Exemples pour débutants de petits programmes écrits en C compilés et testés avec Visual Studio 2010

Programme n°1 « Hello World »

La tradition veut que le premier programme que l'on écrive en C pour tester les outils de développement (ici Visual studio) est un programme de quelques lignes qui se contente d'afficher à l'écran le message « Hello World ».
Ce premier essai nous permet de vérifier que tous les outils sont bien installés et de s'assurer que nous sommes capables d'écrire un petit message à l'écran.

1° Lancer Visual Studio

2° Activer dans le menu Fichier la commande Nouveau Projet Ctrl+Maj+N

3° Ce sera un projet en C++ plus précisément une Application console Win32
Entrez le nom du projet puis sans choisir d'autres options particulières cliquez sur le bouton Terminer

4° On voit apparaitre un petit fichier texte de moins d'une dizaine de lignes

```
// Hello.cpp : définit le point d'entrée pour l'application console.
//
#include "stdafx.h"
int _tmain(int argc, _TCHAR* argv[])
{
	return 0;
}
```

Ces lignes ont été générées automatiquement par VisualStudio.

- Les deux premières précédée par `//` sont des lignes de commentaire.
- La ligne `#include "stdafx.h"` sera nécessaire au compilateur pour lui annoncer les fonctions qu'il sera susceptible de rencontrer dans une application console.
- La ligne `int _tmain(int argc, _TCHAR* argv[])` est la première ligne de la fonction principale (en anglais : *main*) du programme. La syntaxe de cette ligne peut varier d'un environnement de développement à l'autre. La syntaxe de celle-ci est celle qui convient à cette version de VisualStudio. Et puisqu'il génère cette ligne automatiquement, sa syntaxe ne devrait pas nous tracasser mais en gros voici ce que cela signifie : `int` indique que la fonction principale, autrement dit l'application, retournera un nombre entier. C'est ce nombre que l'on pourra tester par la variable d'environnement ERROLEVEL.

5° Passons à la programmation proprement dite. Nous allons ajouter une ligne d'instruction pour afficher le message "Hello Word".

```
int _tmain(int argc, _TCHAR* argv[])
{
	printf("Hello world");
	return 0;
}
```

Nous reviendrons plus tard sur la fonction `printf`. On se contente ici de lui passer la chaine de caractères à afficher comme seul argument.

6° Chercher dans la barre de menu la commande Générer la solution et lancer cette commande.
Les éventuelles erreurs s'afficheront dans le bas de l'écran. Corrigez-les s'il y a lieu et finalement un message doit vous signaler que la génération du programme a réussi.

7° Tester le programme.

- Soit en mode invite de commande : se mettre dans le dossier où se trouve l'exécutable Hello.exe. Puis taper la commande Hello.
- Soit tester ce programme en pas à pas avec la commande Debug

Programme n°2 : Somme de deux nombres

Cet exemple nous montre comment :

- déclarer des variables globales,
- définir une fonction et voir comment lui passer des arguments
- demander à l'utilisateur de saisir un nombre entier décimal
- appeler une fonction écrite par nous.

Nous y ferons aussi la distinction entre variables globales et variables locales

```
1     // Somme.cpp : définit le point d'entrée pour l'application console.
2     //
3     #include "stdafx.h"
4
5     int a,b;
6
7     int somme( int x, int y)
8     {
9           return x+y;
10    }
11
12    int _tmain(int argc, _TCHAR* argv[])
13    {
14          int n;
15
16          printf("Entrez la premier nombre: ");
17          scanf_s("%d", &a);
18          printf("Entrez le second nombre: ");
19          scanf_s("%d", &b);
20          n = somme(a, b);
21          printf("La somme de %d et %d est %d",  a, b, n);
22
23          return 0;
24    }
```

Ligne 5 Déclaration de deux variables globales. Deux nombres entiers qui seront vue de partout dans le module.

Ligne 7 Définition de la fonction `somme()`
Ses arguments sont deux nombres entiers dont les noms x et y n'ont pas de portée en dehors de la fonction somme. Ces arguments x et y recevront une copie provisoire des valeurs respectives de a et b lors de l'appel de la fonction ligne 20

Ligne 9 Retour de la valeur calculée. La fonction est volontairement très simple.

Ligne 12 Appel de la fonction principale du programme `_tmain()`

Ligne 14 Déclaration d'un variable locale à main. Ce nombre entier `n` n'est vu qu'à l'intérieur du bloc commençant avec l'accolade ligne 13 et se terminant par l'accolade ligne 24

Ligne 16 Appel de la fonction `printf()` pour afficher un simple message

Ligne 17 Appel de la fonction `scanf_s("%d",&b)` pour capturer la saisie d'un nombre décimal au clavier. L'argument `"%d"` indique qu'on attend un nombre entré en décimal. L'argument `&a` est l'adresse de la variable a. C'est là que la fonction `scanf_s()` doit mémoriser le nombre saisi.

