Langage C
Devoir maison le 14 mai 2020, durée 24h, le début 9h00

Documents autorisés et consignes

Vous devez impérativement signer est joindre à vos copies la déclaration fournie sur moodle qui précise les modalités de devoir. Vous pouvez soit compléter la déclaration, soit faire une copie manuscrite. Après avoir signé, vous faites une photo que vous devez joindre à votre copie sur moodle (ou envoyer par mail).

Le sujet comporte 6 pages. Pour de raison de formatage ou l'insertion de dessin une longue zone blanche à la fin d'une page ne signifie pas la fin du sujet, regardez s'il n'y a pas de page suivante.

Si vous faites les réponses sous forme manuscrite tâchez d'écrire de façon lisible, avec des indentations et des accolades appropriées permettant de voir la fin de blocs de code (fin de boucles, etc.).

Il est inutile d'écrire les `#include` mais si vous avez déjà des includes dans vos fichier `*.c` ne les supprimez pas.

1 Chaînes de caractères

Exercice 1 :

Écrire la fonction

```
1  char *palindrome( const char *s )
```

qui prend en paramètre un pointeur vers un string [1]. On assume que ce string d'une suite de 0 ou plusieurs espaces, suivies d'une suite de lettres, et il termine par une nouvelle suite de 0 ou plusieurs espaces.
En supprimant les espaces au début et à la fin de `s` on obtient un mot, appelons le `y`, composé uniquement de lettres.
Le résultat retourné par `palindrome` est un mot obtenu en concatenant `y` avec son image miroir. Par exemple si `s` est `" aabca "` alors `palindrome` retournera `"aabcaacbaa"`.

1. dans le sens du langage C, donc avec le caractère nul à la fin

2 Listes chaînées

Dans cette sections vous allez manipuler les listes doublement chaînées qui représentent des polynômes à une variable. Nous allons utiliser les structures de données suivantes :

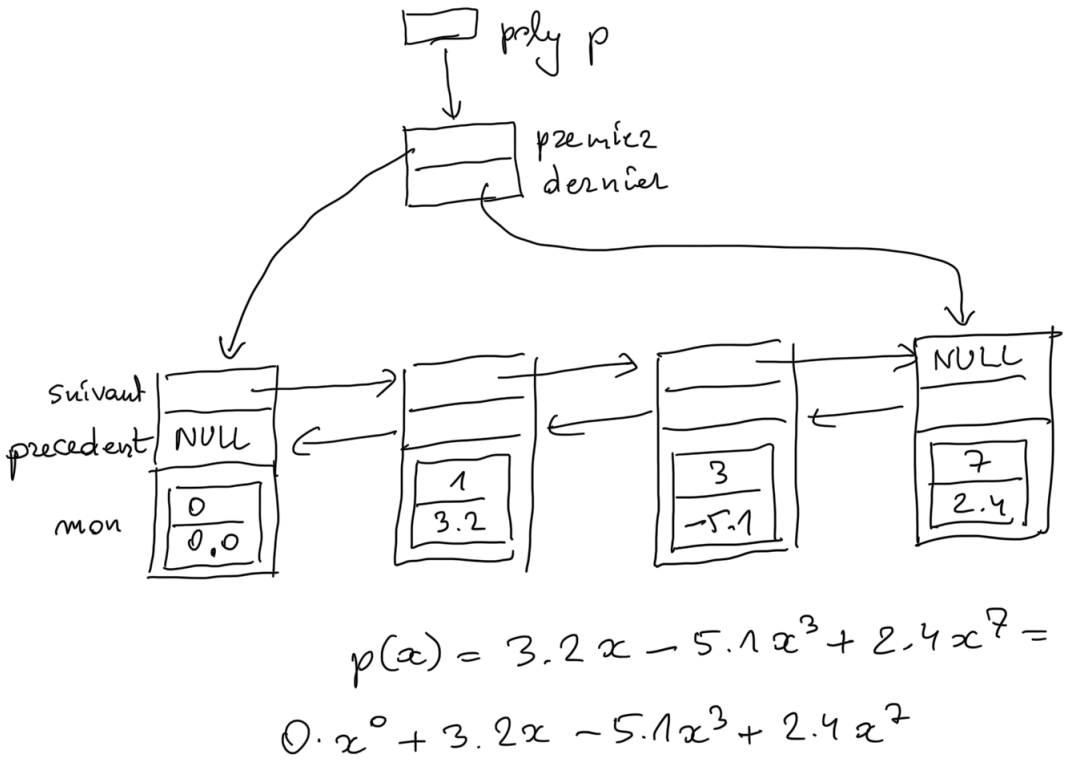
```
typedef struct{
  unsigned degre;
  double coeff;
}monome;

typedef struct elem_liste elem_liste;
typedef elem_liste *mliste;
struct elem_liste{
  mliste suivant;
  mliste precedent;
  monome mon;
};

typedef struct{
  mliste premier;
  mliste dernier;
} spoly;

typedef spoly *poly;
```

