauche, menu "Propriétés", onglet C++, sélectionner le "warning level" 4 (3, moins exigent, est le défaut).

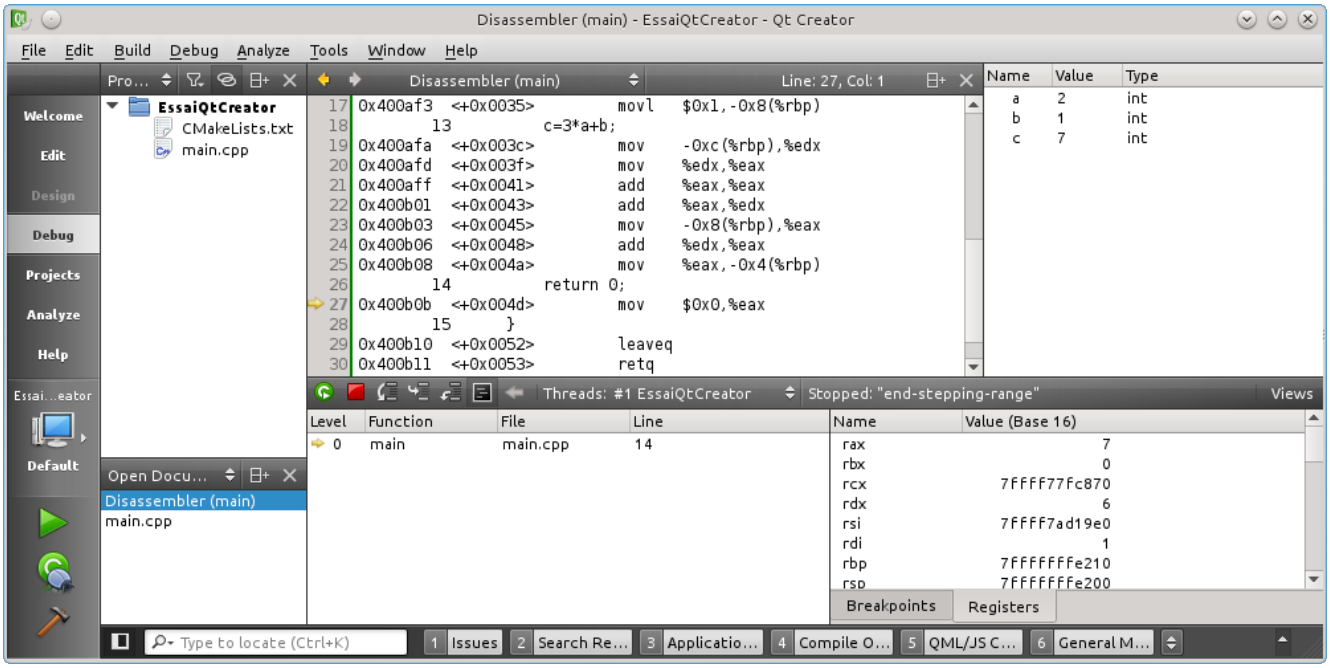

FIGURE 1 – QtCreator montrant le code machine et les registres

```
        k=2;
    return 0;
}
```

2. Pour pouvoir utiliser le debuggeur avec QtCreator ou Kdevelop, il faut compiler en mode Debug. Cela se fait en modifiant une variable de `Cmake`. Sous QtCreator, aller dans l'onglet Projects et ajouter comme argument :

   `-DCMAKE_BUILD_TYPE=Debug`

   Cela écrit dans le fichier `CMakeCache.txt` généré par `Cmake`.
3. Lancer le programme "sous le debuggeur" avec `Build/Start` ou `F5` ou le bouton correspondant. Que se passe-t-il ?
4. Placer un "point d'arrêt" en cliquant dans la colonne de gauche à la hauteur de la ligne i=2; puis relancer le debuggeur.
5. Avancer en `"Step over"` avec `F10` ou le bouton correspondant et suivre les valeurs des variables (dans la fenêtre spéciale ou en plaçant (sans cliquer!) la souris sur la variable).
6. A tout moment, on peut interrompre l'exécution avec `Stop` ou `Maj+F5`. Arrêter l'exécution avant d'atteindre la fin de main sous peine de partir dans la fonction qui a appelé main!
7. Placer un deuxième point d'arrêt et voir que `F5` redémarre le programme jusqu'au prochain point d'arrêt rencontré.
8. Ajouter i=max(j,k); avant le return. Utiliser `"Step into"` ou `F11` ou le bouton correspondant quand le curseur est sur cette ligne. Constater la différence avec `F10`.
9. Enfin, pour voir à quoi ressemble du code machine, exécuter jusqu'à un point d'arrêt puis faire `Debug/Windows/Disassembly`. On peut aussi dans ce même menu voir les registres du micro-processeur. Il arrive qu'on se retrouve dans la fenêtre "code machine" sans l'avoir demandé quand on debugge un programme pour lequel on n'a plus le fichier source. Cet affichage est en fait très utile pour vérifier ce que fait le compilateur et voir s'il optimise bien.
10. Pour faire l'étape précédente sous QtCreator, on peut sélectionner l'option "Operate by Instruction" du menu Build . Les registres sont visibles en allant dans le menu Window puis Views. Voir Figure 1.

Touches utiles :	**F5**	=		= **Debug**
	F10	=		= **Step over**
	F11	=		= **Step inside**

1.4 S'il reste du temps

Téléchargez le programme supplémentaire `Tp1_Final.zip` sur la page du cours (`http://imagine.enpc.fr/~monasse/Info`), jouez avec...et complétez-le !

1.5 Installer Imagine++ chez soi

Allez voir sur `http://imagine.enpc.fr/~monasse/Imagine++`. Installez sur votre ordinateur portable et essayez de refaire ce TP avec QtCreator.