Centre Informatique pour les Lettres et les Sciences Humaines

Apprendre C++ avec QtCreator
Etape 4 : ASCII art

Avant d'abandonner les projets basés sur une interface utilisateur de type "console", nous allons nous intéresser à un autre grand classique de cette époque : la réalisation de patterns graphiques composés de caractères alphanumériques "artistiquement" disposés. Les compositions que nous allons réaliser sont avant tout un prétexte à l'utilisation de fonctions et seront donc moins ambitieuses que certains exemples connus (cf. http://fr.wikipedia.org/wiki/Art_ASCII).

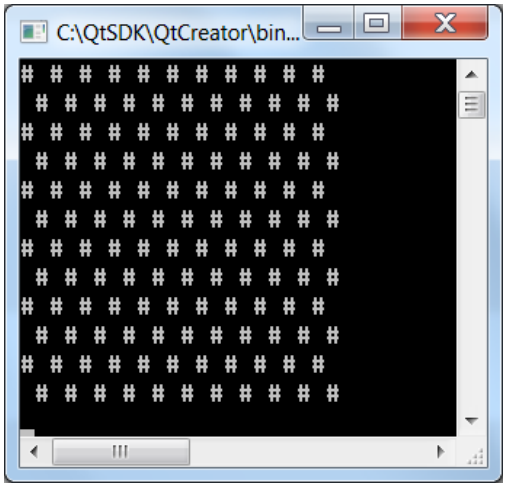
1 - Révisions : utilisation d'une fonction

Créez, comme vous savez maintenant le faire, un projet de type <Application Qt4 en console> ☐.

Donnez à votre fonction `main()` le contenu suivant ☐ :

```
1  int main(int argc, char *argv[])
2  {
3  QCoreApplication a(argc, argv);
4  int ligne(0);
5  do {
6     dessineLigne();
7     std::cout << " "; //un simple espace
8     dessineLigne();
9     ++ligne;
10    } while (ligne < 6);
11 return a.exec();
12 }
```

Définissez la fonction `dessineLigne()` de façon à ce que l'exécution du programme produise l'affichage représenté ci-contre ☐.



Comme cette fonction n'a aucune valeur particulière à renvoyer (elle a pour effet d'afficher des caractères, mais ne procède à aucun calcul produisant un résultat), elle pourra être déclarée de type `void` :

```
void dessineLigne();
```

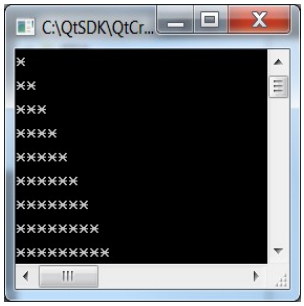
et s'abstenir de tout usage de l'instruction `return`.

2 - Fonctions avec paramètres

L'exemple précédent illustre bien un défaut de l'utilisation que nous faisons des fonctions : comme leur exécution produit toujours exactement le même effet, toute les variations souhaitées (ici, la présence ou non d'un espace en début de ligne) doivent être prise en charge par la fonction appelante (`main()`, dans le cas présent, dont la ligne *7* "aide" la fonction à alterner les patterns).

Remarquez que cette façon de procéder a un coût : la fonction `dessineLigne()` doit être appelée deux fois (lignes *6* et *8*), ce qui fait que la variable de contrôle de la boucle (`ligne`) ne reflète plus directement le nombre de lignes effectivement dessinées.

Imaginons, par exemple, que nous cherchions à créer le "demi sapin de Noël" représenté ci-contre.



Comme ce dessin ne présente pas deux lignes identiques, l'utilisation d'une fonction condamnée à faire toujours la même chose ne présente guère d'intérêt.

Pour rendre nos fonctions capables de se plier aux exigences de celles qui les appellent, nous pouvons les doter d'un type de variable spécial, qu'on appelle des paramètre.

Dans cet exemple, le code prendrait l'allure suivante :

```
1  int main(int argc, char *argv[])
2  {
3  QCoreApplication a(argc, argv);
4  int nb(1);
5  do {
6      ligneCrescendo(nb); //affiche une ligne de nb étoiles
7      ++nb;
8      } while (nb < 10);
9  return a.exec();
10 }
   //*************************************************************************
11 void ligneCrescendo(int nbCol)
12 {
13 int pos(0);
14 do {
15     std::cout << "*";
16     ++pos;
17     } while (pos < nbCol);
18 std::cout << "\n";
19 }
```

Deux points sont à remarquer :

1) A la ligne 6, l'appel de la fonction `ligneCrescendo` est effectué en insérant une valeur entre les parenthèses. Cette valeur est ici désignée par le nom de la variable qui la contient.

> Dans un autre contexte, il serait tout a fait possible d'écrire quelque chose comme
> ```
> ligneCrescendo(36); //affiche une ligne de 36 étoiles
> ```

2) La fonction `ligneCrescendo` comporte une variable qui est définie dans son en-tête, entre les parenthèses qui suivent son nom. Une telle variable ne présente qu'une seule particularité par rapport aux variables locales à une fonction qui sont définies dans le corps de celle-ci : lors de chaque exécution de la fonction, la valeur d'initialisation d'un paramètre est fixée par la fonction appelante.

