

SI4 : Introduction à la programmation



Microsoft
.NET

M-Hicham IGHILAZA

Table des matières

1	Généralités.....	2
1.1	Introduction	2
1.2	Architecture d'un ordinateur	2
1.3	Les différents langages informatiques	3
2	Programmation avec le langage C#.....	6
2.1	Le langage C#.....	6
2.2	Environnement de développement	6
2.3	L’Affichage en console	7
2.4	Les variables	8
2.5	La saisie au clavier	9
2.6	Les opérations arithmétiques.....	10
2.7	Les structures conditionnelles	11
2.7.1	Conditions if else	11
2.7.2	Exercices.....	12
2.7.3	Conditions : switch case	13
2.8	Les structures itératives	14
2.8.1	Boucle for (pour)	15
2.8.2	Exercices.....	16
2.8.3	Boucle while (tant que)	17
2.8.4	Boucle do ... while (faire ... tant que)	18
2.8.5	Exercices.....	18
2.9	Les tableaux	19
2.9.1	Exercices.....	20
2.9.2	Les tableaux à deux dimensions (matrices).....	20
2.9.3	Exercices.....	20
2.10	Les sous-programmes	21
2.10.1	Définition :	21
2.10.2	Les fonctions	21
2.10.3	Les procédures	25

1 Généralités

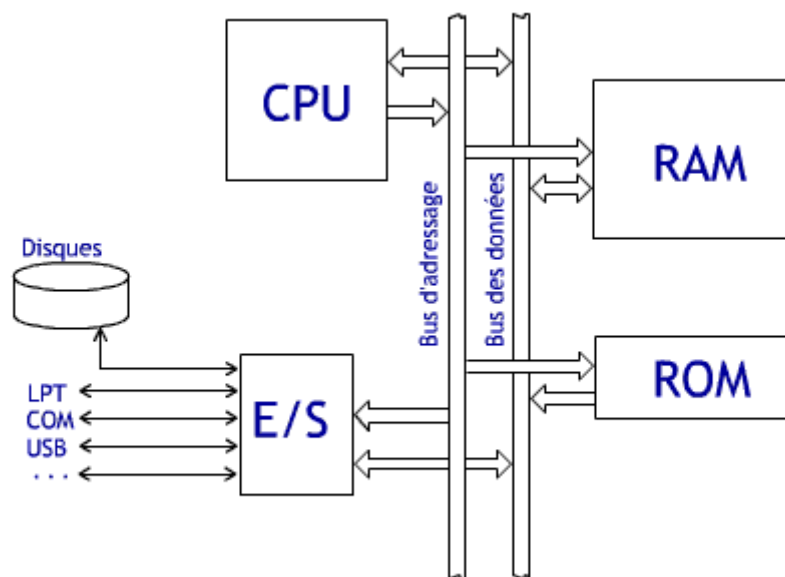
1.1 Introduction

La programmation consiste à faire réaliser un ensemble de tâches plus ou moins complexes par une machine capable de les exécuter.

L'objectif de la programmation est de permettre l'automatisation des tâches répétitives ou complexes réalisées par l'homme et de réduire l'erreur humaine sur certaines tâches critiques. Cela permet d'accroître l'efficacité et la productivité.

1.2 Architecture d'un ordinateur

Afin de bien appréhender les méandres de la programmation, il est important d'avoir quelques notions basiques sur l'architecture d'un ordinateur et sur ce qu'il se passe lors de l'exécution d'un programme.



Un ordinateur dispose de 5 composants principaux :

- **Une mémoire** pour contenir le programme, les données et d'autres informations
- **Un processeur** capable de lire les informations dans la mémoire et les traiter
- **Une horloge** pour faire avancer le programme : sans horloge, le processeur resterait sur la même instruction, indéfiniment
- **Un bus** (et même plusieurs bus) pour permettre à tous les éléments et périphériques de communiquer ensemble. Il s'agit d'un ensemble de pistes (de fils électriques) qui relie la mémoire et le processeur (bus principal) ou d'autres bus pour le clavier, les cartes vidéo, etc.
- **Les ports d'entrées et sorties** afin de relier les périphériques d'entrée sortie tels que le disque dur, l'écran, le clavier etc.

