POLYTECH LILLE

Université Lille1 Sciences et Technologies

Département IMA / 3A (S5) **Programmation Structurée 2012/2013**

http://laure.gonnord.org/pro/teaching/

TP7 - Tris de Tableaux + compilation séparée

Objectifs

Coder et comparer les différents tris vus en cours. Début de la compilation séparée, première utilisation d'un Makefile.

1 Caractères et code ASCII - 10/15 min

Finir l'exercice 2.2 de la feuille de TP numéro 6 (impression des lettres qui ne sont pas dans le fichier). On pourra travailler dans le fichier `exo_tp6.c` disponible sur la page web du cours (après l'avoir sauvegardé dans un endroit adéquat, bien sûr !).

2 Unix - Wget / Tar

1. Ouvrez un Terminal et placez-vous dans *votre répertoire de travail pour la programmation structurée*, répertoire TP7.
2. Récupérez sur le compte de Laure Gonnord, à polytech, l'archive `tp7.tgz` situé dans `~lgonnord/IMA3/`.
3. Désarchivez avec la commande :

   ```
   tar xvfz tp7.tgz
   ```

   (x pour extraire, v pour verbose, f pour file, z pour décompression). La création du répertoire `TP7` est normalement automatique.

3 Préparation : compilation de plusieurs fichiers

Au TP 4, nous avons créé un fichier contenant des fonctions de base sur les vecteurs d'entiers (initialisation avec des valeurs aléatoires, impression sur le terminal, ...).

L'archive fournit une correction des fonctions les plus importantes : le fichier `vecteurs.c` contient ces fonctions, et le fichier `vecteurs.h` fournit les déclarations de ces fonctions. Nous allons utiliser ses fonctions dans un troisième fichier (`tris.c`). Pour compiler tout cela, nous allons utiliser la méthode de *compilation séparée*.

1. Observer les fichiers `vecteurs.c` et `vecteurs.h`. Remarquer que le .c ne contient pas de fonction principale (*main*), et que le .h ne contient pas de code des fonctions, juste leurs signatures (entêtes).
2. Compiler `vecteurs.c` avec la commande :

   ```
   clang -Wall -c vecteurs.c
   ```

Cette commande crée un fichier `vecteurs.o` (vérifier), qui est un fichier objet (voir la feuille annexe sur la compilation)

3. Créer et éditer le fichier `tris.c` (avec emacs), inclure le fichier d'entête (`#include "vecteurs.h"`), puis dans le programme principal(*main*) faites des appels aux fonctions d'initialisation et d'impression.
4. Compiler avec :

   ```
   clang -Wall -c tris.c
   ```

   Et enfin, si la commande précédente s'est passée sans problème (avec génération de `tris.o`), compiler le binaire final avec :

   ```
   clang -Wall -o tris vecteurs.o tris.o
   ```

5. Exécuter. Tester.
6. Modifier le fichier `tris.c` en ajoutant un `printf` par exemple. Quelle(s) commande(s) de compilation est-il suffisant de lancer ? (ne pas exécuter cette(ces) commande(s)).

   Pour automatiser le processus de compilation, on peut utiliser un fichier nommé `Makefile` (Voir http://fr.wikipedia.org/wiki/Make). Ce fichier comprend des *règles de compilation* qui ne sont exécutées que si elles sont strictement nécessaires ; par exemple, si `vecteurs.c` n'est pas modifié, le fichier objet `vecteurs.o` n'est pas re-généré. **Au S5, nous ne ferons qu'utiliser des Makefile simples, mais au S6 nous apprendrons à les fabriquer.**

7. Compiler avec le Makefile fourni : taper `make` et observer.
8. À ce stade, faites une archive compressée avec les fichiers .c et .h, Makefile et pas le reste :

   ```
   tar cvzf tp7_amoi.tgz Makefile vecteurs.h vecteurs.c tris.c
   ```

   puis vérifiez (dans un autre répertoire), que vous savez le décompresser.