Bien entendu, cette valeur est celle qui figure dans l'instruction d'appel de la fonction. On dit que cette valeur est transmise, ou passée, à la fonction appelée par la fonction appelante.

> Dans le cas présent, la fonction `ligneCrescendo` va être appelée 9 fois, et la valeur transmise sera différente lors de chacun de ces appels, puisqu'elle est spécifiée par le contenu d'une variable qui prend successivement les valeurs 1, 2, 3, 4, 5, 6, 7, 8 et 9.

Le principe fondamental gouvernant l'utilisation de paramètres est donc très simple :

Lorsqu'on appelle une fonction comportant un paramètre, on fixe la valeur initiale de celui-ci en passant une valeur à la fonction appelée.

En dehors de cette particularité, un paramètre possède exactement les mêmes propriétés que les autres variables locales d'une fonction :

- Il est détruit lorsque l'exécution de la fonction s'achève et devra donc être recréé lors d'un éventuel appel ultérieur de la fonction.

  > C'est cette recréation qui lui permet de ne pas être toujours initialisé avec la même valeur.

- Son nom n'est connu que de la fonction à laquelle il appartient, et peut donc être changé librement sans que cela affecte autre chose que le code contenu à l'intérieur du corps de celle-ci.

  > En d'autre termes, une fonction est libre de baptiser ses variables comme elle l'entend, cela ne regarde qu'elle. C'est bien normal puisque, de toutes façons, elle seule peut accéder à ses variables en utilisant le nom qu'elle leur a donné.

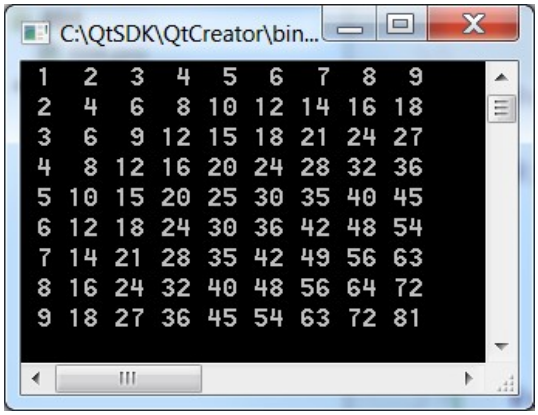
3 - Exercices

Dans les exercices suivants, il s'agit d'utiliser une fonction qui comporte une boucle et un paramètre permettant de modifier l'effet de l'exécution de cette boucle.

Les tables de multiplication

Étant donnée la fonction

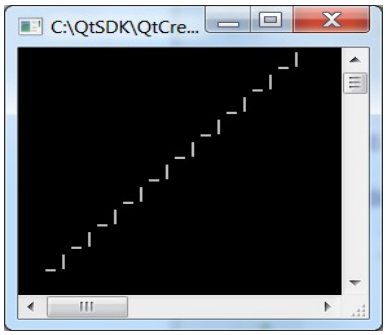
```
void ligneTable(int facteur)
{
int colonne(1);
do {
   int aAfficher = facteur * colonne;
   if(aAfficher < 10)
     std::cout << ' ';
   std::cout << aAfficher << ' ';
   ++colonne;
   } while(colonne < 10 );
std::cout << "\n";
}
```



donnez à votre fonction `main()` un contenu qui permettra au programme d'afficher les tables de multiplication.

L'escalier

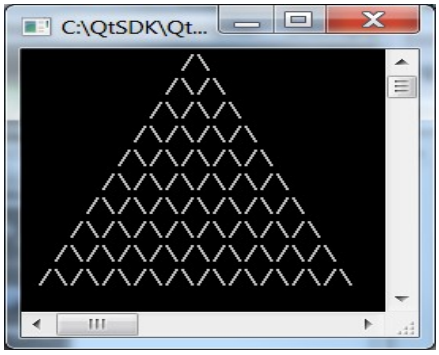
En vous inspirant du programme précédent, créez un programme affichant la figure représentée ci-contre.



> Remarque : le dessin est obtenu en affichant sur chaque ligne la séquence "_|". La seule chose qui change, c'est le nombre d'espaces qui précèdent cette séquence.

Le château de cartes

En vous inspirant du programme précédent, créez un programme affichant ceci :



Ce dessin est obtenu en répétant sur chaque ligne la séquence "/\". Les lignes diffèrent les unes des autres de deux façons : le nombre d'espaces par lequel elles commencent, et le nombre de séquences "/\" qu'elles comportent ensuite.

> Attention, la barre oblique inverse (\, alias "antislash") a une signification spéciale dans le texte littéral en C++ : c'est le caractère qui indique que le caractère suivant est à interpréter et non à prendre "au pied de la lettre". Ainsi, "\n" ne correspond pas à un antislash suivi d'un n minuscule, mais à un passage à la ligne.
>
> Pour dessiner la pyramide, il vous faut donc quelque chose qui va apparaître sous la forme "/\\" dans votre texte source, le redoublement de l'antislash permettant d'obtenir son interprétation littérale.