TP6 - Pile statique, compilation séparée

On veut créer un programme C implémentant une pile statique d'entiers. Ce programme sera par la suite une boîte à outils permettant d'utiliser une pile dans un autre programme. Dans ce TP, vous devez écrire de nombreux sous-programmes **en les testant impérativement au fur et à mesure de leur écriture**.

Implémentation d'un pile au moyen d'un tableau

Première étape

Cette étape rassemble les procédures dépendant du type de données stockées dans la pile.

1. Définir un type ELEMENT qui sera le type des éléments de la pile. Ce type sera pris équivalent à **int** pour simplifier,
2. Ecrire les procédures *Affiche_Elément* et *Saisir_Elément* permettant respectivement d'écrire un élément (type ELEMENT) à l'écran et de lire un élément au moyen du clavier. L'élément sera passé en paramètre,
3. Ecrire une procédure *Affecte* qui reçoit en paramètre deux variables de type ELEMENT et qui affecte la première variable avec la valeur de la deuxième. Si le type ELEMENT était un enregistrement la procédure devrait copier tous les champs.

Deuxième étape

Dans cette partie on implémente la structure de donnée PILE (LIFO en anglais : Last In First Out). On choisit une implémentation statique utilisant un tableau de taille fixée (TMAX déclarée au moyen d'un #**define**). Les éléments sont ajoutés dans le sens des indices croissants du tableau. L'élément en tête de pile est repéré par son indice mémorisé par une variable (Tête). Insérer un élément c'est incrémenter la tête et mettre le nouvel élément dans la case d'indice tête. Retirer un élément consiste à décrémenter la tête (il n'est pas nécessaire de supprimer physiquement l'élément).

1. Définir le type PILE sous forme d'une structure ayant deux champs : le tableau et la tête. Les éléments du tableaux sont de type ELEMENT,
2. Définir la procédure *Initialise_Pile* qui initialise une pile,
3. Définir la procédure *Affiche_Pile* qui écrit à l'écran tous les éléments d'une pile. Cette procédure utilisera *Affiche_Elément*,
4. Définir la procédure *Saisir_Pile* qui demande à l'utilisateur des éléments et les insère dans la pile,

5. Définir la fonction *Pile_Vide* qui teste si une pile est vide. Pour clarifier la programmation on utilisera deux constantes : **#define TRUE 1** et **#define FALSE 0**. On peut aussi utiliser **typedef int boolean** pour pouvoir déclarer les entiers représentant des booléens par : **boolean var ;**
6. Définir la fonction *Pile_Pleine* qui teste si une pile est pleine,
7. Définir la procédure *Empiler* qui reçoit un élément et une pile et ajoute l'"élément dans la pile,
8. Définir la procédure *Dépiler* qui supprime la tête de la pile,
9. Définir la fonction ou procédure *Sommet_Pile* qui renvoie l'élément sommet de la pile.

Modularité

Nous allons aborder la programmation modulaire. Sauver toutes les procédures de la première étape dans un fichier élémentpile.c. Mettre le type ELEMENT dans pile.h ainsi que tous les **#define** et les types définis. Ajouter à élémentpile.c la ligne **#include "pile.h"**. Créer maintenant créer un fichier pile.c contenant les procédures de la deuxième étape et utilisant les procédures de élémentpile.c. Ce fichier devra aussi inclure pile.h. Il nous faut un dernier fichier main.c pour le programme principal.

Pour compiler ce programme on peut utiliser : gcc main.c pile.c élémentpile.c -o pile. Ou encore :

```
gcc -c élémentpile.c
gcc -c pile.c
gcc -c main.c
gcc élémentpile.o pile.o main.o -o pile
```

On peut aussi définir un fichier de configuration pour que la compilation se fasse automatiquement.

Commande make et fichier Makefile

La commande **make** recherche dans le répertoire de travail courant un fichier de nom **Makefile** contenant la description modulaire du programme. Par défaut, **make** essaie de réaliser la première spécification décrite par le fichier **Makefile** donnant les dépendances de la compilation. Il suffit donc de taper la commande **make** pour que la description soit exécutée. Le fichier **Makefile** doit contenir les informations suivantes :

- pile est construit à partir de élémentpile.o, de pile.o et de main.o
- élémentpile.o est contruit à partir de élémentpile.c
- pile.o est construit à partir de pile.c
- main.o est construit à partir de main.c

Un certain nombre de variables résevées permet de définir le compilateur par défaut, les librairies, etc. La construction $(<var>) permet d'obtenir la valeur de <var>.

Exemple de Makefile :

```
CC=gcc
OBJ=élémentpile.o pile.o main.o

util:       $(OBJ)
            $(CC) $(OBJ) -o pile

élémentpile.o élémentpile.c
```

```
pile.o:   pile.c

main.o:   main.c
```

Créer le fichier Makefile décrit ci-dessus et tapez make.