La structure `monome` représente un monome. Par exemple si le champ `degre` contient la valeur 5 et le champ `coeff` la valeur -11.2 alors cela correspond au monome $-11.2x^5$.
Un polynôme peut être représenté par une liste de monomes. Les structures `struct elem_liste` représente les éléments de la liste, les pointeurs `suivant` et `precedent` pointent respectivement vers un élément suivant ou précédent de la liste. Le champ `mon` contient le monome stocké dans cet élément de la liste.
La structure `spoly` sert pour stocker les pointeurs vers le premier et le dernier élément de la liste de monomes.
Et finalement `poly` est un pointeur vers la structure `spoly`.
Par exemple le polynôme $p(x) = 3.2x - 5.1x^3 + 2.4x^7$ sera représenté par l'objet de type `poly` suivant :



On suppose que, comme sur le dessin,

- les monomes sur la liste sont rangés dans l'ordre croissant de degrés en commençant par le monome de degré 0 qui dans notre exemple possède le coefficient 0.0.
- la liste de monomes n'est jamais vide et elle commence *toujours* par le monôme de degré 0, ce monome peut éventuellement avoir le coefficient 0,
- tous les monomes de degré supérieur à 0 ont des coefficients différents de zéro. Seul le monome x^0, toujours le premier sur la liste, peut avoir le coefficient 0.0.

Exercice 2 :

Écrire la fonction

```
monome monome_create( double c, unsigned d)
```

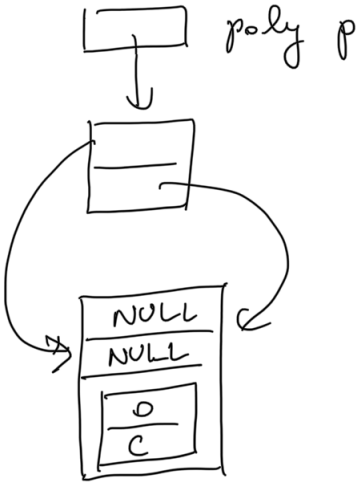
qui retourne un monome de degré `d` et le coefficient `c`.

Exercice 3 :

Écrire la fonction

```
  poly poly_constant( double c )
```

qui construit et retourne un polynôme avec un seul monome cx^0 (voir le dessin ci-dessous).



La fonction retournera NULL si un `malloc` échoue.

Exercice 4 :

Écrire la fonction

```
unsigned poly_degre(poly p)
```

qui retourne le degré de polynôme représenté par `p` (le degré du dernier monome sur la liste de monomes).

Exercice 5 :

Écrire la fonction

```
int poly_add_fort( poly p, monome m)
```

qui ajoute le nouveau monome `m` dans le polynome `p` à condition que `m.degre > poly_degre (p)`.

Plus exactement,

- si `m.degre <= poly_degre(p)` alors la fonction retourne immédiatement 0 pour signaler que le degré de `m` est inapproprié. Aucun nouveau monome ne sera ajouté dans le polynôme représenté par `p`.
- Si `m.degre > poly_degre(p)` alors le nouveau élément contenant le monome `m` sera ajouté comme le dernier élément de la liste de monomes et la fonction retournera 1.
- S'il y a un problème avec la fonction `malloc` alors `poly_add_fort` retournera -1 pour signaler une erreur de `malloc`.

