ULCO **2018 - 2019**
Licence 2 **Mathématiques**

Examen – Informatique

Documents autorisés, pas de livre, pas de calculatrice
Jeudi 17 janvier 2019
Durée : 2h

Exercice 1. [3 points]
Ecrire une procédure `void selection_rec(array<int,100> tab, int debut, int fin)` qui trie le tableau `tab` entre `debut` et `fin` à l'aide d'une version récursive du tri par sélection.

Exercice 2. [5 points]
On se propose de représenter les fruits et légumes d'un maraîcher à l'aide d'une matrice `Caract` de taille 2xN. Cette matrice contiendra dans les indices croissants des colonnes à partir de 0 les caractéristiques (poids en kg, prix du kg en euros) des différents fruits. Dans les indices décroissants des colonnes à partir de N-1 elle contiendra les caractéristiques (poids, prix) des différents légumes. On gardera en mémoire le nombre de types de fruits `NbFruits` ainsi que celui de légumes `NbLegumes`. Le nombre total de tous ces types (fruits et légumes) ne devra pas être supérieur à N (on posera N=100). Toutes ces données seront rassemblées dans une structure `FruitsLegumes` comme suit :

```
int const N=100;

struct FruitsLegumes{
       array < array<float, N>, 2> Caract;
       int NbFruits, NbLegumes;
};
```

Exemple : Supposons qu'on ait des fruits et légumes avec les caractéristiques suivantes

3 types de fruits :

bananes	30kg	2.52€/kg
poires	50kg	3.14€/kg
pommes	100kg	1.5€/kg

4 types de légumes :

carottes	60kg	1.82€/kg
poireaux	20kg	5€/kg
pommes de terre	200kg	1.2€/kg
navets	20kg	3.63€/kg

La structure sera alors :

— Le tableau `Caract`

	bananes	poires	pommes				navets	pdt	poireaux	carottes
poids	30	50	100				20	200	20	60
prix/kg	2.52	3.14	1.5				3.63	1.2	5	1.82

— `NbFruits` = 3
— `NbLegumes` = 4

Si on ajoute un fruit, on ajoutera ses caractéristiques après celles des pommes dans le tableau. Si on ajoute un légume, on ajoutera ses caractéristiques avant celles des navets dans le tableau.

1. **[1pt]** Ecrire une fonction `bool ajouterLegume(FruitsLegumes FL, float pds, float px)` qui ajoute dans la structure `FL` un légume avec le poids `pds` et le prix `px`, et qui met à jour les différents éléments de cette structure `FL`, et qui renvoie `true` si l'ajout a pu avoir lieu et `false` sinon.

2. **[2pt]** Ecrire une fonction `bool ajouterFruit(FruitsLegumes FL, float pds, float px)` qui ajoute dans la structure `FL` un fruit avec le poids `pds` et le prix `px`, et qui met à jour les différents éléments de cette structure `FL`, et qui renvoie `true` si l'ajout a pu avoir lieu et `false` sinon.
3. **[2pt]** Ecrire une fonction `float calculerCA(FruitsLegumes FL)` qui calcule et renvoie le chiffre d'affaires du maraîcher s'il vend tous les fruits et légumes qu'il a mentionnés dans les données de la structure `FL`. Ce calcul consiste à faire la somme de la multiplication du poids par le prix au kg pour chaque article.

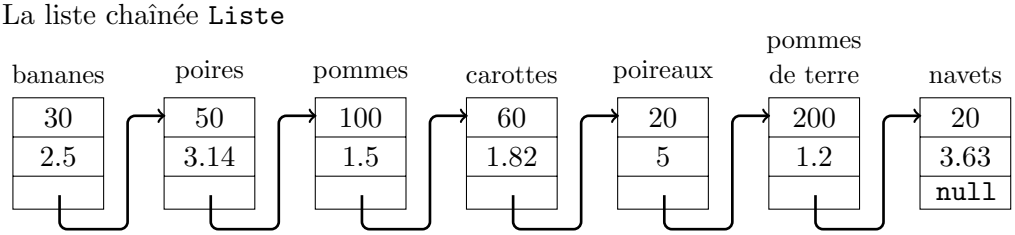
Exercice 3. [**5 points**] Vous allez retravailler avec les mêmes données mais avec une liste chaînée à la place de la matrice. Du coup les caractéristiques de chaque légume seront enregistrées dans l'ordre d'arrivée (voir exemple ci-dessous). Votre structure `FruitsLegumes` se transformera en `FruitsLegumesListe` et sera décrite comme suit :

```
struct Caract{
  float poids;
  float prix;
  Caract* svt;
};

struct FruitsLegumesListe{
       Caract* Liste;
       int NbFruits, NbLegumes;
};
```

Exemple : Avec les caractéristiques de l'exemple de l'exercice 1, la structure `FruitsLegumesListe` sera :

— La liste chaînée `Liste`



— `NbFruits` = 3

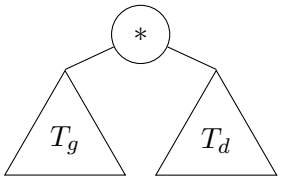
— `NbLegumes` = 4

Si on ajoute un fruit, on ajoutera ses caractéristiques après celles des `pommes` et avant celles des `carottes` dans la liste.
Si on ajoute un légume, on ajoutera ses caractéristiques après celles des `navets`, soit en fin de liste.

