Centre Informatique pour les Lettres et les Sciences Humaines

Apprendre C++ avec QtCreator
Etape 3 : Faire fonctionner "Devine un nombre"

Le but de cette étape n'est pas de rendre programme "Devine un nombre" plus amusant, mais d'utiliser dans son code des techniques qui seront indispensables pour rédiger des programmes plus ambitieux.

1 - Le code obtenu à l'issue de l'étape 2

```
#include <QtCore/QCoreApplication>
#include <iostream>
#include <ctime>
#include <cstdlib>

int main(int argc, char *argv[])
{
QCoreApplication a(argc, argv);
srand(time(0));
int encore(1);
double nbCoups(0);
int nbParties(0);

do {//boucle permettant de faire plusieurs parties
   int solution = (rand() % 32);
   nbParties = nbParties + 1;
   std::cout << "J'ai choisi un nombre entier positif plus petit que 32";
   std::cout << "\nEssayez de le deviner !\n\n";
   int proposition(-1);
   int nbCoupsPartie(0);

   do {//boucle de saisie des propositions
      std::cout << "Proposez un nombre : ";
      std::cin >> proposition;
      nbCoupsPartie = nbCoupsPartie + 1;
      if(proposition < solution)
        std::cout << "mon nombre est plus grand que " << proposition << "\n";
      if(proposition > solution)
        std::cout << "mon nombre est plus petit que " << proposition << "\n";
      } while(proposition != solution);

   std::cout << "Victoire en " << nbCoupsPartie << " coups\n";
   nbCoups = nbCoups + nbCoupsPartie;
   std::cout << "Votre moyenne est de ";
   std::cout << nbCoups/nbParties << " coups par partie\nEncore ?\n";
   std:: cin >> encore;
   } while (encore == 1);

return a.exec();
}
```

Si vous ne vous sentez pas capable d'écrire vous-même (ie. sans recopier un modèle) la fonction ci-dessus, vous devriez **travailler encore l'étape 2** plutôt que de chercher à passer à la suite.

Si vous rencontrez des difficultés, n'hésitez pas à vous manifester (forum, mail...).

Ne croyez surtout pas que ces difficultés vont disparaître par magie si vous les traitez par le mépris. Elles vont au contraire s'aggraver, parce que la suite de notre étude du langage suppose en permanence la maîtrise des mécanismes de base illustrés par cette version de "Devine un nombre" : création et utilisation de variables, boucles et tests.

Une dernière fois :

NE TOURNEZ PAS LA PAGE SI LE CODE CI-DESSUS VOUS SEMBLE MYSTERIEUX !

2 - Création d'une fonction

Si l'on examine le code de "Devine un nombre", on remarque qu'il s'agit fondamentalement d'une boucle enchâssée dans une autre : une partie est une répétition de séquences saisie-verdict, et cette répétition est elle-même répétée tant que l'utilisateur le souhaite.

Sur cette structure fondamentale se greffent des détails techniques tels que la création et l'initialisation des variables nécessaires et les tests qui permettent l'affichage du bon verdict. Une grande partie de la difficulté de mise au point du programme est justement d'arriver à gérer ces détails correctement à l'intérieur des deux boucles.

Lorsque nous allons chercher à écrire des programmes un peu plus complexes, cette façon de rédiger le code source va se traduire par des boucles à l'intérieur de boucles à l'intérieur de boucles à l'intérieur de boucles à l'intérieur....

De plus, le nombre des "détails techniques" qui devront être gérés correctement dans cet enchâssement vertigineux de boucles ne va cesser de croître.

Conclusion : ce n'est pas comme ça qu'il faut s'y prendre.

Un des outils que propose C++ pour dompter la complexité croissante des programmes est le découpage de ceux-ci en fonctions. Dans le cas présent, il est tentant de faire de la gestion d'une partie une fonction autonome. Il devient alors possible de cacher l'enchâssement des boucles : il suffit d'appeler la fonction en question depuis l'intérieur d'une boucle unique :

```
1  int main(int argc, char *argv[])
2  {
3  QCoreApplication a(argc, argv);
4  srand(time(0));
5  int encore(1);
6  double nbCoups(0);
7  int nbParties(0);
8  do {
9     nbParties = nbParties + 1;
10    int reponseFonction = jouePartieEtRenvoieNbCoups();
11    nbCoups = nbCoups + reponseFonction;
12    std::cout << "Victoire en " << reponseFonction << " coups\n";
13    std::cout << "Votre moyenne est de " << nbCoups/nbParties;
14    std::cout << " coups par partie\nEncore ?\n";
15    std:: cin >> encore;
16    } while (encore == 1);
17 return a.exec();
18 }
```

