Centre Informatique pour les Lettres et les Sciences Humaines

Apprendre C++ avec QtCreator
Etape 2 : Devine un nombre !

Le programme que nous allons mettre au point au cours de cette seconde étape est un grand classique de l'initiation à la programmation : il tire un nombre au hasard et propose à l'utilisateur d'essayer de le deviner. Si ce jeu manque totalement d'intérêt, il va cependant nous fournir l'occasion d'aborder les structures de contrôles, qui permettent de créer des programmes qui ne sont pas de simples listes d'instructions exécutées l'une après l'autre, dans l'ordre où elles figurent dans la liste.

1 - Première version : en un coup

Créez, en suivant la procédure décrite pour l'étape 1, un projet de type <Application Qt 4 en console>.

Tirer un nombre au hasard

La librairie standard propose une fonction qui fournit une valeur entière imprévisible. Pour illustrer son fonctionnement, donnez à votre fichier main.cpp le contenu suivant :

```
1  #include <QtCore/QCoreApplication>
2  #include <iostream>
3  #include <cstdlib>
4  #include <ctime>
5  int main(int argc, char *argv[])
6  {
7  QCoreApplication a(argc, argv);
8  std::cout << rand() << "\n" << rand() << "\n" << rand();
9  return a.exec();
10 }
```

Le texte "\n" représente une demande de passage à la ligne suivante et permet donc ici d'obtenir l'affichage des trois valeurs sur des lignes différentes. La succession de << permet d'obtenir des affichages successifs, exactement comme si on employait trois instructions d'affichage différentes.

Si vous n'avez pas oublié d'insérer, avant la fonction `main()`, les directives `#include` nécessaires, vous pouvez faire construire, puis exécuter votre programme.

Les nombres affichés donnent effectivement l'impression d'être aléatoires, mais que se passe-t-il si, après avoir refermé la fenêtre, vous demandez une nouvelle exécution du programme ?

Un ordinateur n'est pas réellement capable d'avoir un fonctionnement aléatoire, et les nombres fournis par la fonction `rand()` sont le résultat d'un calcul qui, pour donner des résultats "imprévisibles", a besoin d'un point de départ lui-même "imprévisible". Ce point de départ peut être fixé à l'aide d'une autre fonction de la librairie standard, et il est de coutume d'utiliser l'heure d'exécution (fournie par une troisième fonction) pour "amorcer" le générateur de nombres "imprévisibles".

```
1  int main(int argc, char *argv[])
2  {
3  QCoreApplication a(argc, argv);
4  srand(time(0));
5  std::cout << rand() << "\n" << rand() << "\n" << rand();
6  return a.exec();
7  }
```

Avant d'utiliser `rand()`, un programme devrait toujours appeler une fois (et une seule) `srand()`.

Modifiez votre fonction `main()` comme suggéré ci-dessus, et exécutez plusieurs fois votre programme pour constater que les valeurs ont cessé de se répéter.

Bien qu'on puisse maintenant les considérer "imprévisibles", les nombres fournis par `rand()` ont encore un aspect qui les rend impropres à l'usage que nous souhaitons en faire : ils peuvent être très grands, et il n'est pas réaliste d'attendre de l'utilisateur qu'il devine un nombre tiré d'un ensemble aussi vaste.

Pour ramener les valeurs tirées dans un intervalle plus raisonnable, nous allons utiliser une propriété mathématique élémentaire : dans une division entière, le reste est toujours strictement inférieur au diviseur. L'utilisation de cette propriété est rendue très facile par le fait que le langage C++ dispose d'un opérateur "reste de la division entière" (aussi appelé "modulo"), noté `%`. Si nous souhaitons obtenir une valeur imprévisible tirée, par exemple, de l'ensemble {0, 1, 2, 3, 4, 5}, nous pouvons écrire :

```
rand() % 6;
```

Si `A` et `B` sont deux nombres, `A % B` est le reste de la division de `A` par `B`.

Si nous choisissons de tirer un nombre inférieur à 32, le début de notre programme pourrait donc être :

```
1  int main(int argc, char *argv[])
2  {
3    QCoreApplication a(argc, argv);
4    srand(time(0));
5    int tirage = rand() % 32;
6    std::cout << "J'ai choisi un nombre entier positif plus petit que 32";
7    std::cout << "\nEssayez de le deviner !\n\n";
8    int proposition (-1);
9    std::cout << "Proposez un nombre : ";
10   std::cin >> proposition;
```

Le problème qui se pose alors est celui du verdict : l'utilisateur a-t-il proposé la bonne valeur ?

Afficher le verdict : l'exécution conditionnelle

Pour savoir s'il convient de consoler ou de féliciter le joueur, il faut comparer sa proposition à notre tirage. Le langage C++ propose six opérateurs permettant de comparer deux valeurs (cf. tableau ci-contre).

Ici, la victoire correspond au cas ou la comparaison

```
proposition == tirage
```

se solde par une réponse affirmative.

