UNIVERSITÉ DE BORDEAUX

Master Bioinformatique

Génie Logiciel 2017/2018

Travaux Pratiques

TP Makefile

1 Gestion des animaux

Reprenez le code permettant la gestion d'animaux.

1. Identifiez l'exécutable et ses dépendances.
2. En vous basant sur l'exemple vu en cours, écrivez un Makefile permettant de compiler le programme.

2 Manipulation de bibliothèques

L'objectif de cet exercice est la compilation d'une interface graphique simple affichant une courbe 2D (cf. Fig 1). Cette interface graphique se décompose en deux éléments principaux :

1. une zone permettant d'afficher la courbe. La courbe est générée dynamiquement à partir du code, il ne s'agit pas d'une image statique.
2. Un bouton permettant de quitter le programme.

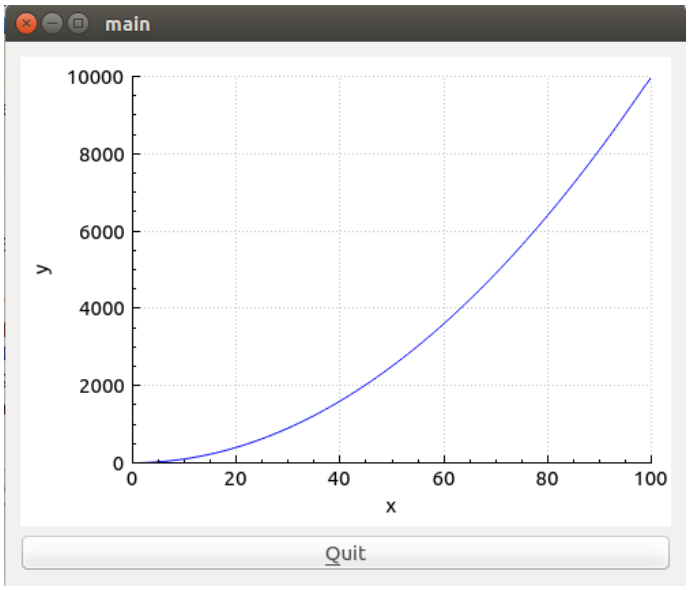

FIGURE 1 – Interface graphique simple de rendu de courbes utilisant Qt.

Afin d'effectuer cet affichage, nous allons utiliser la bibliothèque `Qt`. Les bibliothèques (*libraries* en anglais) permettent d'utiliser des fonctionnalités existantes comme, par exemple, la création d'interface graphique, la manipulation de bases de données, ou l'utilisation d'outils statistiques. Elles regroupent plusieurs fichiers compilés (`.o`) dont les fonctions peuvent être utilisées au sein de notre programme. On distingue deux types de bibliothèques en C/C++ :

1. les bibliothèques **statiques** : il s'agit de bibliothèques destinées à être utilisées en lien avec un seul programme à la fois. Elles sont intégrées physiquement, à la compilation, au code de l'exécutable. L'extension de ces bibliothèques est généralement `.a`.
2. les bibliothèques **partagées** : ces bibliothèques peuvent être utilisées par plusieurs programmes à la fois. Ces bibliothèques sont associées au programme au moment où elles sont exécutées, et ne font donc pas partie intégrante de l'exécutable. L'extension de ces bibliothèques est généralement `.so`.

L'utilisation des bibliothèques dans un code en C/C++ se fait selon deux étapes :

1. **inclusion** : dans le fichier C/C++, on inclut, avec la directive `include`, les en-têtes où les fonctions sont définies. À la compilation, on renseigne la localisation des fichiers d'en-tête (`.h`) de la bibliothèque par l'option `-I`.
2. **édition de liens** : il faut ensuite renseigner où chercher la bibliothèque (`.so`) avec l'option `-L` (inutile si la bibliothèque a été installée en tant que root) et donner le nom de la bibliothèque avec l'option `-l` (obligatoire).

