Travaux Pratiques de programmation n°1

Cours de Programmation C

—L2.1—

▸ **Exercice 1.** *La permutation de Josephus*(n, p)

n personnes se placent en cercle,
on compte p personnes et on élimine la $p^{ième}$*, puis la* $p^{ième}$ *après celle-ci .... On continue tant que toutes les personnes n'ont pas été eliminées (ce sera toi le chat, mais ...)*
La permutation de Josephus est la suite des numéros des personnes éliminées.
La permutation Josephus(6,4) est : 4, 2, 1, 3, 6, 5.
Pour déterminer cette permutation, on crée une liste simplement chaînée circulaire contenant les entiers de 1 *à* n*, puis on élimine les cellules une à une. On utilise les types*

```
typedef struct cel{
int valeur;
struct cel *suivant;
}Cellule,*Liste;
```

1. *Écrire une fonction d'allocation d'une* `Cellule` *qui reçoit une valeur* n *et renvoie, si possible, l'adresse d'une* `Cellule` *contenant la valeur* n *et formant une liste circulaire à un seul élément. En cas d'échec la fonction renvoie* `NULL`
2. *Ecrire une fonction* `int AjoutCirculaire(Liste *l, int val)` *qui ajoute une nouvelle cellule contenant la valeur* `val` *à la liste circulaire. La fonction renvoie 0 en cas d'echec d'allocation, 1 sinon.*
3. *Ecrire une fonction* `int CréationCercle(Liste *l, int nombre)` *qui forme une liste circulaire avec les entiers de 1 à* `nombre`.
4. *Ecrire une fonction* `Liste ExtraitPieme(Liste *l,int p)` *qui extrait la cellule située* `p` *cellules après la cellule de tête. La cellule qui suit celle qui a été supprimée devient la nouvelle cellule de tête.*
5. *Ecrire une fonction* `void Affiche Josephus(int n, int p)` *qui affiche la permutation de Josephus(n,p). La fonction crée une liste circulaire de* n *cellules puis la détruit en supprimant une cellule toutes les* p *cellules.*

▸ **Exercice 2.** *Polynômes*

Dans toutes les réponses, une attention particulière sera donnée à la gestion de la mémoire.
On décide de représenter des polynômes à coefficients entiers à une variable par des listes simplement chaînées de monômes, chaque monôme étant représenté par une structure constituée d'un entier `coef` *pour le coefficient du monôme, d'un entier* `degre` *pour le degré*

du monôme, et d'un pointeur sur le monôme suivant. Un polynôme ne contient pas 2 monômes de même degré.
On ordonne les monômes par degré croissant.

1. *Définir les types* `Monome` *et* `Polynome`
2. *Écrire une fonction* `Monome* CreerMonome(int coef,int degre)` *qui effectue la création du monome* $coef\ x^{degre}$
3. *Écrire une fonction* `void LiberePolynome(Polynome *P)` *qui libère tous les monômes du polynôme* `*P` *et réinitialise le pointeur à* `NULL`.
4. *Écrire une fonction* `void AfficheListe(Polynome P)` *qui affiche les monômes d'un polynôme sous forme de couple* `(coeff, degre)`.
5. *Écrire une fonction* `int AjouteMonome(Monome *m,Polynome *p)` *qui ajoute le monôme* `*m` *au polynôme* `*p`. *Un polynôme ne doit pas avoir deux monômes avec le même degre, cependant, on ne traitera pas pour l'instant le cas des monôme à coefficient nul*
6. *Écrire une fonction* `Polynome LirePolynome(Polynome *p)` *qui crée un polynôme par lecture successive de ses monômes. La lecture s'arrête à la saisie d'un degré négatif (ou en cas de problème de mémoire). Le polynôme donné en exemple pourrait avoir été saisi par :*

```
Donner les monomes sous forme degre coefficient:
0 12
4 2
1 1
13 -5
4 1
-1
```

7. *Ecrire une fonction* `void Nettoie(Polynome *P)` *qui supprime du polynome les monomes de coefficients 0, sauf si le polynome est le polynome nul representé par le monôme de coefficient 0 et de degre' 0*
8. *Modifier la fonction* `LirePolynome` *pour que le polynôme renvoyé corresponde bien au formatage ci-dessus (gestion du polynôme nul)*
9. *Écrire une fonction* `AffichePolynome( Polynome P)` *qui affiche le polynôme P sur une ligne. On représentera la puissance de x par ^ . Le polynôme précédent est affiché sous la forme :*
   `12+3x^4-5x^13`
10. *Reprendre les fonctions* `SommePolynome` *et* `ProduitPolynome` *écrites en TD et les implanter.*