Opérateur	Signification
==	égal à
!=	non égal à (ie. différent de)
<	inférieur à
<=	inférieur ou égal à
>	supérieur à
>=	supérieur ou égal à

Le résultat d'une telle comparaison peut être utilisé pour décider quelles lignes du programme doivent être exécutées. On parle alors d'*exécution conditionnelle*, et cet effet peut être obtenu à l'aide des mots `if` et `else` (*si* et *sinon*). La fin de notre fonction `main()` prend donc la forme suivante :

```
11   if (proposition == tirage){
12      std::cout << "Bravo !";
13   } else {
14      std::cout << "Dommage... Il fallait dire " << tirage;
15   }
16 return a.exec();
17 }
```

> Le langage n'impose aucune règle de"mise en page" (passages à la ligne et indentations) du texte source. Vos chances de succès à moyen terme augmenteront cependant beaucoup si vous adoptez des règles facilitant la relecture et que vous les appliquez SYSTEMATIQUEMENT. Le style adopté dans les documents du cours est celui utilisé par les fonctions de mise en page automatique de QtCreator.

Remarquez que, à la ligne *5*, la valeur tirée (puis ramenée dans l'intervalle souhaité) est stockée dans une variable. Le tirage n'a lieu qu'une seule fois, au moment ou la ligne *5* est exécutée. Toutes les mentions ultérieures de `tirage` désignent donc l'unique valeur tirée. Ceci n'aurait pas été le cas si notre programme répétait l'expression `rand() % 32` aux lignes *11* et *14* : chaque évaluation de cette expression aurait donné un résultat différent, et le programme aurait été incohérent.

Donnez à votre fonction `main()` le contenu suggéré ci-dessus (lignes *1* à *17*) et vérifiez que votre programme fonctionne □.

L'exécution conditionnelle peut être obtenue à l'aide du mot `if` suivi d'une paire de parenthèses entourant une expression dont on peut déterminer si elle est vraie ou fausse. Si (et seulement si) cette expression est vraie, le bloc d'instructions suivant (délimité par un couple d'accolades) est exécuté.

En cas de besoin, ce bloc de code peut être suivi du mot `else` et d'un second bloc d'instructions qui sera exécuté si et seulement si le premier bloc ne l'a pas été.

> La présence du `else` et du second bloc de code est donc optionnelle. En leur absence, et dans le cas où l'expression de contrôle du `if` est fausse, le programme se poursuit normalement, comme si le `if` et son bloc d'instructions n'étaient pas là.

Une expression dont on peut dire si elle est vraie ou fausse est appelée une expression booléenne.

2 - Seconde version : en plusieurs coups

Cette première version du jeu est *vraiment* sans intérêt : l'utilisateur ne peut compter que sur un pur hasard et n'a qu'une chance sur 32 de gagner. Pour améliorer un peu les choses, notre seconde version va laisser l'utilisateur procéder à plusieurs tentatives et lui donner matière à réflexion.

Répéter la question

Une approche envisageable est de continuer à demander une proposition tant que l'utilisateur ne gagne pas. Ceci peut être obtenu à l'aide de l'instruction `do ... while()`, que l'on peut effectivement traduire par *répète ... tant que()*. Notre programme ressemblerait alors à ceci :

```
1  int main(int argc, char *argv[])
2  {
3    QCoreApplication a(argc, argv);
4    srand(time(0));
5    int tirage = rand() % 32;
6    std::cout << "J'ai choisi un nombre entier positif plus petit que 32";
7    std::cout << "\nEssayez de le deviner !\n\n";
8    int proposition = -1;
9    do {
10       std::cout << "Proposez un nombre : ";
11       std::cin >> proposition;
12       } while (proposition != tirage);
13   std::cout << "Bravo !";
14   return a.exec();
15 }
```

> Remarquez que l'exécution conditionnelle réalisée précédemment à l'aide d'un `if else` n'est plus nécessaire : par définition, l'exécution du bloc de lignes *10-11* va être répétée jusqu'à ce que l'utilisateur finisse par trouver (ou par se lasser et refermer la fenêtre, mettant ainsi brutalement fin à cette mauvaise plaisanterie). La ligne *13* ne sera donc, quoi qu'il en soit, exécutée que si l'utilisateur a gagné. Elle peut donc le féliciter sans se poser de question.

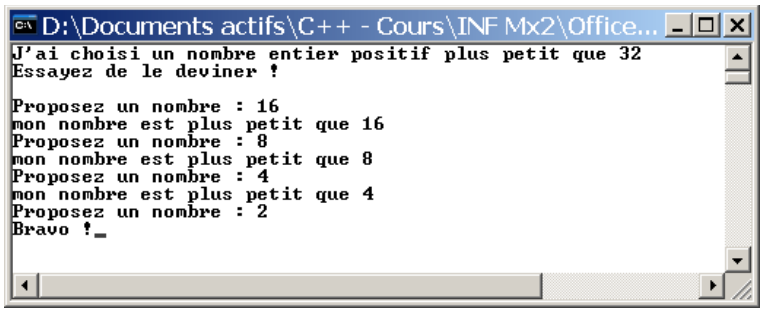
Donnez à votre fonction `main()` le contenu suggéré ci-dessus et vérifiez que votre programme fonctionne ☐.

