ÉCOLE POLYTECH'NICE SOPHIA
SI4
2021-2022

Julien Deantoni

Programmation multi paradigm en C++

TD numéro 1

Premiers Pas

Objectif

L'objectif de ce TD est de prendre en main l'environnement minimal de développement en C++. Votre environnement peut se composer d'un éditeur de texte et d'une console ou passer par un environnement spécialisé (IDE). Dans tous les cas, malgré la possibilité de faire générer un Makefile automatiquement, il vous est demandé de faire votre propre Makefile. Ainsi, si vous utilisé un IDE, vous devrez le paramétrer afin qu'il utilise un Makefile "maison". D'une manière plus générale, si je décide de noter un TD, vous devez pouvoir me l'envoyer dans les 2 minutes et je dois pouvoir le compiler sous linux sans avoir à installer un IDE quelconque.
L'exercice proposé est donc simple et ne représente pas de réelles difficultées algorithmiques. **Prenez le temps de faire les choses proprement** et posez vous des questions afin de construire une base saine pour la suite.

Rappel sous linux

Quelques petits rappels pour l'installation de nouveaux packages.
sous debian ou ubuntu:

- L'installation de package : `sudo apt-get install nom_package`
- La recherche de package : `apt-cache search mot clef`

sous fedora:

- L'installation de package : `sudo yum install nom_package`
- La recherche de package : `yum search mot clef`

sous opensuse:

- L'installation de package : `sudo zypper install nom_package`
- La recherche de package : `zypper search mot clef`

Installer au minimum g++, make et libc6-dev (aussi nommé glibc-devel)...
Il existe de nombreux IDE. Dans un premier temps un éditeur de texte (*e.g.*, *Kate*) me parait plus adapté. Cependant, dans un deuxième temps il est recommandé d'utiliser un IDE tel que eclipse CDT ou CLion, qui semblent les plus aboutis aujourd'hui (mais je ne passe pas mon temps à tester). Ce qui est important si vous utilisez un IDE c'est de ne pas le laisser maîtriser votre programme. Donc si vous voulez ajouter

une librairie particulière ou autre, vous devez savoir le faire, en passant par le Makefile ou par l'IDE. Faites aussi attention à ce que l'IDE fait seul (complétion, squelette de classe, etc). Demandez vous bien si les choix par défaut qu'il fait sont les bons.
NB: lors des TPs notés je compilerai sous linux avec g++. De plus je ne démarrerai aucun IDE et ne serai disposé qu'à taper des commandes de type make, cmake ou équivalent. Si vous voulez vraiment être sous windows, débrouillez vous, je ne démarrerai pas sous windows pour tester votre programme :)
NB2: à tout moment votre programme doit compiler. N'attendez pas d'avoir écrit 50 lignes avant de compiler. À tout moment je suis susceptible de vous dire d'envoyer votre code par mail (vous aurez alors moins de 10 minutes pour le faire). Le code devra compiler pour espérer une note acceptable.

Énoncé du problème

Dans la partie *enseignement*, *Programmation Multi Paradigmes en C++*, *ressources* de ma page web[1], vous trouverez deux Makefiles relativement simples ainsi qu'un fichier de configuration cmake. La v2 n'est pas très à jour concernant la manière d'écrire des Makefiles mais il a l'avantage de tout expliciter. Si vous n'êtes pas très à l'aise avec *make*, Je vous propose de le réutiliser pour vos programmes. La v1 est plus simple à utiliser pour les projets sans template mais devra être modifiée lors du passage aux templates. Le fichier de configuration de cmake est relativement simple à utiliser. Commencez par les ouvrir et lire les commentaires. Vous en choisirez un et le modifierez ensuite en fonction des fichiers créés pour le problème suivant.

1. Palindrome

Un palindrome est, comme chacun sait, une chaîne de caractères qui se lit de la même manière dans les deux sens, comme "otto" ou "madam".
Écrivez un programme qui lit des chaînes de caractères sur l'entrée standard (jusqu'à un retour chariot), vérifie s'il s'agit d'un palindrome et affiche le résultat sur la sortie standard.
Commencez par faire un `main()` simple qui ne fait presque rien comme ceux vus en cours. Compilez le. Testez le. Ensuite, vous le compléterez pour qu'il fasse ce qui est demandé.
remarque : Afin de lire une chaîne de caractère sur l'entrée standard, utilisez l'objet `std::cin` pré-existant. Il s'utilise avec la même philosophie que `std::cout` :

```
int anInteger;
std::cout << "please, give an integer: ";
std::cin >> anInteger;
```

Afin de connaître les API des classes que vous pourrez éventuellement être amené à utiliser (`string` par exemple), vous pouvez regardez ici : `http://www.cplusplus.com/reference/`

2. Automate Cellulaire

Un automate cellulaire consiste en une grille régulière de "cellules" contenant chacune un "état" choisi parmi un ensemble fini et qui peut évoluer au cours du temps. L'état d'une cellule au temps t+1 est fonction de l'état au temps t d'un nombre fini de cellules appelé son "voisinage". À chaque nouvelle unité de temps, les mêmes règles sont appliquées simultanément à toutes les cellules de la grille, produisant une nouvelle "génération" de cellules dépendant entièrement de la génération précédente.[2]

en 1 dimension

- Créez un *container* de Booléen.
- Initialisez le (*e.g.,* de manière aléatoire).

[1] `http://www.i3s.unice.fr/~deantoni/`
[2] `https://fr.wikipedia.org/wiki/Automate_cellulaire`

- À chaque évolution, la valeur booléenne d'une cellule en position 'x' doit être égale à la valeur en 'x-1' *ET* la valeur en 'x+1'.
- Itérez et affichez le contenu du container à chaque itération.

en 2 dimension: le jeu de la vie

Faites une version en 2 dimensions d'un automate cellulaire Booléen (initialisé comme vous le souhaitez) avec les règles suivantes (tirées également de wikipedia[3]): *À chaque étape, l'évolution d'une cellule est entièrement déterminée par l'état de ses huit voisines de la façon suivante :*

- *Une cellule morte possédant exactement trois voisines vivantes devient vivante (elle naît).*
- *Une cellule vivante possédant deux ou trois voisines vivantes le reste, sinon elle meurt.*

Réfléchissez à la complexité de votre programme. Testez les différentes optimisations de G++ et appréciez le résultat en terme de temps de calculs et/ou de taille de l'exécutable. Apprenez à utiliser un *debugger* comme *ddd* par exemple (malgré son air *old school* il est vraiment très intéressant).

Remarque générale

Comme on a du vous le dire dans différentes matières, vous testerez vos programmes. N'attendez surtout pas d'avoir tout fait pour lancer la première compilation et/ou le premier test. Travaillez de manière incrémentale : dès qu'une fonction est réalisée, écrivez un ou plusieurs test pour s'assurer que celle-ci fonctionne. Premier l'habitude de garder ces tests dans votre programme (mettez les en commentaire si vous ne voulez pas surcharger la sortie standard mais ne les effacez pas).
Cette remarque est particulièrement valable si je décide de ramasser un des TDs dans le but de le noter...

[3] `https://fr.wikipedia.org/wiki/Jeu_de_la_vie`