TP6 : Organisation de la mémoire ; E/S en C

CSH : Initiation au C et au shell

Première année

[TP6]

★ Exercice 1. Organisation mémoire d'un processus

On souhaite explorer la disposition mémoire d'un processus C, c'est à dire les adresses mémoire où sont placés les différents éléments possible. Pour afficher l'addresse à laquelle se trouve une variable `i` quelconque, il faut utiliser : `printf("variable i : %p",&i) ;`

▷ **Question 1.**

- Dans un fichier appelé exo1.c, déclarer les variables suivantes :
  - chaîne de 10 caractères globale et statique ;
  - tableau de 10 entiers global ;
  - flottant local au main ;
  - caractère local statique ;
  - entier nommé ex déclaré en externe.

  Dans la fonction main(), faire écrire les adresses des variables.
- Dans un deuxième fichier appelé exo1bis.c mettre la définition de l'entier ex que vous initialiserez à la valeur 20.

▷ **Question 2.** Compilez ce programme avec `gcc -c exo1.c` `gcc -c exo1bis.c` puis réalisez l'édition de liens avec `gcc -o exo1 exo1.o exo1bis.o`. Exécutez `exo1` et concluez sur la place des différentes variables. Quelle est l'effet du modificateur `static` ? Qu'en est-il de `extern` ?

▷ **Question 3.** Modifiez votre programme pour qu'il affiche l'adresse de la fonction main() elle-même. Obtenir l'adresse d'une fonction est similaire à ce qu'on fait pour une variable : `printf("%p\n",&main) ;` Où est placé le code de cette fonction ?

▷ **Question 4.** Ajoutez une variable de type `char*` dont le contenu est l'adresse d'un bloc alloué dynamiquement avec l'appel `malloc(512) ;`. Où est placé cette variable ? Où est placé le bloc alloué ?

★ Exercice 2. Mesurer la complexité d'un fichier C

On souhaite avoir une mesure de la complexité de divers programmes C. Pour cela, plusieurs métriques sont utilisables, que nous allons explorer au fil des questions.

La première idée est de mesurer la longueur du texte. Il est assez courant d'annoncer le nombre de lignes composant un programme donné (Noyau linux 2.6 : 4 millions ; GCC : 2,5 millions ; Windows XP : 40 millions, ...).

▷ **Question 5.** Faites un programme C `complexite` prenant le nom d'un fichier en argument et comptant le nombre de lignes le composant (il faut compter les occurences du caractère '\n').

▷ **Question 6.** Modifiez votre programme pour ne pas tenir compte des lignes blanches (il faut utiliser un booléen réinitialisé à chaque ligne indiquant si on a vu un caractère autre que '\t' et ' ').

▷ **Question 7.** (facultative) Modifiez votre programme pour ne pas compter ce qui se trouve à l'intérieur de commentaires. On souhaite traiter à la fois le style C (`/* ... */`) et le style C++ (`// ... fin de ligne`). On utilisera un booléen indiquant si on se trouve actuellement dans un commentaire ou non.

Cette métrique de complexité est trompeuse car un long programme ne comportant qu'un enchaînement de commandes sans `if`, `for` ou autres est sans doute moins complexe qu'un programme plus court présentant un schéma d'exécution plus complexe.

▷ **Question 8.** Modifiez votre programme pour qu'il compte les occurences du caractère '{', qui est présent dans la plupart des structures syntaxiques du C.

Voici deux extensions possibles pour votre programme, à réaliser si vous avez le temps :

- Comptez séparément les occurences des différentes constructions syntaxiques du C.
- Faites en sorte que votre programme compte les lignes lorsqu'on lui passe l'argument supplémentaire `--long` tandis qu'il compte la complexité si on lui passe l'option `--complexe`.