Introduction à la Programmation

— TP #1 —

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` depuis la page du cours, la décompresser sur le bureau. Le fichier `CMakeLists.txt` décrit le projet. La ligne qui nous intéresse de ce fichier texte 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`.
   `QtCreator` est l'environnement de programmation que nous utiliserons, il a l'avantage de fonctionner sur les principales plates-formes (Linux, Windows ou Mac) et de comprendre nativement `Cmake`.[1] Pour démarrer, lancez `QtCreator` et choisissez "Open Project" pour aller chercher le fichier `CMakeLists.txt`. Regardez en bas de la fenêtre l'onglet 

, [2] qui doit afficher tout un tas de messages : c'est le résultat de la lecture du fichier et la création d'un dossier de "build", où seront mis tous les fichiers générés par `QtCreator`. Cela veut dire aussi que ce dossier n'est pas crucial, car il sera simple à regénérer : quand on a fini un code, on peut sans problème nettoyer en supprimant ce dossier.
3. *Programmation :* insérez une ligne de commentaire avec votre nom au début du fichier `Tp1.cpp`. Les commentaires sont introduits par // et vont jusqu'à la fin de la ligne. Ils sont ignorés par le compilateur, mais utiles pour le programmeur. Rajoutez ensuite `cout << "Hello"<< endl;` sur la ligne avant `return 0;` dans le fichier `Tp1.cpp`.
4. *Génération :* Appuyez sur l'outil marteau  en bas à droite pour générer le programme. Regardez dans l'onglet 4 "Compile Output", il vous dit qu'il a créé un fichier `Tp1.cpp.o`. Constatez avec un gestionnaire de fichiers que le fichier est bien dans un sous-dossier du "build". Observez aussi que le programme `Tp1` est aussi présent.
5. *Exécution :*
   (a) Lancez le programme avec la flèche verte dans `QtCreator`, observez le résultat dans l'onglet 3 "Application Output".
   (b) Vérifiez qu'on a en fait créé un programme indépendant qu'on peut lancer dans une fenêtre de commande :
   — Ouvrez un terminal (sous Windows : `"Démarrer/Exécuter"`, lancez la commande "`cmd`")
   — Sous Windows, tapez `"D:"`, [3] puis
   `"cd \Documents and Settings\login\Bureau\build-Tp1...\Tp1"`
   (apprenez à profiter de la complétion automatique avec la touche `TAB`). Sous Linux, le chemin ne sera pas le même, et remplacez les \ par des /.
   — Vérifiez la présence de `Tp1.exe` ou `Tp1` avec la commande `"dir"` (Windows) ou `"ls"` (Linux).
   — Tapez `"./Tp1"`.
6. *Compression :*
   En cliquant à droite sur le répertoire `Tp1`, fabriquer une archive comprimée `Tp1.zip` (ou `Tp1.7z` suivant la machine). 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 :

1. La plupart des autres environnements, comme Visual Studio (Windows) et Xcode (Mac) ne connaissent pas `Cmake`, et on doit utiliser nous-mêmes `Cmake` pour convertir en un projet natif de l'environnement.
2. Si vous ne la voyez pas, vous pouvez la faire apparaître avec la petite flèche à droite de ces onglets
3. Sur certaines machines, il faut en fait aller sur C :, vérifiez en regardant où est installé votre projet

— 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.

Le dossier précieux représentant votre travail est le dossier *source*, `Tp1_Initial`. Celui jetable est le dossier *build*, `build-Tp1_Initial-....`

7. Vérifiez qu'on peut tout recommencer : quittez `QtCreator`, effacez vos dossiers source et build, extrayez le source de votre archive ; s'il reste un fichier `CMakeLists.txt.user`, créé par `QtCreator`, supprimez-le avant de lancer `QtCreator` et d'ouvrir le projet.

1.2 Premières erreurs

Vous en ferez beaucoup, et c'est tout à fait normal. Autant s'y habituer dès maintenant.

1. *Modifier le programme :*
   Modifier le programme en changeant par exemple la phrase affichée.
   (a) Tester une nouvelle génération/exécution. Vérifier que `QtCreator` sauve le fichier automatiquement avant de générer.

   Lancer directement une exécution sauve et génère automatiquement.

   Du coup, l'outil marteau peut sembler inutile. Néanmoins, il est bon lors du codage de vérifier simplement la compilation très fréquemment, sans se préoccuper de l'exécution car il n'y a encore rien à observer.

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 :

   Double-cliquer sur un message d'erreur (Onglet 1 "Issues") positionne l'éditeur sur l'erreur.

   Notez que l'onglet 1 n'est qu'un extrait automatique et cliquable de l'onglet 4, et que le message complet est dans ce dernier. Parfois l'extraction automatique a des ratés et le message d'erreur de l'onglet 1 n'est pas clair car incomplet.