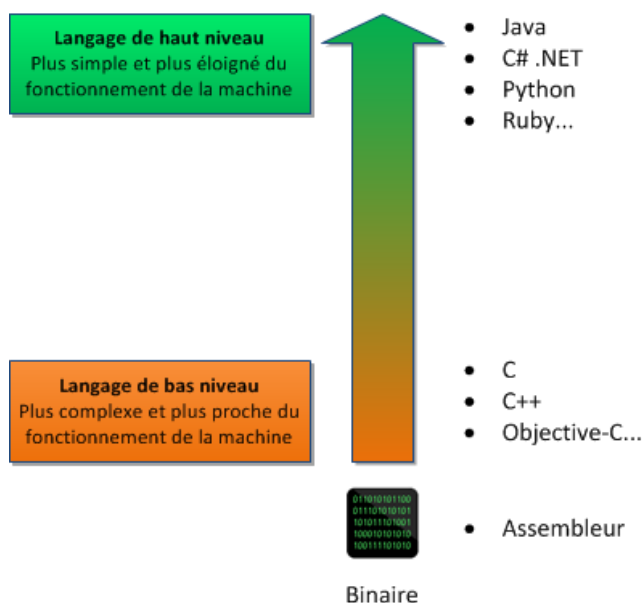
Un programme est initialement stocké sur un périphérique de stockage (disque dur, disque SSD, clé USB ...), au lancement le programme est chargé partiellement ou entièrement dans la mémoire de la machine, pour qu'en suite le processeur exécute les différentes instructions disponibles dans la mémoire.

1.3 Les différents langages informatiques

Les langages sont classés selon deux catégories, on dit qu'ils sont de bas niveau ou de haut niveau.

Un langage dit de haut niveau est un langage assez éloigné du binaire (et donc du fonctionnement de la machine), qui permet généralement de développer de façon plus souple et rapide.

Par opposition au langage de haut niveau, un langage de bas niveau est plus proche du fonctionnement de la machine : il demande en général un peu plus d'efforts mais vous donne aussi plus de contrôle sur ce que vous faites.



Le langage machine :

Le langage machine est une suite de bits(0/1) qui composent des instructions pouvant être exécutées par le processeur, tout processeur possède son propre langage machine.

L'assembleur est un langage de plus bas niveau qui représente le langage machine sous une forme lisible par l'homme.

Plus langage utilisé est proche du langage machine plus la performance est élevée.

Les langages compilés :

Les langages de programmation compilés sont des langages de haut niveau, qui comportent une syntaxe proche du langage humain ce qui facilite la tâche de développement. Ce langage est ensuite compilé à l'aide d'un compilateur.

Un compilateur est un logiciel permettant de traduire un langage de programmation de haut niveau en langage machine et de générer un fichier exécutable afin d'exécuter son programme.

```
34 int ajout_score_piece_posee(int compteur,int val)
35 {
36     if (val==1 )compteur=compteur+10;
37     else if (val>=2 && val<=3) compteur=compteur+20;
38     else if (val>=4 && val<=9) compteur=compteur+30;
39     else if (val>=10 && val<=16)compteur=compteur+40;
40     else if (val>=17 && val<=26)compteur=compteur+50;
41     else compteur=compteur+90;
42
43     return compteur;
44 }
```

L'inconvénient de ce genre de langages est qu'il faut autant de compilateurs différents qu'il n'y a de processeurs ou de systèmes d'exploitation. Afin de pouvoir utiliser son programme sur des plateformes différentes il fallait le compiler pour chaque plateforme.

Les langages interprétés :

Les langages interprétés sont des langages qui ne sont pas traduits en langage machine pour pouvoir être exécutés, mais ils sont directement exécutés par un interpréteur. Il existe plusieurs types d'interpréteurs, il y a par exemple les machines virtuelles .NET ou JAVA Runtime Environnement qui permettent d'exécuter les programmes développés sur n'importe quelle machine à condition que celle-ci comporte la machine virtuelle.

Un autre cas d'utilisation des langages interprétés est l'utilisation langages web tel que HTML ou CSS, qui ne sont pas compilés mais directement interprétés par le navigateur de manière séquentielle.

