POLYTECH NICE-SOPHIA

SI 3 - 2017-2018

Programmation Procédurale

Feuille 5

Pointeurs – Allocation dynamique

Exercice 1: Pile de nombres

Écrire les fonctions permettant de gérer une pile d'entiers de taille quelconque. Les éléments de la pile seront du type :

```
struct element {
  int valeur;
  struct element *suivant;
};

typedef struct element Element;
```

Note: Cette représentation n'est pas du tout efficace en termes d'occupation mémoire. Toutefois, ce qui nous intéresse ici c'est la manipulation de la liste chaînée permettant de représenter une pile.
Les fonctions à implémenter sont les suivantes:

- `push_item` pour ajouter un élément au sommet de la pile;
- `pop_item` pour enlever un élément de la pile;
- `top_value` qui renvoie l'entier en sommet de pile;
- `print_stack` pour imprimer le contenu de la pile.

Écrire deux implémentations de la pile:

- une implémentation où les fonctions de gestion de la pile renvoient la nouvelle valeur de la pile;
- une implémentation où l'on passe la pile en paramètre par référence (ce qui nous conduira à avoir des pointeurs sur des pointeurs).

Exercice 2: Listes chaînées

On veut pouvoir représenter des listes de nombres entiers en C. Pour cela, on réutilise le type d'éléments que l'on avait utilisé dans l'exercice précédent.

- Définir le type `List` permettant de représenter une liste d'entiers.
- Écrire les fonctions C suivantes:

```
// Ajouter la valeur v à la fin de la liste lst
List prepend_element(List lst, int v);

// Ajouter la valeur v au début de la liste lst
List append_element(List lst, int v);

// Ajouter la valeur v dans la liste ordonnée lst
List insert_element(List lst, int v);
```

```
// Supprimer la (première) valeur v de la liste lst
List delete_element(List lst, int v);

// Tester si la valeur v est dans liste lst
int find_element(List Lst, int v);

// Imprimer la liste lst
void print_list(List lst);
```

- Modifier les fonctions précédentes qui ont résultat de type `List` pour qu'elles soient maintenant de type `void`. Modifiez aussi, bien-sûr, votre programme de test en conséquence.

Exercice 3: Lecture de chaînes de taille quelconque

Écrire une fonction permettant de lire une chaîne de caractères de taille quelconque sur le fichier standard d'entrée. Pour cela, on commencera par allouer une zone mémoire de taille `TBLOC`, cette zone mémoire étant étendue si nécessaire à l'aide de la fonction `realloc`. Cette fonction devra renvoyer une copie de la chaîne de caractères lue (i.e. dont la taille est égale au nombre de caractères lus). Lorsqu'on est en fin de fichier la fonction renverra `NULL`.

Exercice 4: La fonction strtok

Écrire la fonction `strtok` dont le prototype est:

```
char *strtok(char *s, const char *delim);
```

Vérifier le comportement de cette fonction dans le manuel. Un exemple d'utilisation de cette fonction est donné ci-dessous.

```
#include <stdio.h>
#include <string.h>
int main(void)
{
  char phrase[] = "Une phrase composée de mots."
  char *p;
  for (p = strtok(phrase, " ."); p ; p = strtok(NULL, " ."))
    printf("mot '%s'\n", p);
  return 0;
}
```

```
$ ./strtok
mot 'Une'
mot 'phrase'
mot 'composée'
mot 'de'
mot 'mots'
$
```