Cette façon de procéder exige évidemment que la fonction responsable du déroulement d'une partie soit elle aussi définie :

```
19 int jouePartieEtRenvoieNbCoups()
20 {
21 int solution = rand() % 32;
22 std::cout << "J'ai choisi un nombre entier positif plus petit que 32\n";
23 std::cout << "Essayez de le deviner !\n\n";
24 int proposition(-1);
25 int nbCoupsPartie(0);
26 do {
27    std::cin >> proposition;
28    nbCoupsPartie = nbCoupsPartie + 1;
29    if(proposition < solution)
30       std::cout << "mon nombre est plus grand que " << proposition << "\n";
31    if(proposition > solution)
32       std::cout << "mon nombre est plus petit que " << proposition << "\n";
33    } while(proposition != solution);
34 return nbCoupsPartie;
35 }
```

Une fonction peut renvoyer une réponse à l'aide de l'instruction `return`

Remarquez que l'en-tête de la fonction *(ligne 19)* mentionne le fait que l'exécution celle-ci s'achève en produisant une valeur de type `int`. Cette valeur est effectivement produite par la dernière instruction de la fonction *(34).* L'instruction qui appelle cette fonction *(10)* prend soin de recueillir cette valeur dans une variable pour pouvoir s'en servir ensuite *(11 et 12)*.

3 - En pratique

Créez, en suivant la procédure décrite pour l'étape 1, un projet de type <Application Qt4 en console>.

Donnez à votre fonction `main()` le contenu suggéré ci-dessus *(lignes 1-18)* et faites-la suivre de la définition de la fonction `jouePartieEtRenvoieNbCoups()` *(lignes 19-35)*.

> N'oubliez pas qu'une directive d'inclusion de `iostream` doit figurer en tête de votre fichier.

Déclaration et définition

Essayez de compiler votre programme.

Le message d'erreur que vous obtenez indique que la fonction `jouePartieEtRenvoieNbCoups()` n'a pas été reconnue par le compilateur :

```
error: 'jouePartieEtRenvoieNbCoups' was not declared in this scope
```

En effet, lorsqu'il il tente de traduire la ligne *(10)* qui appelle cette fonction, le compilateur n'en a encore jamais entendu parler.

Il serait ici possible de résoudre le problème en inversant, à l'aide d'un simple copier/coller, l'ordre des définitions de nos deux fonctions dans le fichier source.

> En effet, si la définition de `jouePartieEtRenvoieNbCoups()` figure dans le texte source avant celle de `main()`, l'appel qui figure dans `main()` ne sera plus rencontré avant que la fonction concernée ne soit définie.

Cette approche n'est toutefois pas assez générale pour être satisfaisante, et nous allons donc plutôt adopter la méthode communément utilisée, qui consiste à placer en tête du fichier source une déclaration des fonctions qui y seront ensuite définies (à l'exception de `main()`, qui ne doit jamais être déclarée explicitement). Cette déclaration prend exactement la forme de l'en-tête de la fonction concernée et doit être suivie d'un point virgule. Notre fichier source adopte donc le plan suivant :

```
#include <QtCore/QCoreApplication>
#include <iostream>

int jouePartieEtRenvoieNbCoups(); //déclaration de cette fonction

//*************************************************************************
int main()
{
//corps de la fonction main()
}
//*************************************************************************
int jouePartieEtRenvoieNbCoups()
{
//corps de la fonction jouePartieEtRenvoieNbCoups()
}
```

> Séparer clairement les définitions des fonctions les unes des autres (en faisant, par exemple, précéder chacune d'entre-elles d'une ligne d'étoiles) facilite grandement la lecture du fichier source, et donc la mise au point du programme.

Rendez le programme compilable en adoptant le plan suggéré ci-dessus.

Utilisation d'un fichier d'en-tête

Dans la plupart des cas, les déclarations nécessaires ne figurent pas *directement* en tête du fichier source, mais y sont "injectées" à l'aide d'une directive d'inclusion. Bien qu'un peu plus lourde à mettre en place, cette façon de procéder s'avère infiniment plus pratique lorsque certaines fonctions en appellent d'autres *qui ne sont pas définies dans le même fichier source*.