Quelques langages de programmation populaires :

Langage	Type	VM
Pascal, Fortran , Delphi	Compilé	non
C / C++	Compilé	non
Java	Interprété	oui
C# (Csharp)	Interprété	oui
PHP	Interprété	Langage principalement utilisé pour le développement web. Nécessite un serveur (Apache, IIS, NGINX...)
Python, Ruby, Perl	Interprété	Langages de scripting dans certaines applications.
HTML, JavaScript	Interprété	Langages principalement utilisés pour le développement web. Ils sont interprétés par les navigateurs.

2 Programmation avec le langage C#

2.1 Le langage C#

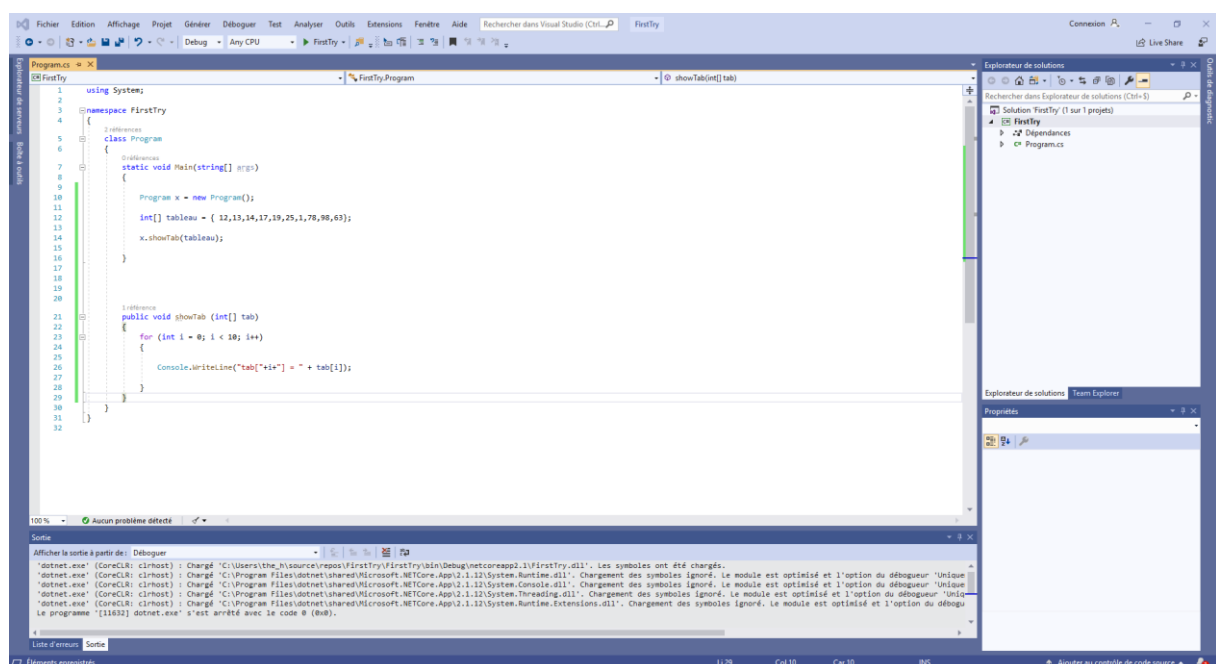
Le C# prononcé “C sharp” est un langage de programmation orienté objet. Il est commercialisé par la société américaine Microsoft depuis 2002 et sert à développer sur la plateforme .NET (prononcé “dot net”). Comme son nom l’indique, ce langage de programmation est directement dérivé du langage C++. Par ailleurs, il est très proche du langage Java, il reprend notamment les principaux concepts ainsi que la syntaxe en y ajoutant certaines notions (surcharge des opérateurs, délégués, indexeurs...). Si le langage utilisé seul reste assez limité, l’utilisation de celui-ci complété par le Framework .NET offre de nombreuses possibilités (création et ouverture de fenêtres, accès réseau, utilisation des bases de données).

Ce langage peut aussi être utilisé pour créer des applications web en utilisant la plateforme ASP.NET. C# est actuellement une compétence très appréciée en entreprise et se présente de plus en plus comme un concurrent du langage Java.