Attention ! `Qt` est une bibliothèque écrite en C++ et non en C, les fichiers suivants sont donc écrits en C++ (d'où l'extension `.cpp`). Dans cet exercice, nous nous concentrons sur l'écriture d'un Makefile, seul le compilateur diffère : on utilisera g++ au lieu de gcc.

Vous pouvez récupérer le code à l'adresse suivante : `http://www.labri.fr/perso/fgrelard/teaching.html`. Étudiez le code avant de poursuivre.

2.1 Compilation de l'interface graphique simple

L'objectif est ici d'écrire un Makefile pour compiler le fichier `gui_simple.cpp`.

Questions

1. La bibliothèque `Qt` a-t-elle été installée en tant que root sur votre machine ? Comment le vérifier ?
2. Lire attentivement le code présent dans `gui_simple.cpp` : qu'est-il supposé afficher ?
3. Sachant que l'on utilise les bibliothèques `QtGui` et `QtCore`, écrivez un Makefile qui compile `gui_simple.cpp` et crée un exécutable nommé `GUISimple` en n'oubliant pas d'utiliser les options `-I` et `-l`. Vous pouvez vous aider de la documentation de g++ au besoin.

2.2 Rendu d'une courbe 2D dans la fenêtre

Nous allons utiliser une bibliothèque externe, nommée `qcustomplot`, basée sur `Qt`, et qui permet de tracer une courbe dans notre interface graphique. Nous allons compléter le Makefile précédent pour compiler le fichier `carre.cpp`.

Questions

1. Il est tout d'abord nécessaire de compiler la bibliothèque `qcustomplot`. Déplacez-vous dans le répertoire `qcustomplot/sharedlib/compilation`. Un Makefile est déjà présent. Compilez la bibliothèque. La bibliothèque générée est-elle statique ou dynamique ?
2. Écrivez un Makefile qui compile `carre.cpp` et crée un exécutable nommé `CourbeCarre` en n'oubliant pas d'utiliser les options `-I, -L, -l`. Vous pouvez vous aider de la documentation de g++ au besoin.
3. Exécutez le programme. Que se passe-t-il ? Pourquoi ?
4. La variable d'environnement `LD_LIBRARY_PATH` permet de renseigner le chemin vers les bibliothèques utilisées à l'exécution des programmes. Dans le terminal, assignez la valeur qui convient à cette variable.
5. Exécutez à nouveau le programme : la fenêtre avec la courbe devrait s'afficher (cf. Fig 1).

2.3 CMake

CMake est un outil permettant la génération de Makefile. Il est particulièrement utile dans le cadre de projets conséquents avec de nombreux exécutables et dépendances. Pour les bibliothèques installlées en tant que root, CMake résout les dépendances automatiquement, il est alors inutile de spécifier les chemins manuellement.

CMake est basé sur un fichier nommé, par convention, `CMakeLists.txt`. Ce fichier renferme les instructions pour générer automatiquement le Makefile et spécifie les dépendances majeures. Il permet de penser uniquement aux dépendances et aux options, sans se préoccuper des instructions de compilation bas niveau.

Pour générer le Makefile à partir de `CMakeLists.txt`, on utilise l'utilitaire `cmake` ou `ccmake` (interface en ligne de commandes).

Questions

1. Lisez le fichier `CMakeLists.txt` et décommentez les lignes commentées en spécifiant les chemins. Quelle est la différence lors de l'édition de liens ?
2. Créez un répertoire `build` à la racine du répertoire `sujet`, qui contiendra les exécutables issus de CMake.

3. À l'intérieur de `build`, utilisez la commande
   `cmake ..`
   ou `ccmake ..`
   pour générer un Makefile. Lisez la documentation liée à ces deux commandes au besoin.
4. Lisez le Makefile généré et comparez avec celui que vous avez écrit dans la partie 2.2.