Département IMA / 3A (S6) / 2013 **Programmation Avancée**
Jonathan Aceituno, Vincent Aranéga, Nathalie Devésa, Rachid Elkouri et Laure Gonnord

TP10 Un peu de généricité et fonctions à nb variables d'arguments

Objectifs

- Savoir passer une fonction en paramètre.
- Savoir utiliser les void*.
- Manipuler des fonctions qui ont un nombre quelconque d'arguments.

On écrira un fichier .c par exercice.

1 Passer une fonction en paramètre

Pour rendre plus générique certaines fonctions (traitement des noeuds d'un arbre binaire lors d'un parcours, par exemple), on peut passer des fonctions en paramètres d'autres fonctions/procédures. On pourra se référer par exemple au document http://www.newty.de/fpt/zip/f_fpt.pdf (section 2, partie C uniquement).

Exo 1 : utilisation simple

1. Écrire une fonction `fois_deux` qui retourne un entier égal au double de l'entier passé en paramètre. Quelle est la signature de cette fonction ?
2. Écrire une fonction `appliquer_tableau(int f(int), int t[], int size)` qui applique la fonction f à chacun des éléments du tableau t.
3. Tester dans le `main` :

```
int tab[]={1,2,3,4};
int (*f) (int) = appliquer_tableau; // remarquer la déclaration et l'init!
appliquer_tableau(f,tab,4);
```

On remarquera qu'on déclare un pointeur de fonction :-)

Exo 2 : modifier la fonction de comparaison d'un tri d'entiers

1. Récupérer dans vos codes de TP précédents un tri de tableaux d'entiers (par exemple le tri sélection).
2. Que faut-il modifier pour faire un tri décroissant ?
3. Écrire une fonction `superieur(int a, int b)` qui renvoie 1 si a est strictement supérieur à b, 0 si $a = b$, -1 sinon.
4. Écrire une fonction `inférieur` sur le même principe.
5. Rendre votre fonction de tri générique en passant une fonction compare en paramètre.
6. Tester dans le `main`

2 Faire de la généricité

Exo 3 : utilisation de `qsort` Dans la même veine que précédemment, la libc fournit une fonction de tri générique sur n'importe quel tableau (entiers, caractères, etc), pourvu qu'on lui passe la fonction de comparaison en paramètre.

1. Faire `man qsort` pour regarder le prototype de la fonction.

   ```
   void qsort(void *base, size_t nmemb, size_t size,
                     int(*compar)(const void *, const void *));
   ```

2. En lisant le manuel, déterminer quel argument correspond au tableau à trier, la taille d'un élément du tableau, la taille du tableau, et la fonction de comparaison. À votre avis, à quoi correspondent les `void*` ?

3. Note : `const` signifie que l'argument n'est pas modifié dans la fonction.

4. Utiliser cette fonction pour trier un tableau d'entiers en utilisant la fonction compare de l'exercice précédent. On modifiera la fonction pour qu'elle ait le bon type :

   ```
   int compare (const void* pav, const void* pbv)
   ```

5. Utiliser pour trier un tableau de chaînes de caractères en utilisant `strcmp`. Tester.

Remarque Cette généricité sera plus facile à mettre en oeuvre dans des langages de plus haut niveau, notamment c++ et Java.

3 Nombre variables d'arguments

Cette partie est plus exotique, beaucoup moins utile. Elle est inspirée de http://cedric.cnam.fr/~lamberta/C.html.

Exo 4

1. Récupérer sur la page du cours le fichier `tp10_nbvar.c`.

2. Compiler, tester, comprendre

3. En s'inspirant de cette fonction, écrire une fonction `imprime_concat` qui prend en argument fixe un caractère `sep`, puis une liste de chaînes de caractères terminée par un pointeur NULL, et affiche les chaînes dans l'ordre, séparées par le caractère `sep`.

4 Pour ceux qui ont fini

Voici en vrac des choses à faire :

1. Essayer de construire une fonction d'impression de tableau générique

2. Essayer de construire un type liste générique.

3. Finir le TP de GDB.