La répétition de l'exécution d'un bloc d'instructions peut être obtenue le faisant précéder du mot `do` et suivre du mot `while` accompagné d'un couple de parenthèses entourant une expression booléenne.

> Le bloc est exécuté une première fois, puis l'expression booléenne est évaluée et, si elle est vraie, l'exécution du programme "remonte" au `do`. Lorsque l'expression booléenne finit par être fausse, l'exécution du programme se poursuit normalement.

Fournir des indices

Pour commencer à ressembler vaguement à un jeu, notre programme doit permettre à l'utilisateur de faire autre chose que taper des nombres au hasard. On peut, par exemple, lui indiquer si le nombre recherché est plus grand ou plus petit que sa proposition :

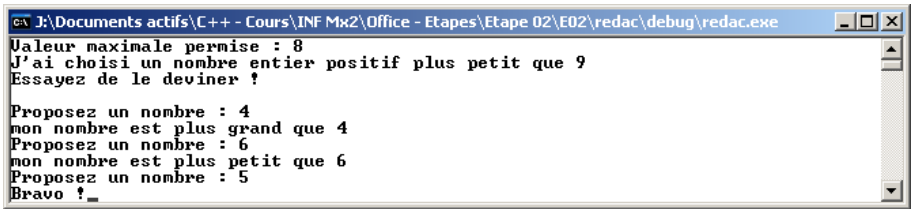


Modifiez le programme pour qu'il se comporte ainsi ☐.

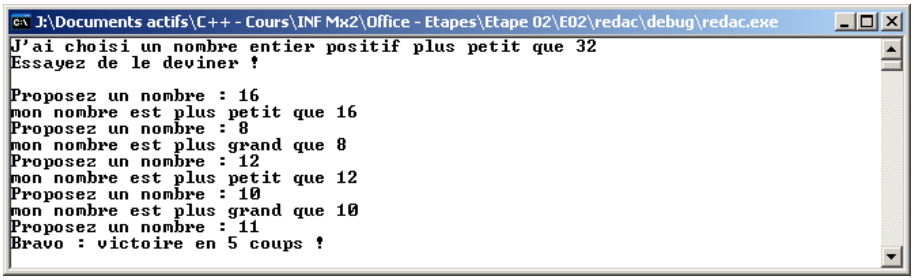
> Vous savez dès à présent tout ce qu'il y a à savoir pour effectuer les modifications requises. Mais la logique des opérations est peut-être un peu plus complexe qu'elle en a l'air, et quelques tests pourraient bien révéler que votre premier programme syntaxiquement correct reste incorrect du point de vue de son comportement (au moment de la victoire ?)...

3 - Exercices

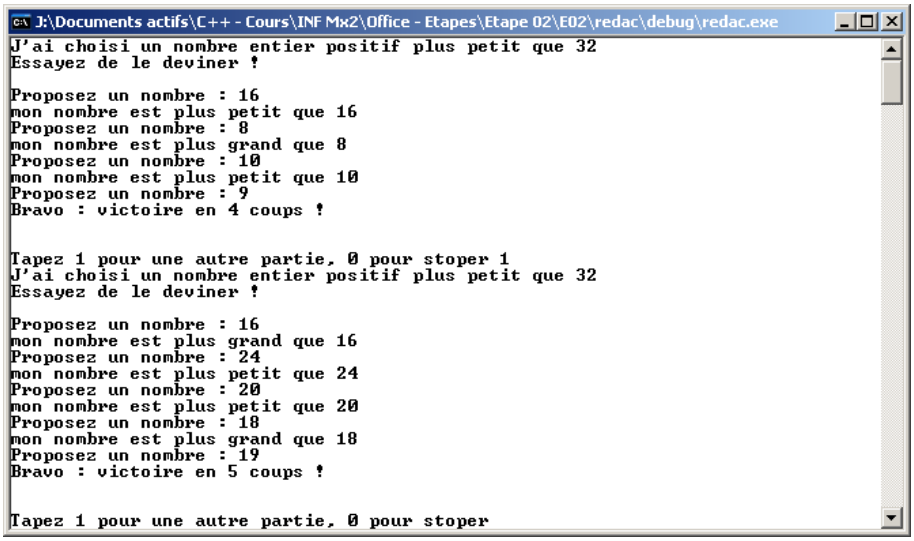
1) Modifiez le programme pour que l'utilisateur puisse, au début du jeu, indiquer la valeur maximale autorisée pour le tirage.



2) Modifiez le programme de façon à compter les propositions faites par l'utilisateur.



3) Modifiez le programme de façon à pouvoir faire plusieurs parties sans avoir à le relancer.



4) Modifiez le programme de façon à ce qu'il affiche le nombre moyen de propositions nécessaires à l'utilisateur pour trouver la solution.