2.2 Environnement de développement

Afin de commencer à développer il n’y a pas besoin d’outil particulier dès lors que l’on dispose d’un compilateur ou d’une machine virtuelle. Il est possible de rédiger son code simplement à l’aide d’un éditeur de texte. Néanmoins il existe des logiciels appelés environnements de développement ou IDE (integrateddevelopmentenvironment), qui offrent un certain nombre d’outils permettant de simplifier certaines tâches du développeur, de gagner en rapidité et de réduire les erreurs. Parmi ces outils il y a la coloration du code, la signalisation des erreurs de syntaxe, l’organisation des projets ...

La figure suivante représente l’IDE Microsoft visual studio.

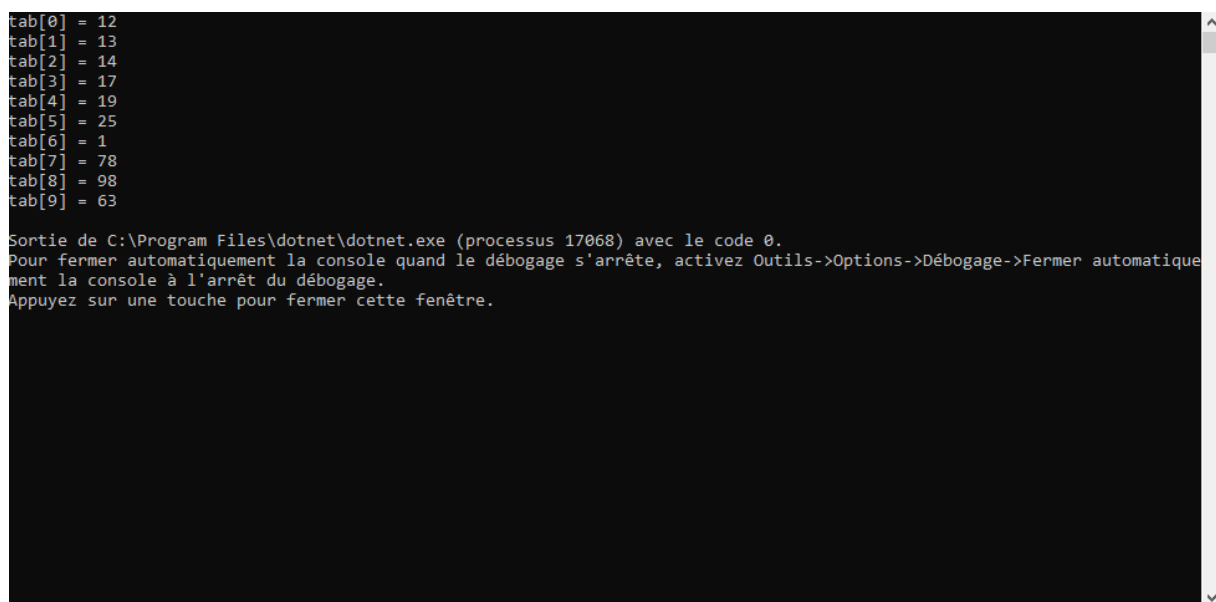


Un programme en C# est composé de classes comprenant un ensemble d'instructions (ordres donnés à la machine) et de sous programmes appelés fonctions ou méthodes. Toute instruction se termine avec un point-virgule « ; », une instruction peut être écrite sur plusieurs lignes.

La méthode qui a pour signature « `static void Main(String[] args)` » représente le point de départ du programme. Tout programme doit contenir une méthode `Main` afin de pouvoir être exécuté.

2.3 L’Affichage en console

Avant l’apparition des Interfaces graphiques, l’affichage sur les ordinateurs se faisait en mode texte sur ce qui est appelé une console.