Remarque 1 *Attention lors du désarchivage d'un tgz (ou tar), si l'archive contient un fichier de même nom qu'un fichier du répertoire courant, le fichier du répertoire courant est écrasé. On fera donc attention à désarchiver dans un répertoire* vide. *Il est déjà arrivé qu'un projet soit complétement PERDU à cause d'une telle manipulation malencontreuse.*

4 Codage de tris

Dans le fichier `tris.c`, écrire et tester les tris du cours suivants (utiliser `make` pour compiler) :

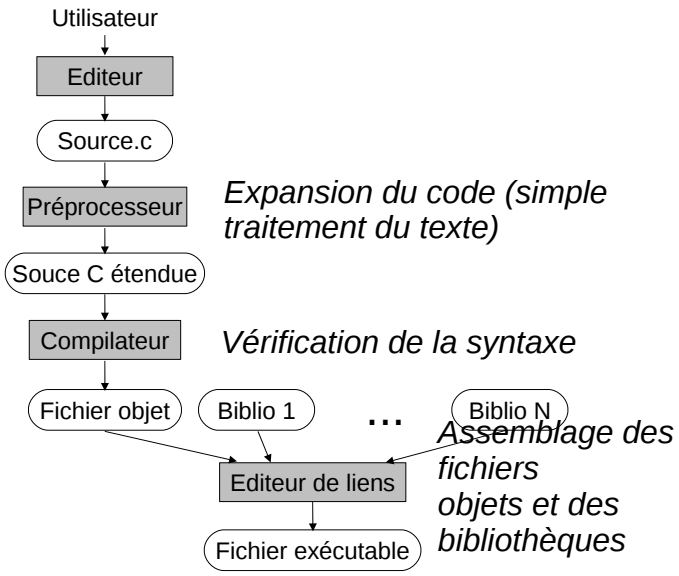
1. Tri insertion.
2. Tri sélection.
3. Tri fusion.

On écrira une procédure par tri, et on testera dans le main, sur des tableaux de taille 10 initialisés au hasard. Pour aller plus loin, on pourra faire varier la taille et mesurer les temps d'exécution avec la commande Unix `time`.

Compilation Pratique en TP (sera revu au S6)

1 Schéma récapitulatif

(Source : Cours de GIS3 par FS)



2 Step by step

Préprocessing Les fichiers d'entête (`#include <stdio.h>`) sont étendus, ainsi que les macros (`#define`). On peut éventuellement donner des valeurs aux macros pendant cette phase. Avec clang on peut réaliser uniquement cette étape avec :

```
clang -E compil.c -o compil.preprocessed.c
```

(compil.c est le `code source`, et la commande produit un code C sans macro). Il faut pouvoir trouver les fichiers `.h` quelque part. L'option`-I` indique le chemin de recherche pour trouver les fichiers inclus dans les macros `include <fichier.h>`. Un certain nombre de répertoires sont prédéfinis et n'ont pas besoin d'être spécifiés, par exemple sous ma Debian /usr/local/include et le répertoire de compilation. L'ordre des répertoires de recherche est important. Une variante existe : les macros `#include "fichier.h"` se limitent au répertoire courant.

Compilation et assemblage Le gros du travail : transformation du fichier texte précédent en langage d'assemblage spécifique à la machine, puis en fichier objet (.o) Les deux phases : préprocessing/compilation-assemblage, peuvent se réaliser à l'aide de la commande :

```
clang -c compil.c
```

et il y a création d'un fichier `.o` (objet) Pour effectuer cette phase, il faut que le compilateur connaisse tous les chemins nécessaires pour trouver les fichiers d'entête (.h), à l'aide de l'option `-I`.

Édition de lien Cette étape prend un ensemble de fichiers objets pour produire un programme exécutable. Dans cette étape, les fonctions de librairie (printf, etc.) sont "ajoutées" à l'exécutable.

```
clang -o ptitnom fichier1.o fichier2.o
```

l'option `-o` sert à donner le nom que l'on veut à l'exécutable, ce nom étant par défaut `a.out`.

Tout en même temps On peut effectuer toutes ces opérations en même temps, par exemple sur un unique fichier :

```
clang -o nombinaire nomfichier.c
```

ou sur plusieurs :

```
clang -o tris vecteurs.c tris.c
```

La compilation séparée Quelquefois, il peut être utile de compiler uniquement un fichier à la fois et pas tous, par exemple dans le tp de tris, vecteurs.c n'a pas besoin d'être recompilé à chaque fois, donc on peut utiliser :

```
clang -o vecteurs.c
clang -o tris.c
clang -o tris vecteurs.o tris.o
```

et à chaque modification de `tris.c`, refaire uniquement les deux dernières lignes.
Ou alors, ne recompiler que ce qui est nécessaire de recompiler, c'est ce genre de choses qui seront permises par les Makefile (vus plus tard).