3. *Erreur de linker*
   Il est un peu tôt pour réussir à mettre le linker en erreur, mais ça arrivera quand on organisera notre code en plusieurs fichiers. 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 générer. C'est pour l'instant une erreur de *compilation* (il ne sait pas ce que f désigne).
   (b) Corriger l'erreur de compilation en rajoutant une ligne (pour l'instant *"magique"*)
   void f(int i); avant la ligne avec main. Générer le programme : le *linker* constate l'absence d'une fonction `f()` utilisée par la fonction main() qu'il ne trouve nulle part.
   (c) Effacez maintenant l'appel à f dans main(). Tout remarche à nouveau. Renommez main en `main` en `main2`. L'absence de fonction `main` sera remarquée par le linker, qui refusera de générer le programme.

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+I = indenter la zone sélectionnée. Ctrl+A,Ctrl+I = indenter tout le fichier.

5. *Warnings du compilateur*
   En modifiant le main(), provoquer les warnings suivants : [4]
   (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) Ajouter `exit;` comme première instruction de `main`. Appeler une fonction en oubliant les arguments arrive souvent ! Exécuter pour voir. Corriger en mettant exit (0);. (La fonction exit () quitte le programme en urgence !)

   Certains warnings sont anodins (faux positifs) et ne sont pas forcément des erreurs. Le problème est que ne pas les corriger peut noyer les vrais positifs, il faut donc s'efforcer de *tous* les corriger.

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
        k=2;
    return 0;
}
```

2. Pour pouvoir utiliser le debuggeur avec `QtCreator`, il faut compiler en mode Debug. Le choix du mode se fait en cliquant sur le petit terminal  au-dessus du triangle vert d'exécution. Observez que la variable `CMAKE_BUILD_TYPE` vaut alors `Debug` dans le mode "Projects".
3. Lancer le programme "sous le debuggeur" avec le triangle vert avec le bug. Le programme s'exécute mais rien ne semble se passer.
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`. 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`.

4. Aller dans le mode "Projects" de `QtCreator`  , rubrique CMake, clicker sur le bouton "Advanced" et ajouter "-Wall -Wextra" à la variable `CMAKE_CXX_FLAGS`.

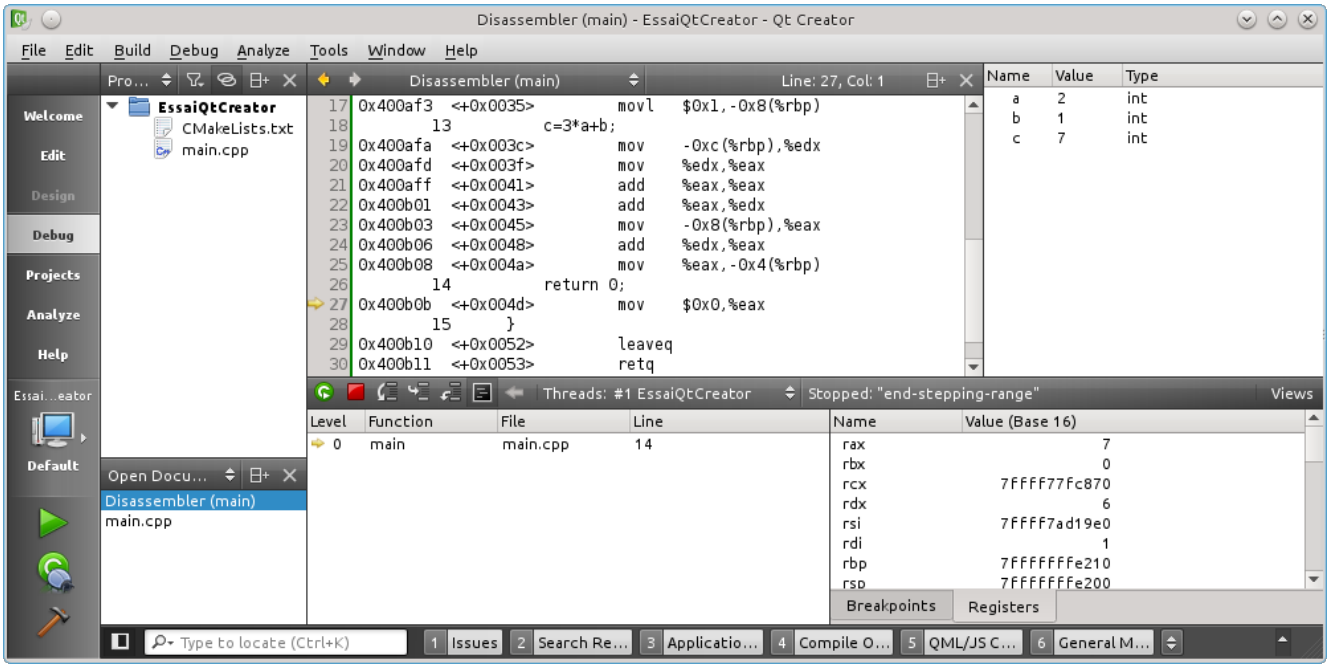

FIGURE 1 – `QtCreator` montrant le code machine et les registres

9. Enfin, pour voir à quoi ressemble du code machine, exécuter jusqu'à un point d'arrêt puis activer le menu `Debug/Operate by Instruction`. 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 au programmeur chevronné pour vérifier ce que fait le compilateur et voir s'il optimise bien.
10. 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`.