TP 6 : Structures, piles, listes

Objectifs :

- S'initier à la manipulation de structures de données fondamentales

Stéphanie CHOLLET – Ioannis PARISSIS
El Mehdi KHALFI – Ayoub SOURY

Structures

1. Notions de base

Lire les sections 6.1 à 6.4 du polycopié ainsi que les sections 9.8 et 9.9.

2. Exercice

2.1. Fractions

On considère le type Fraction (portion de code à insérer dans un nouveau fichier Fraction.h) :

```
typedef struct {
  int num ; /* numérateur */
  int den ; /* dénominateur */
} Fraction ;
```

Nous souhaitons rédiger quatre procédures réalisant les quatre opérations arithmétiques sur des fractions et affichant le résultat. Le profil des procédures est le suivant :

```
void addFraction(Fraction f1, Fraction f2) ;
void subFraction(Fraction f1, Fraction f2) ;
void mulFraction(Fraction f1, Fraction f2) ;
void divFraction(Fraction f1, Fraction f2) ;
```

Chacune de ces procédures calcule la fraction correspondant à l'opération entre ses deux paramètres puis *affiche le résultat* sous la forme d'une fraction réduite (voir exemple ci-dessous). Pour cela, on utilisera une fonction qui calcule le pgcd de deux entiers. Ecrire ces procédures (dans nouveau fichier Fraction.c) et compléter la fonction main()(à insérer dans un nouveau fichier main.c) permettant de les tester comme suit (*entrées utilisateur en italiques*):

Entrer deux fractions : 3/4 4/8
*Entrer une opération (+, -, /, *) : **
Le résultat est 3/8.

```
int main(int argc, char* argv[]){

      Fraction fa, fb ;

      printf("Entrer deux fractions :") ;
      scanf("%d/%d %d/%d", &fa.num, &fa.den, &fb.num, &fb.den) ;
      ...

}
```

Piles, files, listes

3. Notions de base

Nous supposons que les structures de pile, file et liste ont été étudiées en algorithmique. Les exercices ci-dessous permettent d'aborder l'utilisation de ces structures en C (*l'utilisation de pointeurs est volontairement évitée ou « masquée »*).

4. Exercices

4.1. Calcul post-fixe

On veut évaluer des expressions arithmétiques écrites en notation polonaise inversée (ou notation post-fixe). Rappel du principe : tout opérateur arithmétique n'est pas placé entre ses arguments (comme avec la notation courante du type 4 + 7) mais après ses arguments. Ainsi 4 + 7 se note 4 7 +. Par exemple, l'expression (4+2)x5/(6-7) se note en post-fixe 4 2 + 5 x 6 7 - /. L'évaluation d'une telle expression se base sur l'utilisation d'une pile.

Écrire une fonction qui prend en paramètre une telle expression formée des quatre opérateurs +, -, x, / et des nombres entiers de 0 à 9, stockée dans un tableau de N caractères. **On suppose que l'expression est correcte.**

Pour réaliser cet exercice, vous trouverez dans le répertoire pile *les fichiers pile.h et pile.c, fournissant une réalisation simple d'une pile d'entiers (le fichier* main.c*, donné à titre d'exemple, en illustre l'utilisation).*

4.2. Liste chaînée

On souhaite gérer une liste chaînée d'entiers, implémentée à l'aide d'un tableau. On considère les déclarations suivantes (présentes dans le fichier `listeTableau.h` fourni dans le répertoire `listeTableau`) :

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

Nous adoptons le principe de représentation suivant :

Soit L une variable de type liste.
L[0].valeur est non significatif ;
L[0].suivant est l'indice du tableau correspondant au premier élément de la liste.
L[i].suivant est l'indice du tableau contenant le successeur de celui situé à l'indice i.
L[i].suivant = 0 signifie que l'élément d'indice i est le dernier de la liste.
L[i].suivant = -1 signifie que l'élément d'indice i est inoccupé (donc, on ne devrait pas avoir d'élément j dans le tableau tel que L[j].suivant = i).

Ainsi, dans un tableau de taille N on peut représenter une liste d'au plus N-1 éléments.

Exemple : représentation de la liste (3, 4, 12, 15)

X	3	4	4	15	0	3	1	12	2
0		1		2		3		4	

Compléter le fichier `listeTableau.c` du répertoire `listeTableau` avec les fonctions C permettant de gérer la liste et tester en complétant le programme `main.c`. Ce dernier doit tester plusieurs cas : (au moins) insertion et suppression et tête, en fin et en milieu de liste.

- `int elementLibre(liste l)`
- `void creerListe(liste l)` : crée une liste vide (initialise le premier élément ainsi que les éléments inoccupés).
- `void afficherListe(liste l)` : affiche tous les éléments de la liste dans l'ordre.
- `void insererElement(int x, liste l)` : en supposant l triée, insère x dans l en maintenant L triée
- `void supprimerElement(int i, liste l)` : supprime le i-ème élément de l.

Question optionnelle : écrire une fonction permettant de « compacter » le tableau contenant une liste de P éléments de sorte qu'elle occupe les P+1 premières cases du tableau.

- `void compacterListe(liste l)`