Algorithmique et programmation procédurale

TP5 – Liste, File, Pile

Exercice 1 : Liste Chainée

a) Définir en C le type de donnée « liste d'entiers » via la notion de nœud :

```
typedef struct Noeud {
      int valeur;
      struct Noeud* suivant;
} Noeud;
```

Implémenter les opérations suivantes :

- `Noeud* ajoutTete(Noeud* teteliste, int n)`: ajoute un nœud contenant la valeur n en début de liste
- `Noeud* ajoutFin(Noeud* teteliste, int n)`: ajoute un nœud contenant la valeur n en fin de liste
- `Noeud* ajoutKieme(Noeud* teteliste, int k, int n)`: ajoute un nœud à la kième position
- `Noeud* supprTete(Noeud* teteliste)`: supprime le nœud en tête de liste
- `Noeud* supprFin(Noeud* teteliste)`: supprime le nœud en fin de liste
- `Noeud* supprKieme(Noeud* teteliste, int k)`: supprime le kième noeud

Attention à bien gérer les exceptions comme par exemple : l'ajout à une position k > longueur (liste), la suppression dans uns liste vide, etc.

b) Ecrire un programme principal qui permet de tester vos fonctions. A partir d'une liste vide au démarrage, l'utilisateur doit être capable d'effectuer via un menu textuel les opérations de l'exercice précédent. Après chaque action, votre programme affiche l'état actuel de la liste.

Exercice 2 : Calculette polonaise

Le défi : écrire un programme qui évalue une expression arithmétique en notation polonaise. L'utilisateur saisit une telle expression (à vous de voir comment gérer cette saisie le plus facilement), et le programme répond par le résultat du calcul, ou bien par un message d'erreur au cas où l'expression était erronée.

Aide

Nous définissons d'abord la structure de donnée « Pile de flottants » qui permettra de conserver les calculs intermédiaires:

```
typedef struct ElementListe {
  float donnee;
  struct ElementListe* suivant;
} Element;

typedef struct {
  Element* debut; // le premier élément de la pile
  int taille; // on retient la taille actuelle de la pile
} Pile;
```

Implémenter les fonctions de pile suivantes

- `Pile* creerPile()`: revoie un pointeur sur une pile vide
- `void push (Pile* p, float x)`: empile le nombre réel x
- `void pop (Pile* p)`: dépile le plus haut élément
- `float peek (Pile* p)`: renseigne sur le flottant en haut de la pile
- `int estVide (Pile* p)`: renseigne si la pile est vide (0 = faux, 1 = vrai)
- `void libererPile(Pile* p)`: détruit la pile et libère toute mémoire allouée

Ensuite, nous devons créer un type de donnée qui contient soit un flottant, soit un opérateur (+, -, *, /). Ceci est possible en C grâce au concept d'union :

```
enum typeElement { OPERANDE, OPERATEUR };
union dataElement {
      float operande;
      char operateur;
};
typedef struct
{
      enum typeElement type;
      union dataElement data;
} Element;
```

Ainsi, une expression polonaise peut être représentée par un tableau ou une liste chainée de valeurs du type `Element.` Chaque case contient soit un `OPERANDE`, soit un `OPERATEUR`.
Une fois que l'utilisateur a saisi l'expression polonaise, votre programme lit cette expression `Element` par `Element` et maintient dans une pile les calculs intermédiaires selon l'algorithme vu en cours (opérande => empiler, opérateur => dépiler deux fois et empiler le résultat de l'opération). A la fin d'un calcul correct, nous trouvons le résultat sous forme d'un seul et dernier flottant dans la pile.

Bon courage, ou plutôt « powodzenia »!