A screenshot of a console window with a black background and white text. The text shows an array 'tab' with 10 elements, their values, and a message about exiting the application. The values are: tab[0]=12, tab[1]=13, tab[2]=14, tab[3]=17, tab[4]=19, tab[5]=25, tab[6]=1, tab[7]=78, tab[8]=98, tab[9]=63. The message says: 'Sortie de C:\Program Files\dotnet\dotnet.exe (processus 17068) avec le code 0. Pour fermer automatiquement la console quand le débogage s\'arrête, activez Outils->Options->Débogage->Fermer automatiquement la console à l\'arrêt du débogage. Appuyez sur une touche pour fermer cette fenêtre.'

Afin de donner l’ordre à la machine d’afficher quelque chose sur la console en C#, il existe l’instruction `Console.WriteLine(Le contenu à afficher est placé ici);`

2.4 Les variables

Une variable correspond à un espace réservé dans la mémoire de la machine et sert à enregistrer temporairement une valeur.

C# est un langage que l'on dit typé, c'est-à-dire qu'il faut obligatoirement spécifier le type de la variable lors de sa déclaration.

Il existe plusieurs types de variables, chaque type occupe un espace mémoire de taille différente et peut recevoir des valeurs de types différents (numérique, alphabétique, alphanumérique, booléen).

Type C#	Signification	Taille (en octets)	Domaine de valeurs
char	character (caractère)	2	Représente un caractère
string	Chaîne de caractères	variable	Référence sur une séquence de caractères Unicode
int	integer (nombre entier)	4	Entier de -2147483648 à 2147483647
uint	unsignedinteger (nombre entier non signé)	4	0 à 4294967295
long	nombre entier long	8	Entier de -9223372036854775808 à 9223372036854775807
ulong	unsigned long (nombre entier long non signé)	8	0 à 18446744073709551615
sbyte	signed byte (octet signé)	1	-128 à +127
byte	octet	1	Entier de 0 à 255
short	nombre entier court	2	Entier de -32768 à 32767
ushort	unsigned short (nombre entier court non signé)	2	0 à 65535
float	flottant (nombre réel)	4	de -3,402823e38 à 3,402823e38
double	double flottant (nombre réel)	8	de -1,79769313486232e308 à 1,79769313486232e308
decimal	nombre décimal	16	Nombre décimal
bool	booléen	1	true / false
object	référence d'objet	variable	référence d'objet

Une variable a un type un nom et peut prendre une valeur.

- Le nom d'une variable commence toujours par une lettre alphabétique minuscule.
- Il est préférable d'éviter d'utiliser les accents pour nommer les variables.
- Deux variables ne peuvent avoir le même nom.

Déclaration d'une variable :

`int a ;` `int` représente le type de la variable et `a` représente le nom de la variable ;

`String unMot ;`

`bool vraiFaux ;`

Déclaration avec affectation d'une valeur

`int a = 10 ;` « enregistrer la valeur 10 dans la variable a »

`String unMot = "un texte" ;`

`char unCaractere = 'z' ;`

Le symbole `=` n'exprime pas l'égalité, il se lit reçoit et permet d'enregistrer une valeur dans une variable.

2.5 La saisie au clavier

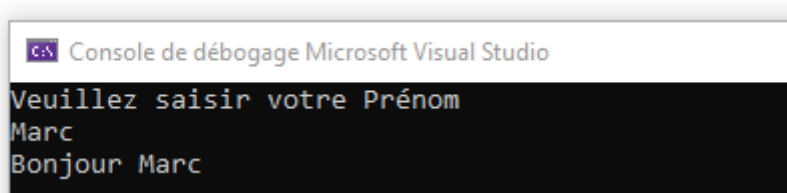
L'instruction `Console.ReadLine()` ; permet de demander à la machine d'attendre qu'une saisie soit faite au clavier. Cela permet de demander à l'utilisateur d'insérer au clavier les données qui seront traitées par le programme.

Exemple :

Code :

```
1  using System;
2
3  namespace Addition
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              String prenom;
10
11              Console.WriteLine("Veuillez saisir votre Prénom");
12              prenom = Console.ReadLine();
13              Console.WriteLine("Bonjour " + prenom);
14          }
15      }
16  }
17
18 }
```

Résultat :



The screenshot shows the 'Console de débogage Microsoft Visual Studio' window. It displays the program's execution: it prompts 'Veuillez saisir votre Prénom', the user enters 'Marc', and the program outputs 'Bonjour Marc'.

