Centre Informatique pour les Lettres et les Sciences Humaines

Apprendre C++ avec QtCreator
Etape 7 : Classes

Lors de l'étape 6, nous avons rencontré une fonction dont l'appel exigeait une syntaxe particulière : la fonction `length()`, qui ne peut être exécutée qu'*au titre d*'un objet de type `std::string`. Cette syntaxe est liée au fait que `std::string` n'est pas un type prédéfini, ce qui nous amène à poser la question "Comment peut-on définir un type ?".

1 - Notion de classe

Lorsque les types prédéfinis (`int`, `char`, `bool`, `double`...) s'avèrent trop simples pour représenter efficacement les données manipulées par un programme, le langage offre la possibilité de créer des types plus sophistiqués.

Certains de ces types sont d'un usage assez général (ils seront utilisés dans de nombreux programmes) et sont mis au point par des spécialistes qui les distribuent aux programmeurs intéressés sous la forme de librairies (Qt est un exemple d'une telle librairie). D'autres sont plus spécifiques à un projet particulier et ne seront définis qu'à l'intérieur du programme correspondant. Dans un cas comme dans l'autre, le langage C++ permet de définir ces types en créant des classes.

Une classe est un type défini en C++

Par opposition aux types prédéfinis qui, eux, ne sont pas définis en C++ mais doivent, en quelque sorte, exister en dehors du langage pour lui permettre de fonctionner. En clair : il n'existe nulle part de code source rédigé en C++ qui définisse ce qu'est un `int`, par exemple.

Lorsqu'une variable a pour type une classe, on dit qu'elle est une instance de cette classe.

Le type `int` n'étant pas une classe, il est légèrement abusif de dire qu'une variable de type `int` est "une instance d'`int`". Cet abus n'est cependant pas scandaleux, car il respecte la signification essentielle du mot instance : exemplaire particulier d'une catégorie. De la même façon, il n'est pas déraisonnable de dire que Albert Einstein et Isaac Newton étaient des instances de la classe "êtres humains doués pour les sciences".

Puisqu'il s'agit d'un type, une classe ne peut être utilisée pour stocker une valeur particulière. Ce sont les éventuelles instances de cette classe qui seront représentées en mémoire lors de l'exécution du programme, et c'est dans ces variables qu'il sera possible de stocker des valeurs.

La classe "êtres humains doués pour les sciences" n'a jamais fait et ne fera jamais de découverte. Seules ses instances ont une existence concrète et peuvent faire quelque chose, pendant la durée de cette existence.

La même classe peut évidemment être adoptée pour créer un nombre quelconque de variables. Toutes ces instances auront les mêmes caractéristiques générales (les caractéristiques qui définissent justement la classe), mais pourront contenir des valeurs différentes (elles auront en effet des représentation distinctes en mémoire pendant l'exécution du programme) et sont susceptibles d'apparaitre ou disparaître indépendamment les unes des autres.

Newton n'avait aucune idée de la théorie de la relativité, et il est mort avant la naissance d'Einstein.

Le type d'une variable ne détermine pas seulement les valeurs que celle-ci peut contenir. En effet, les *opérations* envisageables dépendent elles aussi du type. Pour définir une classe, il va donc falloir spécifier non seulement ce qui peut être stocké dans une instance, mais aussi les traitements qui pourront être appliqués ou demandés à celle-ci.

Bucéphale, le cheval d'Alexandre le Grand, n'était pas une instance de la classe "êtres humains doués pour les sciences". Il aurait été sans espoir de lui demander de résoudre une équation. Une `std::string` contient du texte : il y a une certaine logique à lui demander combien de caractères comporte ce texte. Interroger un `bool` sur sa longueur, en revanche, n'aurait aucun sens.

Pour définir une classe, nous devrons donc décrire les capacités de stockage et le répertoire d'actions possibles pour une instance. Le stockage implique la présence de variables, l'action celle de fonctions.

Les variables et les fonctions qui définissent une classe sont dites membre de cette classe.

2 - Définir une classe

La définition d'une classe exige deux choses :
- la déclaration de tous ses membres (variables et fonctions) ;
- la définition des fonctions membre.

Les variables membre, pour leur part, ne seront définies que lors de l'instanciation : il doit en exister un jeu pour chaque instance (ce qui permet à celles-ci de stocker des valeurs différentes).

Déclaration des membres

Comme toutes les déclarations, celles-ci prennent habituellement place dans un fichier .h qui fera l'objet d'une directive d'inclusion en tête des fichiers dans lesquels la classe est mentionnée.

La syntaxe utilisée est assez simple :

```
class CEtudiant
{
public:
  double noteUn;        //déclaration d'une variable membre
  double noteDeux;      //déclaration d'une autre variable membre
  double moyenne();     //déclaration d'une fonction membre
};
```

Le mot `class` est un mot réservé qui sert simplement à annoncer que ce qui suit est la définition d'un type. Cette définition commence par annoncer le nom du type en question (qui sera nécessaire pour instancier la classe). Les déclarations de membres sont ensuite énumérées entre un couple d'accolades suivi d'un point virgule final.

> Le mot `public:` qui figure ici avant la déclaration du premier membre rend tous ceux-ci accessibles à quiconque à accès à une instance de la classe. Nous reviendrons bientôt sur la question (importante) de la gestion des privilèges d'accès aux membres.

Définition des fonctions membre

Comme toutes les définition de fonctions, celles des fonctions membre d'une classe prend habituellement place dans un fichier .cpp (il s'agit d'un texte source dont la compilation donnera naissance à du code exécutable).

