Vers, strophes et poésies

Travaux dirigés et pratiques - Programmation par objets 2 (UE ULIN606)

Objectifs Lors de ces exercices, nous réviserons les notions suivantes :
- classe, instances, méthodes
- passages de paramètres, tableaux, pointeurs
- définition d'opérateurs

Le sujet dépasse volontairement ce qui peut être fait pendant 1h30 de travaux dirigés et vous permettra de vous exercer au-delà de la séance.

1 Les classes `Vers` et `Strophe`

Nous reprenons l'exemple vu en cours et qui porte sur la représentation des vers et des strophes dans une perspective métrique (étude des régularités de rime et de rythme dans la poésie).

Dans cet exemple, un *vers* est essentiellement une suite de mots à laquelle on peut attacher une rime, la sonorité terminale (écrite sous une forme normalisée). Un vers peut être saisi et affiché (voir la Figure 1). Aux deux attributs `suiteMots` et `rime` sont associées deux opérations particulières, habituellement appelées des accesseurs car elles se spécialisent dans l'accès en lecture (ex. `getSuiteMots`) ou en écriture (ex. `setSuiteMots`).

Une *strophe* est une suite de vers. Dans le cas général (qui admet la poésie en vers libres) nous admettrons que l'on peut saisir une strophe par saisie du nombre de vers puis saisie successive des différents vers (dans l'ordre d'apparition dans la strophe). L'affichage d'une strophe est l'affichage de ses vers successifs.

L'implémentation de la classe `Strophe` (en particulier de son destructeur) s'appuie sur l'hypothèse faite en cours selon laquelle plusieurs strophes peuvent partager des vers (sémantique de l'agrégation en UML).

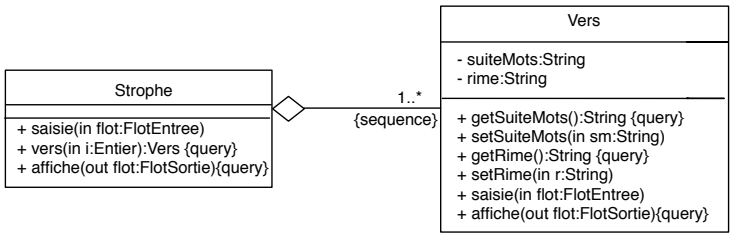

Fig. 1 – Eléments de la représentation des strophes

```
#ifndef vers_h
#define vers_h
class Vers
{
private:
  string suiteMots;     // suiteMots, attribut de type string
  string rime;               // rime, attribut de type string
public:
  Vers();
  Vers(string s);
```

```
  Vers(string s, string r);
  virtual ~Vers();
  virtual string getSuiteMots()const;
  virtual void setSuiteMots(string sm);
  virtual string getRime()const;
  virtual void setRime(string r);
  virtual void saisie(istream& is);
  virtual void affiche(ostream& os)const;
};
#endif

#include<iostream>
#include<string>
using namespace std;
#include"Vers.h"

Vers::Vers(){}
Vers::Vers(string sm){suiteMots=sm;}
Vers::Vers(string sm, string r){suiteMots=sm;rime=r;}
Vers::~Vers(){}

string Vers::getSuiteMots()const
{return suiteMots;}
void Vers::setSuiteMots(string sm)
{suiteMots=sm;}
string Vers::getRime()const
{return rime;}
void Vers::setRime(string r)
{rime=r;}

void Vers::saisie(istream& is)
{cout <<"vers puis rime" <<endl;is>>suiteMots>>rime;}

void Vers::affiche(ostream& os)
{os<<"<<"<<suiteMots<<">>";}

#ifndef Strophe_h
#define Strophe_h
class Strophe
{
private:
  Vers ** suiteVers;  // suiteVers, attribut de type tableau de pointeurs vers des Vers
                           // implémente l'agrégation "une strophe se compose de vers"
  int nbVers;
public:
  Strophe();
  virtual ~Strophe();
  virtual void saisie(istream& is);
  virtual Vers* const vers(int i)const;
  virtual void affiche(ostream& os)const;
};
#endif

#include<iostream>
#include<string>
using namespace std;
#include"Vers.h"
#include"Strophe.h"

Strophe::Strophe(){suiteVers=NULL; nbVers=0;}
Strophe::~Strophe(){if (suiteVers) delete[] suiteVers;}
```

```
Vers* Strophe::vers(int i)const
{if (i>=0 && i<nbVers) return suiteVers[i]; else return NULL;}

void Strophe::saisie(istream& is)
{
  if (suiteVers) delete[] suiteVers;
  cout << "Entrer le nombre de vers : " << endl;
  is>>nbVers; suiteVers = new Vers*[nbVers];
  for (int i=0; i<nbVers; i++)
    {
      Vers *v=new Vers(); v->saisie(is); suiteVers[i]=v;
    }
}
void Strophe::affiche(ostream& os)const
{
  for (int i=0; i<nbVers; i++)
    {
      suiteVers[i]->affiche(os); os << endl;
    }
}
```