2.6 Les opérations arithmétiques

C# permet d'exécuter toutes les opérations arithmétiques de base, à savoir :

Opération	Symbole C#	Exemple
L'addition	+	int z = a + b ;
La soustraction	-	int resultat = 5 - 3 ;
La multiplication	*	int z = a * b ;
La division	/	int z = a / b ;
Le modulo (reste de la division entière)	%	int z = a % b ;

Afin d'afficher la valeur contenue dans la variable z sur la console, il suffit d'utiliser l'instruction « Console.Write(z) ; ». Attention : La casse est importante.

Exercices :

Exercice 1

Ecrire programme qui calcule le double d'un entier.

Exercice 2

Écrire un programme qui calcule la circonférence d'un cercle et l'aire du disque délimité par ce cercle.

Circonférence = $2 \pi r$. Aire = $\pi \times r^2$.

Exercice 3

Écrire un programme qui transforme des degrés Celsius en degrés Fahrenheit. Le zéro Celsius correspond à 32 °F; 100 °C égale 212 °F. °C = (°F - 32) / 1,8.

Exercice 4

Écrire un programme qui calcule le prix TTC. TVA = 20%.

2.7 Les structures conditionnelles

Les structures conditionnelles permettent d'exprimer des ordres à exécuter uniquement si les conditions spécifiées sont validées.

2.7.1 Conditions if else

Syntaxe :

```
if (conditions) { Instructions }
```

```
else if (conditions) { Instructions }
```

```
else { Instructions }
```

Symbole	Signification
if	si
else if	Sinon si
else	sinon

Le tableau suivant représente les différents outils de comparaison utilisés pour exprimer une condition :

Outils de comparaison	
>	Strictement supérieure
>=	Supérieure ou égale
<	Strictement inférieure
<=	Inférieure ou égale
==	Égale
!=	Différente de

- *Les conditions sont toujours délimitées par des parenthèses.*
- *Les instructions à exécuter sont délimitées par des accolades. Dans le cas où il n'y a qu'une seule instruction à exécuter alors les accolades ne sont pas obligatoires.*

Dans le cas où l'exécution des instructions dépend de plusieurs instructions ou bien de la négation d'une instruction, il convient d'utiliser les symboles suivants :

Opérateurs logiques	
&&	ET (condition1 est vraie ET condition2 est vraie)
 	OU (condition1 est vraie OU condition2 est vraie)
!	Négation (l'inverse de la condition est vrai)

Exemple : Soit moyenne une variable qui contient la moyenne d'un étudiant

Si la moyenne est > ou égale à 10 alors l'étudiant est admis	if (moyenne >=10) {Console.WriteLine ("Etudiant Admis")}
Sinon si la moyenne est < 10 et > 9 alors décision du conseil pour le rachat	else if (moyenne < 10 && moyenne >=9) {Console.WriteLine ("Décision conseil")}
Sinon l'étudiant est recalé	else {Console.WriteLine ("Etudiant recalé")}

2.7.2 Exercices

Exercice 1

Écrire un programme qui calcule la valeur absolue de la différence de deux nombres.

Exercice 2

Écrire un programme qui dit si un entier est pair ou pas.

Exercice 3

Écrire un programme qui calcule le maximum de deux nombres.

Exercice 4

Écrire un programme qui calcule le maximum de trois nombres.

Exercice 5

Écrire un programme qui inverse les valeurs de deux variables

Exercice 6

Écrire un programme qui permet de redéfinir 3 variables, X, Y, Z de façon à obtenir : $X < Y < Z$

Exercice 7

Ecrire un programme qui permet de calculer le prix d'un certain nombre de photocopies en appliquant le tarif suivant : les 10 premières coûtent 0,10 E pièce, les 20 suivantes 0,08 E pièce et toutes les autres 0,07 E pièce.

Exercice 8

Écrire un programme qui résout dans IR une équation du premier degré en traitant tous les cas possibles (si $a=0$, etc.)

Exercice 9

Écrire un programme qui résout dans IR une équation du second degré en traitant tous les cas possibles (si $a=0$, etc.)