Exercice 6 :

Écrire la fonction

```
  void poly_print( poly p )
```

qui affiche le polynome `p` de façon habituelle. Vous pouvez utiliser dans votre code la fonction

```
void monome_print( monome m ){
  printf("%+3.1fx^%-2u", m.coeff, m.degre);
}
```

qui affiche un monome (et vous pouvez copier `monome_print` dans votre code). Vous devez afficher le monome x^0 avec son coefficient seulement si ce coefficient est différent de zéro ou quand `p` contient un seul monome $0.0x^0$ c'est-à-dire le polynôme représenté par `p` est le polynôme constant $p(x) = 0$.

Exercice 7 :

Écrire la fonction

```
int poly_add_monome( poly p, monome m )
```

qui fait l'addition de polynôme `p` et monome `m`.
Si le coefficient de `m` est 0 la fonction retourne immédiatement 1.
Si le degré de `m` est plus grand que le degré de `p` servez vous de la fonction `poly_add_fort` de l'exercice 5.
Dans le cas contraire il y a deux cas.

(1) Soit le polynôme `p` ne contient pas d'élément avec le monome de même degré `m.degre`. Dans ce cas il faut insérer un nouveau élément contenant `m` dans la liste de monomes. L'insertion doit respecter l'ordre de monomes par degré croissant. Si l'insertion réussi la fonction retourne 1. Si un problème de `malloc` on retourne -1.

(2) Le polynome `p` contient un monome `q` tel que `q.degre == m.degre`. Dans ce cas il n'y a pas d'insertion de nouveau élément dans la liste. Par contre le monome `q` de la liste aura le nouveau coefficient : `q.coeff + m.coeff`. Si suite à cette addition le nouveau coefficient de `q` est zéro et `q.degre != 0` alors vous devez supprimer l'élément qui contient `q` de la liste [2]. Et finalement `poly_add_monome` retournera 1.

Exercice 8 :

Écrire la fonction

```
   double *poly2vect(poly p)

```

qui retourne le pointeur vers le vecteur de coefficients du polynome `p`. Le $i$ème élément du vecteur sera le coefficient de x^i. Notons que le vecteur retourné contient aussi tous les coefficients 0. Par exemple pour le polynôme $p(x) = 3.1 + 4.9x^2 - 5.6x^4$ la fonction doit retourner le vecteur avec les nombres `3.1, 0.0, 4.9, 0.0, -5.6`.

2. rappelons que tous les coefficients sur la liste doivent être différents de zéro sauf peut-être le coefficient de x^0

3 vecteurs et structures

Soit

```
typedef struct{
  double x;
  double y;
} point;
```

un structure représentant un point sur la surface.

Exercice 9 :

Écrire la fonction

```
point *inserer_point( unsigned *len, point chemin[], point p, unsigned pos )
```

qui prend un vecteur `chemin` de `point`s. Le paramètre `len` contient l'adresse d'un nombre `unsigned int`, c'est le nombre d'éléments dans le vecteur `chemin`. Le nouveau point `p` doit être inséré dans le vecteur `chemin` à la position `pos`. Les positions autorisées satisfont la condition `0 <= pos <= *len`. Si la condition n'est pas satisfaite alors `inserer_point` retourne immédiatement la valeur `chemin` et le chemin n'est pas modifié.
Sinon le nouveau point est inséré, ce qui implique l'augmentation de la taille de vecteur pointé par `chemin`, et déplacement de points à partir de la position `pos`. La fonction retourne le vecteur de point avec `p` inséré et la valeur de `*len` doit être incrémenté de 1 (`len` est aussi bien le paramètre d'entrée que de sortie).
Par exemple si `chemin` contient 4 points `a`, `b`, `c`, `d` et on inséré le point `x` à la position (l'indice) 2 alors on obtient le vecteur avec le points `a`, `b`, `x`, `c`, `d`.
On privilégie, et de loin, une solution qui n'utilise aucune boucles.

Le cas particulier : Il est autorisé d'avoir `chemin == NULL`, `*len == 0` et `pos == 0`. Dans ce cas on construit un nouveau vecteur qui contient un seul point `p`.
La fonction retournera NULL en cas d'échec d'allocation mémoire.