Les fonctions membre d'une classe présentent deux originalités majeures par rapport aux fonctions qui ne sont membre d'aucune classe : elles ont un nom qui rappelle celui de leur classe, et elles ont accès à des variables qui ne leur appartiennent pas.

La définition de la fonction membre de notre classe pourrait se présenter ainsi :

```
double CEtudiant::moyenne()
{
return (noteUn + noteDeux) / 2;
}
```

Le nom complet d'une fonction membre comporte deux parties : le nom de la classe (c'est un peu le nom de famille de la fonction) et le nom déclaré dans la liste des membres (c'est un peu le prénom de la fonction). Ces deux parties sont reliées par un symbole spécial, le "double deux points".

Comme toutes les fonctions, les fonctions membre peuvent disposer de paramètres. Celle de notre exemple n'en comporte aucun, car elle accède aux données qui lui sont nécessaires en utilisant la spécificité principale des fonctions membre : elles peuvent mentionner par leur nom les variables membre de la classe à laquelle elles appartiennent.

Comment est-ce possible ?
Nous avons vu que l'idée même de créer un type implique celle d'instanciations multiples. Chacune de ces instanciations va se traduire en mémoire par une concrétisation du "plan" spécifié par la définition de la classe. Dans notre exemple, chaque variable de type `CEtudiant` va être capable de stocker une `noteUn` et une `noteDeux`. Dans ces conditions, comment la fonction `CEtudiant::moyenne()` peut-elle savoir sur quelles données effectuer le calcul qui lui est demandé ?

La réponse est très simple, et nous la connaissons déjà : les fonctions membre sont exécutées au titre d'une instance particulière. C'est donc sur les données contenues dans cette instance qu'elles opéreront par défaut.

3 - Utiliser une classe

Une fois la classe définie, elle peut servir à créer des variables.

Instanciation

Créer une variable dont le type est une classe ne présente aucune difficulté particulière : il s'agit toujours d'annoncer le type et le nom choisi pour la variable :

```
int main(int argc, char *argv[])
{
  QCoreApplication a(argc, argv);
  CEtudiant toto;
  CEtudiant lulu;
  return a.exec();
}
```

> Si la classe est définie dans un fichier .h, il faudra évidemment inclure ce fichier en tête de main.cpp

Notez au passage que, telle que nous l'avons définie, la classe `CEtudiant` ne permet pas d'initialiser ses instances. Pour permettre l'initialisation, une classe doit être un peu moins rudimentaire que ce premier exemple.

Accéder aux membres

Une fois une instance définie, on peut accéder aux variables membres en utilisant leur nom complet :

```
1  int main(int argc, char *argv[])
2  {
3    QCoreApplication a(argc, argv);
4    CEtudiant toto;
5    CEtudiant lulu;
   //premier partiel
6    toto.noteUn = 18;
7    lulu.noteUn = 12;
   //second partiel
8    toto.noteDeux = 15;
9    lulu.noteDeux = 7;
10   return a.exec();
11 }
```

Le nom complet d'une variable membre est obtenu en reliant le nom de l'instance concernée et le nom déclaré dans la définition de la classe au moyen d'un point.

Une fois que les variables contiennent des données valides, il devient possible d'effectuer des calculs :

```
1  int main(int argc, char *argv[])
2  {
3    QCoreApplication a(argc, argv);
4    CEtudiant toto;
5    CEtudiant lulu;
   //premier partiel
6    toto.noteUn = 18;
7    lulu.noteUn = 12;
   //second partiel
8    toto.noteDeux = 15;
9    lulu.noteDeux = 7;
   //affichage des moyennes
10   std::cout << "La moyenne de toto est : " << toto.moyenne() << "\n";
11   std::cout << "La moyenne de lulu est : " << lulu.moyenne() << "\n";
12   return a.exec();
13 }
```

C'est donc l'instance au titre de laquelle la fonction `CEtudiant::moyenne()` est appelée qui détermine quelles sont les variables utilisées lors du calcul effectué par sa ligne *3*:

```
1  double CEtudiant::moyenne()
2  {
3  return (noteUn + noteDeux) / 2;
4  }
```

Lorsque la fonction est appelée au titre de `toto`, ce sont les variables `toto.noteUn` et `toto.noteDeux` qui sont utilisées, alors que lorsque la fonction est appelée au titre de `lulu`, ce sont les variables `lulu.noteUn` et `lulu.noteDeux` qui sont utilisées. Il serait donc vain d'essayer d'utiliser le nom complet des variables membre dans le corps de la fonction `CEtudiant::moyenne()`.

> Non seulement c'est impossible (la fonction ne dispose d'aucune instance de la classe), mais ce serait extrêmement nuisible, puisque cela rendrait la fonction incapable d'opérer sur n'importe quelle instance, ce qui est sa raison d'être.

4 - Exercice

Créez, comme vous savez maintenant le faire, un projet de type "Application Qt4 en console" □.

Essayez, en vous aidant le moins possible du code proposé sur les pages précédente, de faire fonctionner le programme que nous venons de décrire □.

> La définition de la fonction `CEtudiant::moyenne()` peut figurer dans le fichier main.cpp ou, et c'est la solution que nous préférerons par la suite, prendre place dans un nouveau fichier que vous baptiserez etudiant.cpp. Pour créer ce fichier, faites comme pour créer un .h, mais sélectionner l'option "fichier source", plutôt que "fichier d'en-tête". Ce fichier .cpp devra comporter une directive d'inclusion du fichier .h correspondant (celui qui contient la définition de la classe `CEtudiant`).