> L'éclatement du programme en plusieurs fichiers source devient inéluctable lorsque le volume de code s'accroît un tant soit peu. Un cas voisin est celui des fonctions qui ne sont pas définies dans un fichier source (ie. un texte en C++) mais dans une librairie (ie. sous la forme de code déjà compilé). C'est pourquoi nous avons déjà rencontré à plusieurs reprises des directives d'inclusion.

Dans le Menu <Fichier>, sélectionnez la commande <Nouveau fichier ou projet...>.

Dans la boîte de dialogue qui s'ouvre alors, sélectionnez <C++> et <Fichier Header C++>, puis cliquez sur [Ok].

Le dialogue suivant permet de baptiser le fichier. Donnez lui pour nom "declarationsDeMesFonctions" et cliquez sur [Suivant], puis sur [Terminer] dans la fenêtre suivante.

Ne donnez jamais à vos projets ou fichiers des noms comportant des espaces ou des minuscules accentuées, même si votre système d'exploitation l'autorise.

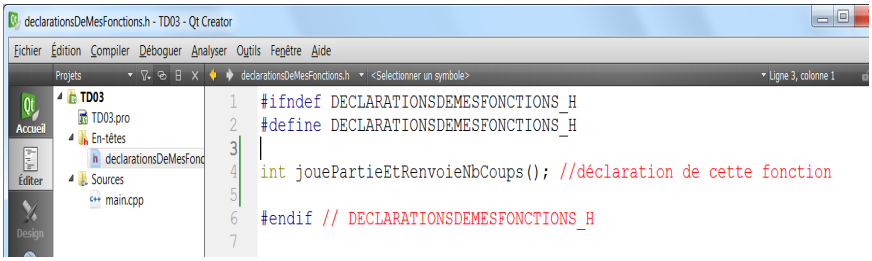
> QtCreator fait appel à des logiciels vénérables, développés à l'origine sous des systèmes qui n'envisageaient pas l'usage d'une langue autre que l'anglais. Ces contraintes sont le prix qu'il faut actuellement payer pour utiliser ces outils gratuits.

Dans le dernier dialogue, vérifiez que l'option <Ajouter au projet> est bien cochée et cliquez sur [Terminer].

L'éditeur ouvre alors le nouveau fichier, dans lequel QtCreator a placé pour vous quelques lignes destinées à vous défendre contre des cas particuliers difficiles à repérer et à gérer. Remerciez-le mentalement et souvenez vous que :

Dans un fichier d'en-tête, il faut toujours placer les déclaration entre les directives `#define` et `#endif`.

Donnez donc à votre fichier le contenu suivant :



> Remarquez que QtCreator a attribué à votre fichier l'extension ".h", ce qui correspond à la tradition concernant les fichiers d'en-tête en C++ (en-tête = **h**eader, en anglais).

Dans votre fichier main.cpp, remplacez la déclaration de la fonction par une directive d'inclusion :

```
#include <QtCore/QCoreApplication>
#include <iostream>
#include "declarationsDeMesFonctions.h"

//*************************************************************************************
int main()
{
//corps de la fonction main()
}
//*************************************************************************************
int jouePartieEtRenvoieNbCoups()
{
//corps de la fonction jouePartieEtRenvoieNbCoups()
}
```

Vérifiez que votre programme reste compilable en dépit des faits que main.cpp ne comporte plus de déclaration directe de la fonction `jouePartieEtRenvoieNbCoups()` et que celle-ci y est appelée (par `main()`) avant d'y être définie.

4 - Exercice

Dans sa version actuelle, le programme n'effectue aucune vérification sur la saisie effectuée par l'utilisateur, qui peut très bien être négative, supérieure à 32, ou même ne pas être un nombre. En cas de faute de frappe, un essai échoué est donc comptabilisé dans la partie, au détriment du joueur.

Créez une fonction `saisieProposition()` à laquelle la fonction `jouePartieEtRenvoieNbCoups()` fera appel (à la ligne *27*, en lieu et place de l'instruction `std::cin >> proposition;`).

Cette fonction devra s'assurer que la saisie est bien un nombre positif inférieur à 32 et, dans le cas contraire, devra exiger une nouvelle saisie (en affichant de préférence un message pour indiquer à l'utilisateur ce qui ne vas pas). Cette fonction ne renverra donc que des propositions vraisemblables.

> Cet exercice porte sur la création (déclaration et définition) d'une fonction et sur son utilisation (appel). Ne cherchez pas à sophistiquer outre mesure les tests de validité et les messages utilisés. Renoncez, en particulier, à l'idée de gérer élégamment les saisies non numériques (c'est un problème plus complexe qu'il en a l'air).