Dans les questions suivantes on supposera que la liste `FLL` contient au moins un fruit.

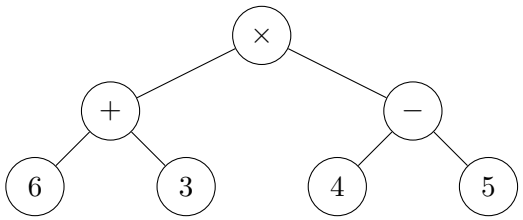
1. **[1pt]** Ecrire une fonction `void ajouterFruit(FruitsLegumesListe FLL, float pds, float px)` qui ajoute dans la structure `FLL` un fruit avec le poids `pds` et le prix `px` et qui met à jour les différents éléments de cette structure `FLL`.
2. **[1pt]** Ecrire une fonction `float calculerCA(FruitsLegumesListe FLL)` qui calcule le chiffre d'affaires du maraîcher s'il vend tous les fruits et légumes qu'il a mentionnés dans les données de la structure `FLL`.
3. **[3pt]** Ecrire une fonction `void ajouterLegume(FruitsLegumesListe FLL, float pds, float px)` qui ajoute dans la structure `FLL` un légume avec le poids `pds` et le prix `px` et qui met à jour les différents éléments de cette structure `FLL`.

Exercice 4. [9 points] On note $\mathcal{A}$ l'ensemble des expressions algébriques valides formées à partir d'entiers et des opérations binaires $+$, $-$ et $\times$. Dans cet exercice on se propose de représenter les expressions de $\mathcal{A}$ à l'aide d'arbres binaires. Si $*$ est l'une des opérations $+, -$ ou $\times$, et g, d deux éléments de $\mathcal{A}$ alors on représente l'expression $g * d$ de $\mathcal{A}$ par l'arbre binaire :

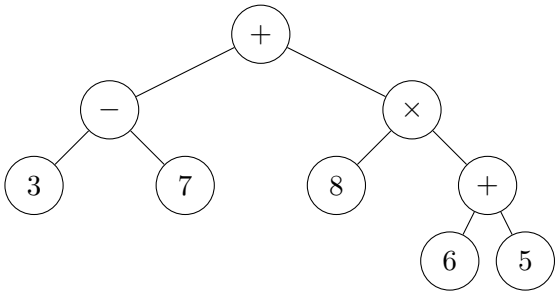


où T_g et T_d sont les arbres associés à g et d respectivement.

Exemple : l'expression $(6+3) \times (4-5)$ est représentée par l'arbre binaire T_1



1. **[1pt]** Donner l'expression algébrique représentée par l'arbre suivant T_2 :



2. **[1pt]** Où se trouvent les nombres d'un arbre binaire représentant une expression de $\mathcal{A}$? Et les opérations binaires ?

Nous proposons la structure `ArbreAlg` suivante pour les arbres binaires d'expression de $\mathcal{A}$.

```
struct ArbreAlg{
   ArbreAlg* gauche;
   ArbreAlg* droite;
   bool est_nombre;
   int valeur;
};
```

La variable `est_nombre` vaut `true` si le noeud contient un nombre entier et `false` s'il contient une opération binaire. Si `est_nombre` vaut `true` alors `valeur` contient le nombre associé au nœud. Si `est_nombre` vaut `false` alors `valeur` contient 0, 1 ou 2 si le noeud contient $+$, $-$ ou $\times$ respectivement.

3. **[1pt]** Réécrire l'arbre T_2 à l'aide de la représentation précédente, en précisant pour chaque noeud la valeur des variables `est_nombre` et `valeur`.
4. **[1pt]** Ecrire la valeur de chaque nœud de l'arbre précédemment obtenu pour un parcours infixé. De même pour un parcours préfixé.
5. **[3pt]** Ecrire une procédure `bool est_valide(ArbreAlg* T)` qui teste si l'arbre `T` représente une expression algébrique valide. On testera les critères obtenus en **2.**
6. **[2pt]** Ecrire une procédure `int evalue(ArbreAlg* T)` qui retourne l'entier donné par l'expression algébrique représentée par un arbre valide `T`. Par exemple `evalue`(T_1) et `evalue`(T_2) retourneront -9 et 84 respectivement.