Les Constantes - Les variables – Les itérations

Exercice 1

Rappel :

```
// La fabrication d'un programme compilé passe par les étapes suivantes :
// - Création du 'code source' en langage C dans les fichiers .c et .h à l'aide d'un éditeur de texte
// - Génération du programme 'exécutable' en deux opérations simultanées ou non à l'aide du
//   compilateur CL :
//      * La compilation fabrique un fichier 'objet' dont l'extension est '.obj'
//      * L'édition de liens (link) qui fabrique le fichier correspondant au programme exécutable.
// A chaque modification du code source, il est nécessaire de réaliser à nouveau la sauvegarde, la
// compilation, puis le link une fois les problèmes résolus lors de la phase de compilation.
// L'exécution du programme est alors possible en tapant simplement le nom du programme dans un shell
// de commande (appelé aussi Command Prompt).
```

Dans cet exercice, il s'agit de découvrir les étapes basiques et manuelles qui permettent de fabriquer un programme exécutable de façon non automatisée, sans créer de script de fabrication et sans utiliser d'environnement de développement intégré (IDE) :

- Vérifier que Windows est démarré et que le logiciel Visual Studio Enterprise est installé avec toutes les options, ou au minimum, avec le support du langage C/C++.
- A partir de l'interface Modern UI, démarrer un Shell de commandes Visual Studio 2015 : *Developer Command Prompt for VS2015* (dans le groupe de programmes Visual Studio 2015).
- Créer un répertoire de travail sur le disque D en tapant les commandes suivantes (appuyer sur la touche de validation après chaque commande) et en respectant les espaces.

```
D:
cd  \
md  tpc
cd  tpc
md  .\BIN
md  .\SRC
md  .\OBJ
dir
```

La commande `D:` permet de changer le lecteur courant
La commande `cd` permet de changer de répertoire *courant* (change directory)
La commande `md` permet de créer un répertoire (make directory)
La commande `dir` permet d'afficher le contenu d'un répertoire

- Démarrer un éditeur de texte afin de créer un nouveau programme source *test.c* dans le répertoire *SRC* en tapant la commande :

```
notepad  .\SRC\test.c
```

Ce programme va contenir le point d'entrée du programme : la fonction *main()*, *main* signifiant *principal(e)* en anglais.

- Modifier le fichier *test.c* avec le code C suivant :

```c
// inclusion des fichiers de déclarations utiles

#include <stdlib.h>
#include <stdio.h>


main() { // accolade ouvrante = début de bloc d'instructions


    // les instructions du programme principal : main()
    /********** Debut de programme ****************/

    printf("\nMon premier programme ...\n");

    /********** Fin de programme *****************/

    return(EXIT_SUCCESS);

} // accolade fermante = fin de bloc d'instructions
```

- Supprimer tous les fichiers sauf les fichiers sources (pas obligatoire) :

```
del /S  *.obj  *.exe
```

- Création de l'exécutable en deux étapes : La **compilation** puis le **link**.
  taper la commande : (␣ indique un caractère d'espace, sinon ne pas mettre d'espace)

```
cl ␣ /c ␣ /W3 ␣ /Fo.\OBJ\test.obj ␣ .\SRC\test.c
```

  /c : compilation seulement , pas de link
  /W3 : Warning level 3
  /Fo : nom et chemin du fichier objet : .\OBJ\test.obj
  .\SRC\test.c : nom et chemin du fichier source

  Information sur les options du compilateur :
  https://msdn.microsoft.com/en-us/library/fwkeyyhe.aspx

  taper la commande :

```
link ␣ .\OBJ\test.obj ␣ /OUT:.\BIN\test.exe
```

- Exécution du programme :

  .\BIN\test

Utilisation de l'IDE Visual Studio : (traduire les options de menu en français au besoin et suivant l'installation que vous avez choisie pour Visual Studio)

Utiliser *l'environnement de développement intégré Visual Studio* pour découvrir le langage *C* et comprendre de quelle manière il manipule les données.
En cas de doute sur le comportement de Visual Studio pour la mise en place d'un projet en langage C, il est possible de réinitialiser l'IDE avec une configuration par défaut adaptée à ce langage. Pour Cela, il existe une option « import and export settings .. » dans le menu « Tools ».

Organisation :

Créer une **solution** pour l'ensemble du TP et y ajouter un nouveau **projet** pour chaque exercice. Le projet sera une «Applications console Win32 » en langage C :
Menu: File / New / Project / Other Project types / Visual Studio Solutions : nommer le projet *TP1* et cliquer sur OK.

La fenêtre ***Solution Explorer*** affiche la solution « *TP1* » et les projets contenus dans la solution. Pour ajouter le projet nommé *Exo1* : Faire un clic droit sur « Solution TP1 » et sélectionner **Add** .. dans le menu, puis sélectionner **New Project ..**
Pour choisir le langage C, aller dans « Visual C++/Win32/Win32 Console Application » et donner un nom comme *Exo1* à votre projet. Dans l'assistant, vérifier les choix et cliquer sur *Next*. Sur la page suivante de l'assistant cocher **Empty Project**, puis cliquer sur *Finish*.

