

Travaux Dirigés Programmation C avancée

Informatique 1ère année.

—Julien Allali - allali@enseirb-matmeca.fr —

1 Convention de code

Introduction :

Une “convention de codage” est un document qui décrit un certain nombre de règles quand à l’écriture de programme. Ce document fixe par exemple la langue (français/anglais), le nommage des structures (majuscule, minuscule, utilisation de l’underscore), des macros, des fonctions, des variables locales et globales. Egalement, on y trouve des règles générales : type de retour de fonction, gestion des erreurs, position de l’accolade.

►Exercice 1. Mise en place d’un répertoire de projet :

Créer un répertoire nommé *pg106*, ce sera la racine de votre projet. Dans ce répertoire créer un fichier *REAME.md*.

Le fichier *REAME.md* sera rédigé au format *Markdown* (chercher sur internet). Dans ce fichier écrire une section *coding convention* dans laquelle vous décrierez l’ensemble des règles d’écriture de code pour votre projet. Vous essayerez d’être exhaustif et de donner des exemples illustratifs.

2 Remaniement de source

►Exercice 2. Récupérer l’archive *hash.c.gz* en ligne.

Décompresser le fichier à l’aide du programme *gunzip*. Ce source est une implémentation des tables de hachage. Le principe est de stocker des éléments dans la table de manière efficace de façon à ce que la recherche d’éléments se fasse rapidement.

Pour cela, une table de hachage est composée de N listes chaînées. On ne connaît pas la nature exacte des éléments de la table mais on dispose d’une fonction qui renvoie un entier associé à un élément. Cet entier détermine la liste dans laquelle l’élément doit être inséré. On dispose également d’une fonction qui permet la comparaison de deux éléments. Ainsi, les listes sont chaînées dans l’ordre croissant des éléments.

- Quelle structure encode la liste chaînée ?
- Quelle est le type de liste chaînée ?
- À quoi correspondent les paramètres de *hash_init* ?

Dans votre répertoire de projet, créer un répertoire *thirdparty* et dans ce répertoire le répertoire *hash* qui contiendra le fichier source.

Nous allons maintenant, étape par étape, mettre au propre ce code source.

►Exercice 3. Homogénéisation du nommage des fonctions.

Commencer par renommer les fonctions en respectant votre convention de codage. Ajouter des commentaires devant chaque fonction avec une description en une ligne de ce que fait la fonction.

►Exercice 4. Découpage.

Le source contient une implémentation de table de hachage et un programme d’exemple. Séparer les deux.

►Exercice 5. Le header

Pour pouvoir garantir que le programme et l’implémentation des tables parlent de la même façon, il faut écrire un fichier d’entête.

Créer l'entête. Ajouter dedans les fonctions, structures etc... nécessaires.

Au niveau de la macro `HASH.SIZE`, celle-ci doit-elle aller dans le header ou non ? Faire en sorte que sa valeur puisse être redéfinie au moment de la compilation.

3 Compilation

► **Exercice 6.** *La compilation :*

Compiler les deux fichiers sources en objets puis lier les deux en un programme exécutable. Vérifier que le programme fonctionne.

► **Exercice 7.** *Les asserts :*

Ecrivez un deuxième programme d'exemple dans lequel vous appellerez la fonction d'ajout en spécifiant `NULL` comme premier argument. Compiler et exécuter : que se passe-t-il ?

Compiler à nouveau en ajoutant l'option `-DNDEBUG`. Exécuter : que ce passe-t-il ?

Consulter la documentation de la fonction `assert`.

► **Exercice 8.** *Bibliothèque :*

Compiler le module `hash` sous la forme d'une bibliothèque static `libhash.a` et une bibliothèque dynamique `libhash.so`.

Compiler le programme d'exemple en utilisant l'une et l'autre de ces bibliothèques (cela produit donc 2 executables différents). À l'aide des programmes `nm` et `objdump`, lister les symboles présents dans vos executables. Utiliser le programme `ldd` pour vérifier les dépendances.

Modifier la fonction d'init pour qu'elle affiche un message sur la sortie standard. Recompiler les deux bibliothèques puis tester les executables précédents sans les recompiler.

► **Exercice 9.** *Script shell :*

Ecrire un script shell qui automatise la compilation de vos sources. Votre script prendra en paramètre "dynamic" ou "static" pour indiquer quel mode de compilation est souhaité.

Sachant que dans un script shell la ligne :

```
DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"
```

permet de stocker dans la variable `DIR` le nom de répertoire contenant votre script shell, ajouter la fonctionnalité d'outsourcing à votre script. Ceci signifie que tous les fichiers générés par la compilation (les `.o`, bibliothèques et executables) doivent être placés dans le répertoire courant et non avec les sources (sauf si le répertoire courant est le répertoire des sources bien sûr).