programmation

examen

Un système de Lindenmayer (ou L-Système) peut être utilisé pour dessiner des fractales. Par exemple, le système ci-contre permet de dessiner la fractale de Hilbert :

```
Hilbert {
   Angle 4
   Axiome X
   X=-YF+XFX+FY-
   Y=+XF-YFY-FX+
}
```

Angle spécifie un nombre entier *a*, qui permet de produire l'angle 360/*a*, qui sert dans les rotations à gauche(+) ou à droite (-) de la plume.

Les règles sont des règles de réécriture. Un système de Lindenmayer fait des dérivations jusqu'à un certain rang *n* à partir de l'axiome. Une réécriture consiste à remplacer tout symbole de la chaîne à réécrire par :

- la partie droite de la règle dont il est partie gauche, si cette règle existe.
- lui-même s'il ne se trouve pas en partie gauche de règle.

Une dérivation au rang 1 du système Hilbert produira la chaîne `-YF+XFX+FY-` et au rang 2 la chaîne : `-+XF-YFY-FX+F+-YF+XFX+FY-F-YF+XFX+FY-+F+XF-YFY-FX+-`

Une chaîne obtenue par dérivation peut être interprétée pour tracer un dessin avec la plume. Les symboles prédéfinis sont interprétés pour tracer un courbe, les autre symboles sont ignorés :

- 'F' la plume baissée avance d'une distance *d*
- 'G' la plume levée avance d'une distance *d*
- '+' la plume tourne à gauche de l'angle 360/*a, a* est la valeur qui suit '*Angle'* dans la définition du système.
- '-' la plume tourne à droite de l'angle 360/*a*
- '[' on empile l'état de la plume
- ']' on dépile l'état de la plume

Nous utiliserons la hiérarchie de classes suivante pour représenter les symboles :

```
class Symbole{
protected :
    vector <Symbole *> _reEcrit;  // liste de symboles partie droite de
                                  // règle
public :
    Symbole(){_reEcrit.push_back(this);};
    virtual void interprete()=0;
    virtual AnsiString chaine()=0;
    virtual void setReEcrit(vector <Symbole *> & r ){ _reEcrit=r;};
    vector <Symbole *> getReEcrit(){ return _reEcrit; }
    bool enPartieGauche(){
      return _reEcrit.empty()||_reEcrit.size()!=1||_reEcrit.begin()!=this;
    }
};
```

Le vecteur *_reEcrit* contient la liste des symboles (éventuellement vide) en partie droite de la règle dont le symbole est partie gauche si elle existe, et si le symbole ne se trouve pas en partie gauche de règle, le vecteur *_reEcrit* ne contient qu'un pointeur sur le symbole lui-même.

```
class F : public Symbole{
public :
    void interprete();
    AnsiString chaine();
};
```

```
class CrochetO : public Symbole{
public :
    void interprete();
    AnsiString chaine();
};
```

```
class G : public Symbole{
public :
    void interprete();
    AnsiString chaine();
};
```

```
class CrochetF : public Symbole{
public :
    void interprete();
    AnsiString chaine();
};
```

```
class Plus : public Symbole{
public :
    void interprete();
    AnsiString chaine();
};
```

```
class Auxiliaire : public Symbole{
    char nom;
public :
    Auxiliaire (char c):nom(c){}
    AnsiString chaine();
    void interprete();
};
```

```
class Moins : public Symbole{
public :
    void interprete();
    AnsiString chaine();
};
```

Question 1. (3 points)

Ecrire les définitions des fonctions *AnsiString chaine()* pour les différentes classes. La fonction *chaine* renvoie le symbole sous forme de chaîne de caractères.

Question 2. (3 points)

Ecrire les définitions des fonctions *void interprete()*pour les différentes classes. Les fonctions *interprete* utilisent les variables globales *plume*, *d* et *a* pour effectuer l'action correspondant au symbole.

Pour représenter un L-Système nous définissons la classe :

```
class LSysteme{
private :
     int angle;
     AnsiString nom;
     vector<Symbole *> axiome;
public :
     LSysteme();
     LSysteme(int a, AnsiString n, vector<Symbole *> ax);
     LSysteme(const Lsysteme &ls);
     ~LSysteme();
     AnsiString derivation (int n);
     void interprete(int n);
};
```

Question 3. (4 points)

Définir les constructeurs et destructeur de la classe *Lsysteme*.

Question 4. (5 points)

Définir la fonction *AnsiString derivation(int n)* qui calcule la chaîne de caractère obtenue par *n* dérivations à partir de l'axiome.

Question 5. (5 points)

Définir la fonction *void interprete(int n)* qui interprète la liste des actions à effectuer lors d'une dérivation de profondeur *n*, sans calculer explicitement cette liste d'actions.