La fenêtre ***Solution Explorer*** affiche sous la solution « *TP1* » une arborescence de projets et de dossiers dont l'un se nomme « Source File ». C'est dans ce dossier qu'il faut créer le programme source en langage C. Faire un clic droit sur le dossier «Source File» et sélectionner « **Add / New Item** » afin de créer un Fichier C (choisir le modèle correspondant C++ File).

Attention : Pour écrire du code C, et non du « code C++ », forcer le nom du source avec l'extension **.c** en tapant « **MyProg.c** » dans le champ « Nom ».

Par la suite, pour faire un nouvel exercice, créer un nouveau projet (ex Exo2) et le définir **As startup Project** (clic droit sur le nom du projet) avant de compiler et d'exécuter le programme correspondant à l'exercice en cours.

Remarques :

- Votre programme doit respecter les recommandations liées à la réalisation d'un code source, clair et structuré dans sa présentation, commenté de façon pertinente, comportant des noms de variables significatifs, être modulaire et réutilisable.
- Chaque variable a un type et une visibilité adaptés à son usage et est initialisée avec une valeur adaptée à VOTRE algorithme. En langage C, la valeur par défaut d'une variable n'est pas définie.
- Les instructions du langage C 90, telles qu'elles sont proposées dans votre manuel sont souvent dites « deprecated » ou « dépréciées ». Une version plus sécurisée de ces commandes est utilisée par Visual Studio. Toutefois, on peut forcer l'utilisation des anciennes commandes en mettant en tout début de fichier source, la commande suivante :

```
#define _CRT_SECURE_NO_WARNINGS
```

Aide et approfondissement : Vous disposez de l'aide locale ou en ligne de Visual Studio, et aussi, de votre manuel « Programmer en Langage C ».

Exercice 2

Rappel :

```
// le type 'char' caractérise un entier positif codé sur 8 bits compris entre -128 et +127,
// le bit de poids fort sert au codage du signe.
// un 'unsigned char' pourra donc représenter un nombre positif compris en 0 et 255,
// soit 256 valeurs.
// La représentation en hexadécimale de 255 est FF (notée 0xFF)
// Le type booléen n'existe pas en C90, une valeur non nulle
// dans une expression logique est comprise comme 'VRAI'.
```

Dans la solution TP1, créer le projet Exo1. Le fichier **source** à utiliser se nomme ***main.c*** :

```
// inclusion des fichiers d'entêtes (header) d'extension .h

#include <stdlib.h>
#include <stdio.h>

// définition de macros
// #define DEBUG
// programme principal

// définition de type

typedef unsigned char boolean ;
typedef unsigned char byte ;

main() { // accolade ouvrante = début de bloc d'instructions
        // Déclaration de variables réservées au main(), et initialisations facultatives
        char OneLetter = 'A';
        char AnOtherLetter = 66;
        byte toto = 0;
        unsigned char VerySmallCounter = 255;
        short SmallCounter = 32767;
        int Counter = 0;
        long LargeCounter = 2147483647;
        unsigned long FullRangeLargeCounter = 4294967295;
        int ChienDeGarde = 0;
        float Dim1 = 1.414F;
        double Dim2 = .5L;
        boolean TestResult = 1;
        // Déclaration de constante
        const double Pi = 3.14159265L;
        // les instructions du programme principal : main()
        /********** Debut de programme ****************/
        printf("\n");
        printf("exemple d'affichage d'un caractere %c, codage sur %d octets\n",OneLetter, sizeof(OneLetter));
        printf("exemple d'affichage d'un entier court %d, codage sur %d octets\n",Counter, sizeof(Counter));
        /********** Fin de programme ****************/
        system("pause");
        return(EXIT_SUCCESS);
} // accolade fermante = fin de bloc d'instructions
```

- Compléter le code par des appels à la fonction d'affichage « printf » pour afficher la valeur de chacune des variables dans un format correct. Attention, le choix du filtre « % », impacte le résultat car il réalise une conversion de la donnée avant son affichage dans le format souhaité. Donc, ce qui s'affiche est ce qui a été demandé, et pas nécessairement la donnée dans son format d'origine.
- Donner pour chaque type, le nombre d'octets utilisés pour le codage, et indiquer quelle valeur minimum et quelle valeur maximum (dynamique) on peut représenter avec une variable qui aurait ce type. Donner la réponse sous la forme de commentaires dans votre fichier source.

Références :

Noter que le manuel de référence du langage indique dans quel fichier *.h* une fonction est déclarée. Il faut donc ajouter une directive #include <stdio.h> en début de code source. En effet, d'après l'information suivante https://msdn.microsoft.com/en-us/library/vstudio/wc7014hz.aspx , on découvre que la fonction **printf** requière **stdio.h** (section Requirements).

