INFO0902 - Structures de données et algorithmes

Projet 0: prise en main (facultatif)

Pierre Geurts — Jean-Michel Begon — Romain Mormont

10 février 2017

Enoncé

Le but de ce projet facultatif est double. D'une part, il s'agit de se familiariser avec la routine de soumission des projets cotés, et d'autre part, d'apprendre à éviter les erreurs les plus courantes en débuggant un court programme.

Le projet est constitué des 5 étapes suivantes :

(a) Créer un compte sur la plateforme de soumission.

(b) Créer un compte sur les machines ms8xx.

(c) Compiler le code fourni.

(d) Debugger le code à l'aide de `valgrind`.

(e) Soumettre la correction.

Plateforme de soumission

Dans le cadre de ce cours, nous utiliserons la plateforme de Montefiore : http://submit.montefiore.ulg.ac.be.

Si vous ne disposez pas d'un compte, vous devez en souscrire un via le formulaire. Une fois votre compte validé, vous pourrez sélectionner le cours et le projet afin de soumettre votre solution.

C'est également via cette plateforme que les cotes des projets seront communiquées.

Créer un compte sur les machines ms8xx

Les machines du réseau ms8xx seront utilisées pour compiler et tester les projets. Il est donc primordial que votre code y fonctionne ! Prenez l'habitude d'y tester votre code avant de le soumettre afin d'éviter les mauvaises surprises.

Les modalités pratiques sont reprises à la page www.montefiore.ulg.ac.be/ jmbegon/?sda2016_2017#testingmachines.

Ces machines disposent également de l'outil `valgrind` qui sera utile pour la suite.

Compiler le code fourni

Avec cet énoncé est fournie une archive `p0.tar.gz` contenant les fichiers :

`Median.h` Il s'agit d'un *header* définissant la fonction `median` qui génère un tableau aléatoire, le trie et renvoie l'élément médian.

`Median.c` Il s'agit d'un fichier `c` implémentant la fonction `median`, ainsi que plusieurs fonctions auxiliaires [1].

`main.c` Il s'agit du *main* qui fait appel à la fonction `median`. Il imprime en console le résultat.

Après décompression :

```
tar xzvf p0.tar.gz
```

les fichiers peuvent être compilés avec la commande :

```
gcc main.c Median.c --std=c99 --pedantic -Wall -Wextra -Wmissing-prototypes -o prog
```

Plusieurs *warnings* devraient s'afficher. Le programme peut être lancé via :

```
./prog
```

ce qui devrait engendrer une erreur de segmentation (`Erreur de segmentation (core dumped)`).

Débugger à l'aide de `valgrind`

Une erreur de segmentation est le fruit d'un accès à une zone mémoire interdite. Le système d'exploitation fournit peu d'informations permettant de trouver l'origine de cette erreur. `Valgrind` est un programme qui permet d'en lancer un autre dans un environnement contrôlé afin de détecter plusieurs types d'erreur. Si sa vocation première est la détection des fuites de mémoire, `valgrind` est également utile en cas d'erreur de segmentation.

Pour tirer meilleur parti des rapports de `valgrind`, il vaut mieux compiler le code en mode debuggage (en ajoutant l'option `-g`) :

```
gcc main.c Median.c --std=c99 --pedantic -Wall -Wextra -Wmissing-prototypes -g -o prog
```

Pour afficher le rapport, il faut donner la commande de lancement du programme à `valgrind` :

```
valgrind ./prog
```

Le but est donc de décortiquer les différents éléments du rapport afin de trouver les causes des erreurs qui se sont glissées dans le code et de les corriger.

Le code n'étant pas particulièrement propre, nous vous encourageons à l'améliorer tout en prenant connaissance des critères de cotation utilisés dans le cadre de ce cours (disponibles à l'adresse www.montefiore.ulg.ac.be/ jmbegon/?mark_criteria).

Soumission

Une fois le fichier `Median.c` corrigé, vous devez le renommer en `Median2.c` et l'archiver (seul) au format `tar.gz`. Par exemple, via la ligne de commande :

```
tar czvf soumission.tar.gz Median2.c
```

Contrairement au nom du fichier source, le nom de l'archive (ici "`soumission`") n'a pas d'importance. Seul le format compte. Vous pouvez ensuite soumettre l'archive sur la plateforme *cicada*. Notez qu'il est possible de soumettre plusieurs fois votre archive, seule la dernière soumission étant prise en compte.

Deadline

Une deadline est associée à la remise de chaque projet. Passé cette limite, la plateforme n'acceptera plus aucune soumission. Pour ce projet facultatif, la deadline est fixée à **jeudi 23 février, 23h59**.

Bon travail !

1. Quelques informations utiles relatives aux algorithmes utilisés sont fournies dans les annexes.

Annexes

La génération aléatoire et le concept de *seed*

Un ordinateur étant par essence une machine déterministe, il ne lui est pas possible de générer des nombres entièrement aléatoires. Il existe toutefois des algorithmes capables de générer des séquences de valeurs d'apparence aléatoire, utilisés pour créer des *Random Number Generators* (ou RNG). Ceux-ci sont généralement initialisés à l'aide d'une *seed* (ou graine, en français), une valeur issue d'un intervalle très large.

Dans le cadre de ce projet, nous vous fournissons dans une fonction simple pour générer des entiers signés issus d'un intervalle donné. Une *seed* donnée sous forme d'entier non signé est toutefois nécessaire.

Le tri à bulles (*Bubble sort*)

Le tri d'un vecteur d'éléments en un vecteur ordonné est un problème fondamental en informatique, qui sera par ailleurs étudié en profondeur lors des cours *ex cathedra*. Pour ce projet, le programme qui vous est fourni utilise un algorithme de tri simple : le tri à bulles (ou *Bubble sort*).

Le tri à bulles consiste à parcourir le tableau et, pour chaque paire d'éléments voisins, à les permuter quand la valeur du premier est plus grande que celle du second. Cela a pour conséquence de faire *remonter* les éléments ayant une grande valeur à la fin du vecteur, telles des bulles remontant à la surface de l'eau. Toutefois, pour obtenir un vecteur entièrement trié, il est nécessaire de faire plusieurs parcours du vecteur jusqu'à ce qu'il n'y ait plus aucune permutation.

`Median.c` implémente une variante de cet algorithme, qui consiste à parcourir le vecteur à l'envers pour faire *descendre* la plus petite valeur et recommencer sur le même vecteur en omettant la première valeur (puisque c'est la plus petite), et ainsi de suite, de sorte que la taille du vecteur à trier diminue à chaque parcours.