Centre Informatique pour les Lettres et les Sciences Humaines

Apprendre C++ avec Qt : Fonctions

1 - Logique d'exécution des programmes

Lorsqu'on observe le texte source, un programme apparaît comme une succession de parties portant chacune un nom, un peu comme les chapitres d'un roman.

Dans le cas d'un programme, ces parties ne s'appellent pas des chapitres, mais des fonctions.

Tout comme les chapitres d'un roman, les fonctions sont disjointes les unes des autres : c'est seulement lorsque le texte correspondant à l'une d'entre elles est achevé que le texte correspondant à la suivante peut commencer, de sorte que

La définition d'une fonction ne figure jamais à l'intérieur de celle d'une autre fonction.

Si l'analogie avec les chapitres d'un roman décrit bien *l'aspect* du texte source, la lecture d'un roman et *l'exécution* d'un programme obéissent à des logiques totalement différentes.

La lecture normale d'un roman commence au premier chapitre et se déroule ensuite linéairement : à la fin d'un chapitre, on passe tout naturellement au suivant, dans l'ordre où on les trouve.

L'exécution d'un programme C++ est beaucoup plus simple, puisqu'une seule fonction est exécutée, celle qui est baptisée `main()`. Oui, vous avez bien compris,

Lors du lancement d'un programme, la seule fonction qui s'exécute spontanément est `main()`

Les fonctions possèdent en revanche une caractéristique dont sont normalement dépourvus les chapitres d'un roman : elles peuvent s'appeler les unes les autres. Du point de vue de son exécution, c'est un peu comme si le programme ne comportait qu'un chapitre (`main()`, en l'occurrence) et que toutes les autres fonctions n'étaient que des notes de bas de page qui ne seront lues/exécutées que dans la mesure où elles sont appelées dans un passage qui se trouve être lui-même lu/exécuté.

Si une fonction autre que `main()` est exécutée, c'est que son appel a été déclenché lors de l'exécution d'une autre fonction.

> Il existe des livres dont l'utilisation ressemble plus à l'exécution d'un programme qu'à la lecture d'un roman. Il est ainsi très rare que l'on cherche à lire un dictionnaire ou un livre de recettes de cuisine de la première à la dernière page. Il arrive en revanche que, au cours de la lecture d'une des parties de ces ouvrages, on s'interrompe pour aller lire une autre partie. Ainsi, si la définition que je lis comporte un mot que je ne connais pas, je peux aller consulter la définition de celui-ci avant de revenir à la lecture de la définition initiale. De même, si j'essaie de faire une pièce montée en suivant une recette, je vais devoir, le moment venu, aller consulter les recettes de la pâte à choux et de la crème anglaise. L'exécution d'un programme, c'est la réalisation de la recette `main()` : si une autre recette est réalisée au passage, c'est qu'elle a été exigée lors de l'une des étapes de de la recette `main()`.

Une conséquence de cette logique est que

L'ordre dans lequel les fonctions apparaissent dans le texte source n'a aucun lien avec l'ordre dans lequel elles seront (éventuellement) exécutées.

> Vous sentez-vous obligé d'exécuter toutes les recettes figurant dans le livre de cuisine, et ceci dans l'ordre où elles sont présentées ?

Une autre conséquence est que, à la fin de l'exécution d'une fonction, le programme ne passe pas à la "fonction suivante", mais reprend l'exécution de la fonction qui a appelé celle qui vient de s'achever.

> La fin de l'exécution d'une fonction ressemble beaucoup à la fin de la lecture d'une note de bas de page : on retourne d'où on venait.

A la fin de l'exécution d'une fonction, le programme reprend là où il a été interrompu par l'appel à celle-ci.

Le contexte d'exécution d'une fonction autre que `main()` comporte donc toujours deux fonctions :
- la fonction appelée, qui est en cours d'exécution ;
- la fonction appelante, dont l'exécution est suspendue dans l'attente de la fin de l'exécution de la fonction appelée.

2 - Définition de fonctions

La définition d'une fonction, c'est le fragment de texte source qui comporte les instructions qui seront exécutées lorsque la fonction sera appelée.

Ces instructions figurent dans un bloc de code qui constitue ce qu'on appelle le corps de la fonction.

Le corps d'une fonction est toujours précédé d'un en-tête, qui joue un rôle analogue au titre d'un chapitre. Outre le nom de la fonction définie, cet en-tête fournit deux informations concernant l'utilisation de celle-ci : le type de la valeur renvoyée par la fonction et les paramètres qu'elle utilise.

La définition d'une fonction aura donc l'allure générale suivante :

```
leType leNom(les paramètres) //en-tête de la fonction
{
 //le corps
}
```

Valeur renvoyée par une fonction

Certaines fonctions effectuent des traitements qui se soldent par l'obtention d'une valeur qui constitue le résultat de la fonction. Ces fonctions peuvent renvoyer le résultat en question, c'est à dire rendre la valeur obtenue disponible pour la fonction appelante.

Une fonction peut renvoyer son résultat à l'aide de l'instruction `return`

Par commodité de langage, le type du résultat renvoyé par une fonction est aussi appelé le *type de la fonction*.

Les fonctions peuvent aussi effectuer des traitements qui se soldent par un effet : formatage d'un disque dur, impression d'un texte, extinction de l'ordinateur... Certaines fonctions n'existent que pour l'effet que produit leur exécution, et n'ont donc pas de résultat à renvoyer.

Une fonction qui ne renvoie pas de résultat est de type `void`

> Le type `void` (vide, en anglais) n'est qu'un mot que C++ utilise pour expliciter l'absence. Il n'est évidemment pas possible de créer des variables de type `void`.