Liens d'aide pour les formats d'affichage avec printf :

http://msdn.microsoft.com/en-us/library/vstudio/56e442dc.aspx
http://msdn.microsoft.com/en-us/library/vstudio/hf4y5e3w.aspx

Quels sont les types qui ne sont pas natifs* du langage C (version 90), à partir de quels autres types sont-ils définis ici ?
*natif : Qui a été créé de façon originale dans le langage.
La taille réellement occupée pour le codage de ces types de données peut varier en fonction du compilateur et de l'architecture de la machine.

Type	Taille en octets	Valeur Minimum	Valeur Maximum	Natif (oui/non)	Défini à partir de : (si pas natif)
char					
byte					
unsigned char					
short int					
int					
long					
unsigned long					
float					
double					

Rappel :

```
// Une variable est caractérisée par son nom (ou identifiant), par le type de donnée quelle peut
// recevoir  et un emplacement en mémoire où elle stocke ses données.
// L'emplacement des données en mémoire s'appelle l'ADRESSE
// En C, on fait référence à l'adresse d'une variable à l'aide du caractère '&'
//
// Par exemple, si on déclare la variable entière Nombre :
// int Nombre;
// Nombre = 12;
// alors &Nombre est l'adresse en mémoire de cette variable qui contient la valeur 12.
// L'adresse est une valeur entière codée sur 64 bits dans un environnement 64 bits.
// Il est probable que deux variables déclarées l'une à la suite de l'autre aient des adresses
// mémoire très proches.
```

- Compléter le code en déclarant en une seule instruction les trois variables A, B et C de type *int*. Les variables A, B et C sont des variables déclarées **localement** au main(), donc à l'intérieur des accolades du main() { }.

- Donner une valeur initiale aux variables A, B et C en les assignant respectivement des valeurs constantes 1, 2 et 3.
- En une seule instruction, afficher les valeurs en décimal et l'adresse mémoire en notation hexadécimale de ces trois variables, utiliser '**\n**' pour afficher les valeurs sur trois lignes différentes. Que penser de ces trois adresses quand on les compare ?
  Remarque: l'option de formatage pour afficher une valeur entière en hexadécimale est %X

Exercice 3

Rappel :

```
// Les ordinateurs sont des machines conçues pour faire des opérations répétitives, très rapidement.
// La manière dont se déroulent ces opérations est contrôlées par un algorithme.
// Les boucles sont des algorithmes qui exécutent un groupe d'instructions un certain nombre de fois.
// Il se peut que l'on connaisse le nombre d'itérations (boucles) avant de commencer le traitement, ou
// que l'on réalise des itérations jusqu'à ce qu'un événement se produise. Cela dépend du contexte
// et des informations dont on dispose.
// Par exemple, monter les 20 marches de cet escalier et s'arrêter, ou, monter les marches jusqu'à
// atteindre le 1er étage et s'arrêter.
// Le premier cas est le plus simple à traduire, on connait une valeur de début et une valeur de fin.
// De plus, l'incrément, c'est-à-dire la valeur qu'on ajoute pour passer au suivant est ici de 1
// (on monte une marche à la fois).
// Idéalement, on implémente (traduction dans le langage de programmation) une boucle FOR :
// Ce qui donne en pseudo-langage,
// Pour NoMarche allant de 1 à 100
// faire
//      MonterMarche
// fait.
//
// et en langage C,
// for (NoMarche=1 ; NoMarche <= 100 ; NoMarche++ )
//      {
//      MonterMarche();
//      }
//
// NoMarche = 1 : Donne une valeur initiale à la variable qui indique le numéro de marche
// NoMarche <= 100 : Condition à vérifier pour continuer à boucler
// NoMarche++ : incrémenter la variable NoMarche de 1 après avoir utilisé sa valeur.
// Remarque : ne pas confondre a=1 avec a==1
// dans le 1 er cas 'a' prend la valeur 1, dans le second il s'agit d'une comparaison.
```

- Dans un nouveau projet **Exo2**, ajouter le code C permettant d'afficher, en les séparant par une espace, tous les nombres entiers compris entre 0 et 100 inclus.
- Ajouter au programme le code C permettant d'afficher une table des codes ASCII pour les codes compris entre 32 et 121, en les répartissant en sept lignes de quinze caractères.
- Afficher la liste décroissante des valeurs comprises entre 20 et 0, par pas de 0.5. Utiliser une notation décimale avec un chiffre après la virgule.
- Pour chacune de ces trois boucles, ajouter l'affichage de la valeur de la variable d'itération (parfois appelée *compteur*) avant et après la boucle.
- Afficher la table des sinus des angles allant de 0 à 90 degrés en faisant varier l'angle de 10 en 10 degrés. Quel fichier d'entêtes *.h* doit-on utiliser ?