2.7.3 Conditions : switch case

Syntaxe :

Switch (variable)

```
{  
    Case valeur 1 : instructions ;  
        break ;  
    Case valeur 2 : instructions ;  
        break ;  
    case valeur n : instructions ;  
        break ;  
    default : instructions ;  
        break ;  
}
```

Exemple :

```
int telephone = 18;  
switch (telephone)  
{  
    case 15: //Le switch sautera à cette condition si telephone vaut 15  
        Console.WriteLine( "Numéro du samu" ) ; // puis execute cette instruction  
        break; //le break permet de sortir du switch  
    case 17: //Le switch sautera à cette condition si telephone vaut 17  
        Console.WriteLine( "Numéro de la police" );  
        break;  
    case 18: //Le switch sautera à cette condition si telephone vaut 18  
        Console.WriteLine( "Numéro des pompiers" ) ;  
        break;  
    default: //default correspond à tous les autres cas  
        Console.WriteLine( "Numéro inconnu" ) ;  
        break;  
}
```

2.8 Les structures itératives

Jeu :

Dans un jeu de devinette un joueur à trois tentatives pour deviner un nombre entre 1 et 7.

Coder ce jeu en C#

Solution :

```
String motATrouver = "unMot";
bool gagne = false;
Console.WriteLine("Veuillez saisir un mot");
String mot = Console.ReadLine();

if (mot.Equals(motATrouver))
{
    gagne = true;
}
else
{
    Console.WriteLine("Veuillez saisir un mot");
    mot = Console.ReadLine();
    if (mot.Equals(motATrouver))
    {
        gagne = true;
    }
    else
    {
        Console.WriteLine("Veuillez saisir un mot");
        mot = Console.ReadLine();
        if (mot.Equals(motATrouver))
        {
            gagne = true;
        }
    }
}

if (gagne == true) Console.WriteLine("Gagné ! ");
else Console.WriteLine("Perdu !");
```

Afin d'éviter de réécrire les mêmes instructions plusieurs fois, il existe en C# des structures permettant de répéter des instructions sans devoir les réécrire, tant que des conditions sont validées.

2.8.1 Boucle for(pour)

Syntaxe :

```
for ( initialisation; test ; itération )  
{  
    Instructions à répéter  
}
```

Exemple :

```
for (int i = 0; i < 10; i++)  
{  
    Console.WriteLine("Bonjour");  
}
```

Dans l'exemple nous avons une boucle for qui :

- initialise la variable i à la valeur 0 (initialisation).
- ensuite elle passe au test et vérifie si la valeur de la variable i est inférieure à 10.
- si le test est validé, elle exécute l'instruction `Console.WriteLine("Bonjour");`;
- elle passe ensuite à l'itération et incrémente la valeur de i de 1 (i passe de 0 à 1)
- elle répète les tâches b à c jusqu'à ce que le test ne soit plus validé (jusqu'à ce que i devienne supérieur ou égale 10)

Remarques :

`i++` ; correspond à `i = i + 1` ; (incrémenter)

La variable i utilisée dans l'exemple sert de compteur et augmente de 1 à chaque tour de boucle.

Attention à ne pas mettre de `;` à la fin de la ligne commençant par `for`.

Une boucle for est généralement utilisée dans les situations où on connaît le nombre d'itérations nécessaires.

Dans la partie test, il est possible de définir des conditions qui ne sont pas en relation avec le compteur.

La partie itération n'est pas obligatoirement une incrémentation de 1 mais peut être n'importe quelle opération (décrémenter, incrémentation de 2, multiplication ...)

Jeu :

Nous allons maintenant reprendre le jeu de devinettes et refaire le code avec une boucle for :

```
String motATrouver = "unMot";
String mot;
bool gagne = false;
for (int i = 0; i < 10; i++)
{
    Console.WriteLine("Veuillez saisir un mot");
    mot = Console.ReadLine();
    if (mot.Equals(motATrouver)) gagne = true;
}
if (gagne == true) Console.WriteLine("Gagné ! ");
else Console.WriteLine("Perdu !");
```

2.8.2 Exercices

Exercice 0

Écrire un programme qui affiche une table de multiplication par 10.

Exercice 1

Écrire un programme qui calcule la somme des n premiers nombres (on utilisera une boucle).

