Licence 2, Partiel LC4, Développement en C

15 mars 2008, durée 2h

Le sujet comporte 4 pages. *Documents manuscrits et tous les documents de cours sont autorisés (tds, tps, corrections, cours).* **Les livres ne sont pas autorisés.** *Tous les exercices sont indépendants. Il n'est pas demandé de faire* `#include` *pour les fichiers en-tête de la bibliothèque standard.*

Votre code doit être écrit de façon lisible, avec des indentations et des accolades appropriées permettant de distinguer les fins de blocs de code (fins de boucles, ...).

Le barème est donné seulement à titre indicatif.

Exercice 1

Qu'affichera le programme suivant:

```
#include <stdio.h>
#include <stdlib.h>
int main(void){
  double tab[] = { 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0 };
  double *d = tab;
  double *f = d + sizeof tab / sizeof(double);
  int  i = 4;
  double *p = &tab[i];
  double x;
  printf("%3.1f\n",*(d+2));
  printf("%3.1f\n",*d + 2);
  printf("%d\n",p-d);
  printf("%3.1f\n",*(p-2)+3);
  x = *p++;
  printf("f-p = %d\n", f-p);
  printf("x = %3.1f, *p= %3.1f\n", x, *p);
  return EXIT_SUCCESS;
}
```

Exercice 2

Écrire une fonction

```
char * coder(const char *text, const char *from, const char *to)
```

qui sera utilisée pour coder la chaîne `text`. La fonction retournera une nouvelle chaîne de caractères de même longueur que la chaîne `text` (il faut allouer la mémoire pour la nouvelle chaîne, la chaîne de `text` ne doit pas être modifiée). Les chaînes `from` et `to` donnent une correspondance de lettres dans le codage. Par exemple `from=''abacd''` et `to= ''xyzx''` définissent la correspondance suivante:

a $\longrightarrow$ x,
b $\longrightarrow$ y,
c $\longrightarrow$ x,

c'est-à-dire la première lettre de `from` correspond à la première lettre de `to`, la deuxième lettre de `from` correspond à la deuxième lettre de `to`, etc.

Noter toutefois que seule la première occurrence d'une lettre dans `from` compte, par exemple la deuxième occurrence de a dans `from` ne donne pas lieu à une correspondance.

Noter aussi que la lettre `d` dans `from` n'a pas de correspondance parce que la chaîne `to` est plus courte.

La chaîne codée sera construite de la façon suivante: chaque caractère qui n'a pas de correspondance sera recopié sans changement, si un caractère dans `from` a une correspondance dans `to` alors c'est le caractère codé qui sera recopié dans la chaîne codée. Dans notre exemple les caractères `a,b,c` seront remplacés respectivement par `x,y,x` tandis que tous les autres caractères seront recopiés sans modification.

Exercice 3

On représente un polynôme $p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$ par le type `polynome` défini de façon suivante:

```
struct poly{
     int n;
     double *coeffs;
};
typedef struct poly *polynome;
```

c'est-à-dire par un pointeur vers une structure dont le premier champ donne le degré du polynôme et le deuxième champ est un pointeur vers le tableau de coefficients, le coefficient a_i se trouvant dans le i-ème élément du tableau.

Par exemple la figure 1 représente le polynôme $f(x) = 2.0 - 6.0x^2 + 2.0x^3$.

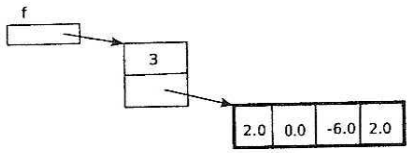


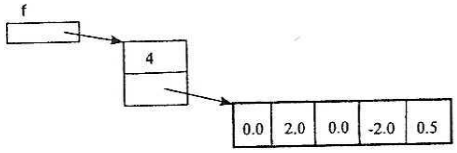
Figure 1: Polynôme $f(x) = 2.0 - 6.0x^2 + 2.0x^3$.

Écrire une fonction

`polynome integral(polynome p)`

qui calcule et retourne le polynôme de l'intégrale de $p(x)$. Pour $p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$ son intégrale est $q(x) = a_0x + \frac{a_1}{2}x^2 + \frac{a_2}{3}x^3 + \cdots + \frac{a_n}{n+1}x^{n+1}$, c'est-à-dire le coefficient de x^0 est 0 et le coefficient de x^i est $\frac{a_{i-1}}{i}$ pour $i > 0$.

Par exemple pour le polynôme de la figure 1 la fonction doit construire le polynôme du degré 4 présenté ci-dessous:



Exercice 4

Une pile de nombres doubles sera représentée par le type suivant:

```
struct ep{
    int erreur;
    double *haut;
    double *courant;
    double *bas;
};
typedef struct ep *pile;
```

La pile est implantée par un tableau de nombres doubles géré avec des pointeurs: le pointeur `bas` donne l'adresse du premier élément tableau, le pointeur `haut` donne l'adresse juste après le dernier élément du tableau, `courant` donne l'adresse du premier élément libre dans le tableau. Le champ `erreur` sera utilisé pour signaler des erreurs lors des opérations `empiler`, `depiler`, etc.