Ligne 20 Appel de la fonction somme avec deux arguments a et b

Ligne 21 Utilisation de la fonction `printf()` avec cette fois 3 variables à afficher. La chaîne de caractères passée comme premier argument de la fonction sert à composer le message en indiquant qu'il y a 3 nombres à afficher en décimal → %d

Ligne 23 Le programme rend la valeur 0. Comme s'il n'y avait aucune erreur.
A vrai dire aucun test n'a été fait à ce point de vue.

Programme n°3 : Boucle pour afficher 10 valeurs consécutives

```
1   // Compte.cpp : définit le point d'entrée pour l'application console.
2   //
3
4   #include "stdafx.h"
5
6   int _tmain(int argc, _TCHAR* argv[])
7   {
8      int i = 1;
9      while ( i <= 10)
10     {
11        printf("N = %d", i);
12        i = i + 1;
13     }
14     return 0;
15  }
16
```

Ce bout de programme est un prétexte pour monter comment faire une boucle de compter jusqu'à 10 et d'écrire ces 10 premiers nombres à l'écran :

```
N = 1
N = 2
N = 3
N = 4
N = 5
N = 6
N = 7
N = 8
N = 9
N = 10
```

La boucle while

Les instructions contenues dans cette boucle sont répétées tant que la condition, ici i inférieur ou égale à 10 est vérifiée.
La valeur initiale de la variable i a été définie à la ligne 8 lors de la déclaration de l'entier i
Cette variable i est incrémentée à la ligne 12 par l'instruction `i = i + 1`
En C cette même instruction aurait pu s'écrire `i += 1` ou encore plus succinctement `i++`

Deux autres variantes de ce programme montrent comment faire ces mêmes boucles avec une boucle **`do ... while`** ou une boucle **`for`**

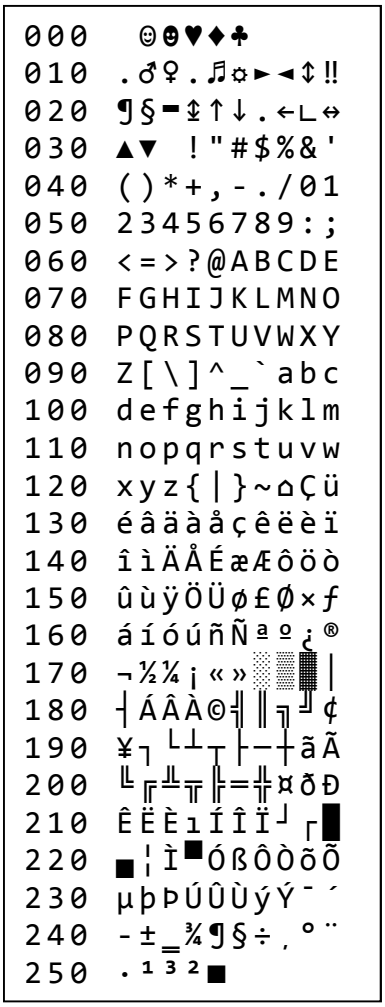
```
1   // Compte.cpp
2   //
3
4   #include "stdafx.h"
5
6   int _tmain(int argc, _TCHAR* argv[])
7   {
8      int i = 1;
9      do
10     {
11        printf("N = %d \n", i);
12        i = i + 1;
13     } while ( i <= 10)
14     return 0;
15  }
```

```
1   // Compte.cpp
2   //
3
4   #include "stdafx.h"
5
6   int _tmain(int argc, _TCHAR* argv[])
7   {
8      int i;
9      for ( i=1; i <= 10 ; i++)
10     {
11       printf("N = %d \n", i);
12     }
13     return 0;
14  }
15
```

Bien que la boucle **`for`** soit un peu moins facile à comprendre au premier abord, c'est la forme de boucle qui est généralement préférée. L'instruction **`for`** regroupe dans une même ligne, l'initialisation de la variable i, la condition d'exécution de la boucle et l'instruction i++ qui fait avancer le compteur.

Programme n°4 : Affichage de la table des codes ASCII

Le bout de programme proposé ici génère une liste de 256 caractères tels qu'on les voyait affichés à l'époque du DOS ou encore actuellement en mode invite de commande.
Le résultat ressemblera au tableau ci-contre :




```
// ListeCodesASCII.cpp
// Auteur : Luc De Mey
// Original : 4 novembre 2011

#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
    int i;
    int col = 0;

    printf("%03u ", 0);
    for ( i = 0; i < 256; i++)
    {
        if (i== 0x1A || i== 0xA || i == 0xD)
        {  // Mettre des points à la place
           // des caractères non imprimables !
           putc('.', stdout);
        }
        else
        {
           putc( i, stdout);
        }
        col++;
        if (col == 10)
        {
           printf("\n%03u ", i+1);
           col = 0;
        }
    }
    return 0;
}
```