Paramètres d'une fonction

Le corps d'une fonction est un bloc d'instructions. Ce bloc peut évidemment contenir des définitions de variables, et les règles habituelles s'appliquent : ces variables sont locales à la fonction, c'est à dire que leur nom n'est connu que de la fonction en question et peut donc être choisi librement sans que cela n'affecte le fonctionnement du reste du programme, et ces variables sont détruites lorsque l'exécution du bloc s'achève.

A la différence des autres blocs, le corps d'une fonction peut posséder des variables locales qui ne sont pas définies dans le bloc, mais dans l'en-tête de la fonction. Ces variables locales particulières s'appellent des paramètres et leur seule singularité est que

La valeur d'initialisation d'un paramètre est transmise par la fonction appelante.

> Une fonction peut évidemment être appelée à plusieurs reprises au cours de l'exécution d'un programme. Lors de chacune de ces exécutions, un jeu complet de variables locales doit être (re)créé, puisque ces variables ont été détruites à la fin de l'exécution précédente. Le statut des paramètres leur confère donc une particularité importante : leur valeur initiale n'est pas forcément la même lors de chaque exécution de la fonction.

3 - Appel d'une fonction

L'appel d'une fonction est une instruction exécutable et ne peut donc figurer que dans le corps d'une (autre) fonction, la fonction appelante.

Cette instruction prend simplement la forme du nom de la fonction appelée, suivi d'un couple de parenthèses.

Si la fonction appelée comporte des paramètres, les valeurs destinées à initialiser ceux-ci doivent figurer entre les parenthèses de l'appel.

Si la fonction appelée comporte plusieurs paramètres, la fonction appelante devra fournir plusieurs valeurs d'initialisation.

La mise en correspondance des valeurs et des paramètres repose uniquement sur leur ordre : la première valeur figurant entre les parenthèses de l'appel est destinée à initialiser le premier paramètre de la fonction, et ainsi de suite.

Chaque valeur fournie par la fonction appelante devra être d'un type correspondant à celui du paramètre qu'elle est destinée à initialiser.

Si la fonction appelée renvoie une valeur, la fonction appelante peut utiliser l'instruction d'appel comme s'il s'agissait d'une valeur. Dans le cas le plus simple, elle peut par exemple affecter cette valeur à une variable du même type que la fonction.

4 - Déclaration d'une fonction

Le type de la valeur renvoyée, le nom de la fonction et la liste de ses paramètres constituent des informations essentielles à l'utilisation de celle-ci.

Ces informations doivent évidemment être connues du programmeur au moment où celui-ci insère une instruction d'appel dans la fonction appelante, mais elles doivent aussi être disponibles pour le compilateur au moment où celui-ci procède à la traduction de cet appel en instructions exécutables par le processeur.

Le moyen le plus simple et le plus sûr pour garantir cette disponibilité est de faire figurer, en tête du fichier qui va contenir la définition des fonctions, une déclaration de toutes les fonctions appelées par celles-ci.

La déclaration d'une fonction est identique à son en-tête, mais doit se terminer par un point-virgule.

> Ce point virgule annonce qu'il s'agit d'une simple déclaration, c'est à dire que la suite du texte ne doit pas être interprétée comme constituant le corps de la fonction. Dans une définition de fonction, il ne faut donc **jamais** mettre de point-virgule entre l'en-tête et le corps...

La pratique habituelle est de faire figurer les définitions de fonctions dans des fichiers portant l'extension .cpp et les déclarations dans des fichiers portant l'extension .h.

Dans ce contexte, la présence des déclarations en tête du fichier contenant les définitions est assurée à l'aide d'une directive d'inclusion du fichier .h pertinent.

5 - Exemple

Le programme suivant est dépourvu de signification particulière et donc de tout intérêt en dehors du fait qu'il illustre la mise en œuvre des mécanismes que nous venons de décrire. Il est composé d'un fichier nommé mesFonctions.h :

```
1 #ifndef MESFONCTIONS_H
2 #define MESFONCTIONS_H

3 void uneFonctionSansResultatNiParametre();
4 double jeSaisFaireUneAddition(double n1, double n2);

5 #endif // MESFONCTIONS_H
```

Les lignes 3 et 4 constituent deux déclarations de fonctions, les autres lignes ont été rédigées automatiquement lors de la création du fichier par QtCreator et n'ont aucun intérêt dans le cas présent.

La définition des deux fonctions en questions, ainsi que celle de la fonction `main()`, figurent pour leur part dans le fichier main.cpp :

```
1  #include <QtCore/QCoreApplication>
2  #include <iostream>
3  #include "mesFonctions.h"     //assure la présence des déclarations

   //*************************************************************************
4  int main(int argc, char *argv[])
5  {
6      QCoreApplication a(argc, argv);
7      uneFonctionSansResultatNiParametre();           //appel d'une fonction
8      int resultat(0);
9      resultat = jeSaisFaireUneAddition(2,3);         //appel, passage de valeurs
10     std::cout << "premier resultat : " << resultat << "\n";
11     resultat = jeSaisFaireUneAddition(resultat,5);  //second appel, autres valeurs
12     std::cout << "second resultat : " << resultat << "\n";
13     return a.exec();
14 }
   //*************************************************************************
15 void uneFonctionSansResultatNiParametre()
16 {
17    std::cout << "exécution d'uneFonctionSansResultatNiParametre" << "\n";
18 }
   //*************************************************************************
19 double jeSaisFaireUneAddition(double n1, double n2)
20 {
21    double somme(0); //une variable locale à la fonction
22    somme = n1 + n2;
23    return somme;
24 }
```