Enseignants : A. Bonhomme, L. Bougé le 08/01/01
Email : {Alice.Bonhomme,Luc.Bouge}@ens-lyon.fr

# TD Programmation 12 : Compléments de C

## 1 Écriture d'un `toplevel` à l'aide de `setjmp/longjmp`

L'objectif de cet exercice est d'écrire un petit toplevel qui saisit au clavier des instructions, et exécute les opérations associées, qui ont été précédemment définies. Ainsi, dans le code du toplevel, on a un ensemble de couple (*nom d'instruction*, *pointeur sur la fonction à exécuter*). Pour simplifier, on utilisera des fonctions qui ne prennent pas d'argument et qui affichent leur nom à l'écran. Un couple sera donc représenté par la structure suivante :

```
#define N 3

typedef struct{
   char op_name[40];
   void (*fct)(void);
} instruction;

instruction tab[N];
```

Le tableau `tab` est donc un tableau d'instructions, et on a des opérations du type :

```
void f2(void) {
  printf("Called: f2\n");
}
```

- Écrire l'initialisation du tableau `tab` avec 3 fonctions que vous définirez.
- Écrire une boucle qui saisit un nom de fonction et lance l'exécution de cette fonction.
- Pour pouvoir arrêter le toplevel il faut une fonction qui a un traitement différent. Cette fonction sera appelée `end`. Une première solution est de rajouter un appel à la fonction `exit` dans la fonction `end`. Une autre solution est d'utiliser les fonctions `setjmp` et `longjmp`. La fonction `end` fait un branchement (via `longjmp`) vers le programme

principal (qui a fait un `setjmp`). En testant la valeur retournée par la fonction `setjmp`, le programme décide de se terminer ou non.

- Écrire la fonction `end` avec la première solution.
- Écrire la fonction `end` avec la deuxième solution et modifier la boucle principale en conséquence.

# 2 Gestion de signaux

## 2.1 Rattraper un signal

L'objectif de cet exercice est d'écrire des programmes simples pour apprendre à manipuler les fonctions `raise` et `signal`.

- Écrire un programme dont la boucle principale est une boucle infinie. Une façon d'interrompre ce programme est de taper Ctrl C. Rajouter à votre programme le rattrapage de ce signal (avec la fonction `signal`). Lors du rattrapage de ce signal, afficher un texte à l'écran puis sortir du programme.
- Modifier ce programme pour qu'il se termine non pas lors de la réception d'un Ctrl C mais lorsqu'il en a reçu 5 ( utiliser une variable statique pour le compteur).
- Écrire un programme qui rattrape une division par 0 faite dans la boucle principale. Que se passe-t-il lors de l'exécution de votre programme ? Pourquoi ? Modifier le programme précédent pour obtenir un comportement raisonnable.

## 2.2 Utilisation de signaux dans le `toplevel`

Dans le toplevel écrit dans le premier exercice, il est possible d'effectuer tout type d'opération. En particulier, il est possible d'exécuter des opérations qui sont un peu longues. Aussi, on souhaiterait pouvoir interrompre ces opérations, sans toutefois sortir du toplevel.

Une solution est d'envoyer un signal (Ctrl C par exemple) au toplevel qui le rattrape et fait le traitement associé.

- Écrire une opération `work_hard(void)` qui imprime un point à l'écran toutes les secondes pendant un temps assez long, 10 secondes par exemple.
- Proposer une solution à l'aide de signaux, permettant d'interrompre l'opération `work_hard` avec un Ctrl C et d'activer `longjmp`. Noter que la valeur retrournée par `setjmp` doit être différente de la valeur qui force la terminaison !

# 3 Allocation et réallocation de tableaux

L'objectif de cet exercice est de manipuler les fonctions d'allocation et de réallocation dynamiques.

Pour ceci, on considère un tableau d'entiers représenté par une structure comportant le pointeur sur la zone mémoire allouée pour ce tableau, et la taille du tableau. Cette structure est définie de la manière suivante :

```
typedef struct {
    int * tab;
    int size;
 } Stab;
```

- Écrire une fonction `init(Stab * tab, int size)` qui initialise le tableau et alloue une zone mémoire de taille `size`.
- Écrire une fonction `put(Stab tab, int i, int item)` qui ajoute l'élément `item` dans la ième case du tableau. Si le tableau n'est pas assez grand (c'est-à-dire si `i` est plus grand ou égal à la taille du tableau), celui-ci doit être agrandi (utilisez la fonction `realloc`).
- Écrire une fonction `get(Stab tab, int i, int *item)` qui récupère l'élément qui est dans le ième case du tableau après avoir vérifié que celui-ci est bien dans l'intervalle de définition.