ULCO **2017 - 2018**
Licence 2 **Mathématiques**

Examen – Informatique

Documents autorisés, pas de livre, pas de calculatrice
Vendredi 19 janvier 2018
Durée : 2h

Exercice 1. [3 points] On se propose de représenter les nombres complexes à l'aide de la structure

```
struct Complexe{
   float a,b;
};
```

correspondant au nombre complexe $\texttt{a} + \mathbf{i}\texttt{b}$.

1. **[0.5 pt]** Ecrire une fonction `bool est_reel(Complexe z)` qui teste si le nombre complexe `z` est un réel.
2. **[1 pt]** Ecrire une fonction `conjugue` qui prend en paramètre un nombre complexe `z` et qui retourne le conjugué $\overline{\texttt{z}}$ du nombre complexe `z`.
3. **[1.5 pt]** Ecrire une fonction `addition` qui prend en paramètres deux nombres complexes `z1` et `z2` et retourne le nombre complexe $\texttt{z1} + \texttt{z2}$.

Exercice 2. [4 points] On rappelle que pour des entiers n et k vérifiant $0 \leqslant k \leqslant n$, le coefficient binomial $\binom{n}{k}$ satisfait la relation de récurrence :

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & \text{si } 0 < k < n \\ 1 & \text{si } k = 0 \text{ ou } k = n \end{cases} \tag{1}$$

1. **[0.5 pt]** A partir de (1), calculer à la main tous les coefficients binomiaux pour $n \in \{0, 1, 2, 3, 4\}$.
2. **[1 pt]** Ecrire une fonction récursive `int binomial(int n,int k)` basée sur la récurrence (1) et retournant le coefficient binomial $\binom{n}{k}$.

Le triangle de Pascal est une présentation particulière des coefficients binomiaux :

$$\begin{array}{ccccccc} & & & \binom{0}{0} & & & \\ & & \binom{1}{0} & & \binom{1}{1} & & \\ & \binom{2}{0} & & \binom{2}{1} & & \binom{2}{2} & \\ \binom{3}{0} & & \binom{3}{1} & & \binom{3}{2} & & \binom{3}{3} \end{array} \tag{2}$$

Le niveau n du triangle de Pascal correspond à tous les coefficients binomiaux $\binom{n}{k}$ pour k prenant successivement les valeurs $0, 1, ..., n$. Pour la suite, on note T_n le tableau correspondant au niveau n du triangle de Pascal : $T_n = [\binom{n}{0}, \binom{n}{1}, ..., \binom{n}{n}]$.

3. **[0.5 pt]** Donner les tableaux T_n pour $n \in \{0, 1, 2, 3, 4\}$. Quelle est la taille de T_n pour $n \geqslant 0$?
4. **[2 pts]** Le but de cette question est d'écrire une fonction récursive `tableau<int> pascal_rec(int n)` retournant le tableau T_n pour $n \geqslant 0$. Dans le cas $n > 0$, on calculera le tableau T_{n-1} avec un appel correct à `pascal_rec` (récursivité) permettant ensuite de déterminer les coefficients de T_n. On utilisera la fonction `tableau<int> creer_tableau(int n)` initialisant à 0 et retournant un tableau de taille n.
5. **[3 pts bonus]** Proposer une fonction `tableau<int> pascal(int n)` non récursive retournant le tableau T_n, ne faisant pas appel aux factorielles ni aux fonctions du **2.** et du **4.** et ne faisant appel qu'une seule fois à la fonction `creer_tableau`.

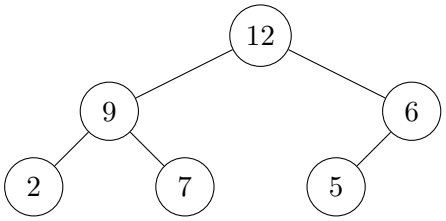
Exercice 3. [7 points]
On considère des listes chaînées d'entiers données par les structures

```
struct Element{
    Element* tete;
    int valeur;
  };
```

```
struct Liste{
    Element* tete
  };
```

1. **[1 pt]** Ecrire une fonction `bool meme_longueur(Liste l1,Liste l2)` qui teste si les deux listes chaînées `L1` et `L2` sont de même longueur.
2. **[2 pts]** Ecrire une fonction `Liste somme(Liste l1,Liste l2)` qui étant données deux listes chaînées `L1` et `L2` retourne la liste chaînée obtenue en faisant la somme des coefficients de `L1` et `L2` élément à élément si `L1` et `L2` sont de même longueur et la liste vide sinon. On pourra utiliser la fonction `Element* creer_element()` qui crée un nouvel élément et retourne un pointeur sur celui-ci.
   Exemple : Pour `L1 : 1 -> 2 -> 3 -> 4` et `L2 : 2 -> 2 -> 3 -> 1`, la fonction `somme` retournera `3 -> 4 -> 6 -> 5`.
3. **[2 pts]**Ecrire une fonction `bool est_triee(Liste l)` qui teste si la liste `l` est triée pour l'ordre croissant.
4. **[2 pts]** Ecrire une fonction `Liste fusion(Liste l1,Liste l2)` qui étant donnée deux listes triées `l1` et `l2` retourne une liste triée obtenue en faisant la fusion des listes `l1` et `l2` (semblable à la fusion de deux tableaux pour le tri fusion). On pourra encore utiliser la fonction `Element* creer_element()`.

Exercice 4. [6 points]
On note T l'arbre binaire d'entier suivant



1. **[1 pt]** Donner l'ordre de visite des sommets de T lors des parcours préfixé, infixé et postfixé.

On représente les arbres binaires d'entiers à l'aide de la structure suivante

```
struct Arbre{
  int valeur;
  Arbre* droite;
  Arbre* gauche;
};
```

2. **[1 pt]** Ecrire une fonction `int feuilles(Arbre* A)` qui retourne le nombre de feuilles de l'arbre `A`.

On dit qu'un arbre binaire `A` est un tas si en tout noeud x de `A`, les valeurs des fils gauche et droite de x sont plus petites que la valeur en x. L'arbre `T` est un tas mais si on replace la valeur 2 par 10, ce n'est plus un tas.

3. **[2 pts]** Ecrire une fonction `bool est_tas(Arbre* A)` qui teste si l'arbre `A` est un tas.
4. **[0.5 pt]** Si l'arbre `A` est un tas non vide, où se trouve la plus grande valeur de `A` ? Et la seconde plus grande valeur ?
5. **[1.5 pts]** Ecrire une fonction affichant la plus petite et la plus grande valeur d'un tas non vide.