UGA - UFR IM^{2}AG **Master 2 CCI**

Programmation en C

Examen du 8 décembre 2021 (session 1)

Durée : deux heures – **Tous documents autorisés** – Les questions marquées d'une * sont plus difficiles

Exercice 1 : exécution d'un programme C ($\sim$ 4 points)

Quels sont les affichages produits à l'écran lors de l'exécution du programme C ci-dessous :

```
#include <stdio.h>

void f1(int a, int b) {
  a = a*b ;
}

int f2(int a, int *b) {
  int x ;
  x = a + *b ;
  return x ;
}

int main() {
  int x, y ;
  int *p ;
  x=2 ; y=5 ;
  f1 (x, y) ;
  printf("(1) x=%d, y=%d\n", x, y) ;
  p = &y ;
  y = f2(x, p) ;
  printf("(2) x=%d, y=%d\n", x, y) ;
  return 0 ;
}
```

Exercice 2 : codage RLE (∼ 6 points)

Le codage RLE (*Run Length Encoding*) est une technique de compression que l'on peut appliquer à des séquences de données. Elle consiste à remplacer toute sous-séquence d'au moins 2 éléments consécutifs identiques (par exemple `"XXXXXX"`) par la sous-séquence comprenant le nombre de répétitions suivi de l'élément répété (`"6X"` sur l'exemple précédent).

Dans cet exercice nous allons appliquer une version simplifiée de ce codage à des chaînes de caractères, terminées par la marque de fin `'\0'`, et ne **comportant pas de chiffres** (donc pas de caractères entre `'0'` et `'9'`).

Ainsi, la chaîne `"AAAAHHHHHHHH!"` sera codée en RLE par `"4A8H!"`.

Q1. Quel problème peut-il se poser si on applique ce codage à une chaîne contenant des chiffres ?

Q2. On donne un algorithme qui lit au clavier une chaîne de caractères (sans chiffres) et affiche à l'écran son codage RLE :

```
lexique
    M : la constante caractère '\0'
    N : la constante entière 256
    c : un tableau sur [0..N] de caractères
    i, j : des entiers positifs
debut
    lire (c)
    i ← 0
    tantque c[i] ≠ M
        j ← i+1
        tantque c[j] ≠ M et puis c[i] = c[j] : j ← j+1
        si j > i+1 alors ecrire(j-i)
        ecrire(c[i])
        i ← j
fin
```

Ecrivez ce programme en C.

Q3(*). Ecrivez en C un programme qui lit au clavier une chaîne de caractères codée en RLE et affiche à l'écran la chaîne décodée. Ainsi, si le programme lit en entrée la chaîne codée `"4A8H!"`, alors il affichera la chaîne `"AAAAHHHHHHHH!"`.

On supposera que les entiers présents dans la chaîne codée en RLE sont toujours strictement inférieurs à 10. Pourquoi cette hypothèse simplifie t'elle le programme à écrire ?

Exercice 3 : Représentation de polynômes ($\sim$ 10 points)

Un *polynôme* est une séquence (non ordonnée) de *monômes*, où chaque monôme peut être défini par un couple de deux entiers (c, d), avec $c \in \mathbb{Z}$ et $d \in \mathbb{N}$, représentant respectivement le *coefficient* et le *degré* du monôme. Ainsi, le polynôme $P = 8x^3 + 5x^2 - 1$ peut être représenté par la séquence

$$[(8, 3), (5, 2), (-1, 0)]$$

Le fichier `polynome.h` donné en Annexe fournit des primitives permettant de parcourir et modifier un polynôme.

Q1. Ecrivez en C la fonction suivante qui renvoie la valeur du polynôme `P` pour x valant `v` :

```
int Evaluer(Polynome P, int v) ;
```

On supposera que `P` est un polynôme non vide et on utilisera (sans l'écrire) la fonction `int puissance (a,b)` supposée renvoyer $\texttt{a}^{\texttt{b}}$.

Q2. La dérivée d'un polynôme est la séquence des dérivées de chacun de ses monômes de degré strictement positifs. La dérivée du monôme (c, d) pour $d > 0$ est le monôme $(c \times d, d - 1)$. Ainsi, la dérivé du polynôme $P = 8x^3 + 5x^2 - 1$ est représentée par la séquence

$$[(24, 2), (10, 1)]$$

Ecrivez en C la fonction suivante qui produit dans le paramètre résultat `P2` la dérivée du polynôme `P1` :

```
void Deriver(Polynome P1, Polynome *P2)
```

Q3. On souhaite représenter un polynôme par une **liste chaînée** de monômes.

1. Complétez la définition des types `Monome` et `Polynome` du fichier `Polynome.h` ;
2. Donnez le contenu du fichier `Polynome.c` qui implémente les fonctions définies dans `Polynome.h`

Q4(*). Ecrivez en C la fonction suivante qui produit dans le paramètre résultat `P3` la somme des polynôme `P1` et `P2` :

```
int Addition(Polynome P1, Polynome P2, Polynome *P3) ;
```

On supposera pour cette question que les polynômes sont des séquences de monômes **ordonnées par ordre décroissant de leurs degré**.

Annexe : le fichier `polynome.h`

```
typedef ... Monome ; // definition du type Monome

typedef ... Polynome ; // definition du type Polynome

/* Constructeurs */

Monome CreerMonome (int c,  unsigned int d) ;
// renvoie le monome (coefficient, degre)

void AjouterMonome (Polynome *P, Monome M) ;
// ajoute le monome M au polynome P
// P est donc modifié par cette action (parametre resultat)

Polynome PolynomeVide() ;
// renvoie un polynome "vide"
// (que l'on pourra compléter en lui ajoutant des monomes ...)

/* Selecteurs */

unsigned int Degre(Monome M) ;
    // degre du monome M

int Coefficient(Monome M) ;
// coefficient du monome M

/* Parcours de la sequence des monomes d'un polynome */

int EstVide(Polynome P) ;
// vaut vrai si et seulement si P est un polynome vide

Monome PremierMonome(Polynome P) ;
// renvoie le "premier" monome du polynome P (si P est non vide !)

void MonomeSuivant(Polynome *P) ;
// modifie P pour acceder au monome suivant (si P est non vide !)
// P est donc un parametre donnee-resultat

/* un parcours des monomes du polynome P pourra donc s'ecrire :
Polynome P ;
Monome M ;
while (! EstVide(P)) {
M = PremierMonome(P) ;
MonomeSuivant(&P) ;
}
*/
```