2 Définition d'opérateurs

Pour définir un opérateur (par exemple +), on définit soit une fonction, soit une méthode nommée `operator`*symboledel'operateur* (par exemple `operator+`). Tous les opérateurs sont surchargeables sauf : `. .* :: ?: # ##`. On ne peut changer ni le nombre d'opérandes, ni la priorité, ni le sens d'associativité (droite à gauche, comme `=`, ou gauche à droite comme les opérateurs arithmétiques). Les opérateurs `=,[],()`, `->` doivent obligatoirement être écrits comme des méthodes. Les autres opérateurs peuvent être écrits sous forme de méthode ou de fonction, mais si on choisit de faire une méthode, elle doit obligatoirement appartenir à la classe du premier opérande.

Pour déterminer la signature, on peut se représenter l'appel sous une forme préfixe de l'opérateur. Par exemple, si on a défini dans une classe A la méthode `bool operator== (const A&)`, `x==y` équivaut à `x.operator==(y)`. De même, si on a défini la fonction `operator+(const A& a, const A& b)`, `x+y` équivaut à `operator+(x,y)`.

Question 1 *Ajouter une définition des opérateurs << et >> aux deux classes* ***`Vers`*** *et* ***`Strophe`*** *afin d'insérer ou d'extraire les objets de ces classes dans des flots (ou dit plus familièrement afin d'afficher ou de saisir ces objets).*

Question 2 *Ajouter à la classe* ***`Strophe`*** *une définition de l'opérateur = ayant la même sémantique que le constructeur par copie de manière à rendre la définition de la classe plus cohérente. Cela peut vous inspirer une réécriture du constructeur par copie.*

Question 3 *Ajouter à la classe* ***`Strophe`*** *une définition de l'opérateur [] afin de donner accès au pointeur du vers d'un certain rang. Cette définition doit permettre d'écrire les instructions suivantes :*

```
Strophe S;
S.saisie(cin); // hypothèse : saisie de 8 vers
cout << *S[0] ; // affiche le premier vers
S[1]=new Vers("comme je descendais des fleuves impassibles","ible");
       // modifie le deuxième vers
```

Proposez ensuite les instructions ci-dessus dans le cas où ***`S`*** *est de type* ***`Strophe*`*** *(pointeur vers une strophe).*

3 Définition de la classe `Poesie`

Question 4 *Une poésie est une suite ordonnée de strophes que l'on peut classer dans un genre (ballade, sonnet, libre, etc.). Proposez un schéma UML et une implémentation en C++ de cette classe.*

4 Manipulation de tableaux

Nous désirons à présent ajouter une méthode vérifiant qu'une strophe correspond à un schéma rimique passé en paramètre. Le schéma rimique est une chaîne de caractères qui exprime les équivalences en rimes dans la strophe et impose le nombre de vers. Par exemple dans un quatrain croisé le schéma rimique est `ABAB`, qui exprime le fait que le premier et le troisième (resp. le deuxième et le quatrième) vers riment ensemble. Nous ne ferons aucune hypothèse sur les schémas rimiques, qui sont donc simplement des suites finies de caractères contenant des répétitions.

La strophe ci-dessous est une strophe de schéma rimique `ABAB`.

Deux fois je regarde ma montre,
Et deux fois à mes yeux distraits
L'aiguille au même endroit se montre :
Il est une heure... Une heure après.
T. Gautier, La montre, « Emaux et Camées »

Celle qui suit a pour schéma `AABCCCB`.

Et voici venir, dans les prés,
Les fiers coquelicots, parés
De leur pourpre cardinalice ;
Voici les jacinthes mouvant
Leurs cloches roses dans le vent,
Et les tulipes, élevant
Le saint calice !
J. Rameau, Procession des fleurs, « Nature »

Pour faciliter la vérification, nous introduisons une structure intermédiaire qui transcrit le schéma rimique sous une forme plus facile à utiliser. Cette structure est un tableau d'entiers de même taille que le schéma rimique. Une case i de ce tableau contient -1 lorsque la rime du ième vers apparaît pour la première fois dans la strophe, sinon elle contient l'indice j du vers contenant la première occurrence de la rime (voir Figure 2). Il est fortement conseillé d'écrire une méthode ou une fonction auxiliaire qui construit cette structure intermédiaire, puis de se servir de cette structure dans la vérification, plutôt que du schéma rimique. Expliquez le choix de la protection que vous placez sur la méthode de calcul auxiliaire (private ou public).

numero de vers	schema rimique	structure auxiliaire
0	A	–1
1	A	0
2	B	–1
3	C	–1
4	C	3
5	C	3
6	B	2

FIG. 2 – Schéma rimique et structure auxiliaire pour la strophe de J. Rameau

Question 5 *Ecrire la méthode vérifiant qu'une strophe correspond à un schéma rimique passé en paramètre.*