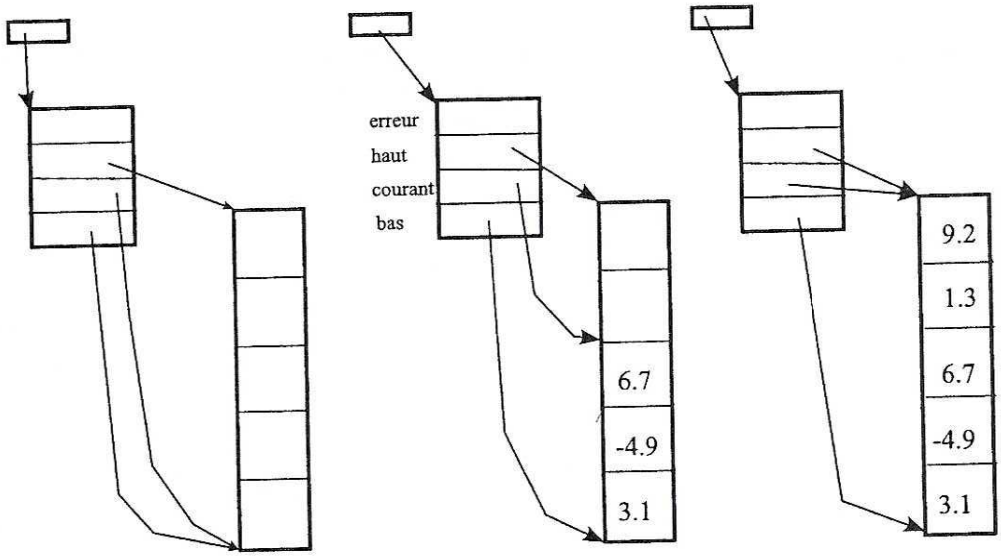


Figure 2: Le dessin représente trois états d'une pile de capacité 5, la pile à gauche est vide, la pile au milieu contient 3 éléments, 3.1 est au fond de la pile et 6.7 au sommet, la pile à droite contient déjà 5 éléments, elle est pleine. Pour ajouter un élément dans la pile à droite il faudra agrandir le tableau utilisé pour stocker les éléments.

Les questions qui suivent implémentent les opérations sur une pile. Même si toutes ces questions forment un ensemble, elles peuvent être traitées dans n'importe quel ordre. En particulier, si vous êtes bloqués sur une question n'hésitez pas à passer à la question suivante. De plus en traitant une question on peut supposer que les fonctions demandées dans les questions précédentes sont déjà disponibles même si vous ne les avez pas implantées.

Question 1: Écrire une fonction

```
pile faire_pile(int capacite)
```

qui construit une pile vide qui peut stocker jusqu'à `capacite` d'éléments doubles.

Question 2: Écrire une fonction

```
int pile_vide(pile p)
```

qui retourne 1 si la pile est vide et 0 sinon.

Question 3: Écrire une fonction

```
double depiler(pile p)
```

qui dépile et retourne l'élément au sommet de pile. Si la pile est vide, la fonction retourne 0 et met 1 dans le champ `erreur`. Si la pile était non-vide, la fonction met 0 dans le champ `erreur`

Question 4: Écrire une fonction

```
int empiler(pile p, double d)
```

qui met `d` au sommet de la pile.

Si la pile était pleine avant de mettre `d` sur la pile, il faut doubler la capacité de la pile (`realloc`).

La fonction met la valeur 0 dans le champ `erreur` sauf si elle a tenté d'agrandir la pile et échoué, auquel cas il faut mettre 2 dans le champ `erreur`.

La fonction retourne la valeur mise dans `erreur`.

Question 5: Écrire une fonction

```
int operation(pile p, char op)
```

Le paramètre `op` peut prendre une des quatre valeurs: `'+'` `'-'` `'*'` `'/'`. La fonction dépile deux nombres doubles du sommet de la pile, effectue sur ces deux nombres l'opération indiquée par le paramètre `op` (addition, soustraction, multiplication, division) et elle met le résultat de l'opération au sommet de la pile.

Si le paramètre `op` n'est pas correct, la fonction met 5 dans le champs `erreur`.

Si la pile contenait moins de 2 éléments et que l'opération ne peut pas être effectuée faute d'un nombre suffisant d'éléments sur la pile alors la fonction met 6 dans le champ `erreur`, dans le cas contraire il faut mettre 0 dans `erreur`.

La fonction `operation` retourne la valeur mise dans `erreur`.

Question 6: Écrire une fonction

```
void detruire(pile p)
```

qui libère la mémoire occupée par la pile `p` (tableau et structure).