Écrire une variante qui calcule la somme des n premiers nombres pairs.

Exercice 2

Écrire un programme qui calcule X puissance n, avec n entier positif ou nul. On traitera tous les cas particuliers.

Exercice 3

Écrire un programme pour déterminer si un nombre est premier.

Exercice 4

Écrire un programme itératif qui permet de calculer N !

2.8.3 Boucle while (tant que)

Syntaxe :

```
while(conditions)
{
    Instructions à répéter
}
```

Exemple :

```
int i = 0;
while (i < 10)
{
    Console.WriteLine("Bonjour");
    i++;
}
```

Jeu :

Nous allons maintenant reprendre le jeu de devinettes et refaire le code avec une boucle for :

```
String motATrouver = "unMot";
String mot;
bool gagne = false;
int i = 0;
while (i < 3 && gagne != true)
{
    Console.WriteLine("Veuillez saisir un mot");
    mot = Console.ReadLine();
    if (mot.Equals(motATrouver)) gagne = true;
    i++;
}
if (gagne == true) Console.WriteLine("Gagné ! ");
else Console.WriteLine("Perdu !");
```

Remarque :

La principale différence entre une boucle for et une boucle while est une question de pragmatisme ; on utilise généralement for lorsque le nombre d'itérations est connu, et on utilise while lorsque le nombre d'itérations n'est pas connu à l'avance.

2.8.4 Boucle do ... while (faire ... tant que)

Syntaxe :

```
Do {  
    Instructions à répéter  
}  
while(conditions)
```

La boucle do ... while exécute d'abord une fois les instructions avant de vérifier les conditions.

Exemple :

```
char choix;  
do  
{  
    Console.WriteLine("Veuillez taper 1 ou 2 pour effectuer votre choix");  
    Console.WriteLine("1 - SLAM");  
    Console.WriteLine("2 - SISR");  
    choix = Console.ReadKey(false).KeyChar;  
}  
while (choix != '1' && choix != '2');
```

Avec une boucle while il aurait fallut initialiser la valeur de la variable choix avant la boucle car le test des conditions aurait été exécuté avant les instructions de la boucle.

2.8.5 Exercices

Exercice 5

On désire calculer le terme d'ordre n de la suite de Fibonacci définie par :

$F(0)=0$, $F(1)=1$, $F(n)=F(n-1)+F(n-2)$.

Écrire une fonction itérative (c'est-à-dire pas récursive) qui permet de calculer $F(n)$ et un programme pour la tester.

Exercice 6

Un nombre est dit parfait s'il est égal à la somme de ses diviseurs propres. Par exemple, 6 est parfait, $6=1+2+3$.

- 1) Ecrire un programme qui dit si un nombre est parfait ou pas.
- 2) Ecrire un programme qui, pour une valeur donnée de n, affiche les nombres parfaits $< n$

Exercice 7

Un triplet d'entiers naturels (x, y, z) est dit pythagoricien si et seulement si on a : $x^2 + y^2 = z^2$.

Ecrire un programme qui, pour une valeur de n donnée, affiche tous les triplets pythagoriciens tels que $x^2 + y^2 \leq n$;

Exercice 8

Réaliser un jeu du pendu : un joueur tape le mot à faire deviner et ensuite le deuxième joueur à la possibilité de deviner, une à une, les lettres composants le mot mais n'a droit qu'à 7 erreurs.

Le mot saisi par le premier joueur doit être caché.

2.9 Les tableaux

Jusqu'à présent, nos programmes ont utilisé uniquement des variables stockant une seule valeur, appelées parfois des variables scalaires.

On souhaite écrire un programme qui fait saisir 5 valeurs à l'utilisateur, puis calcule et affiche leur somme. Une solution pourrait être la suivante :

```
Console.Write("Entrez le nombre 1 : ");
int nb1 = Convert.ToInt32(Console.ReadLine());
Console.Write("Entrez le nombre 2 : ");
int nb2 = Convert.ToInt32(Console.ReadLine());
Console.Write("Entrez le nombre 3 : ");
int nb3 = Convert.ToInt32(Console.ReadLine());
Console.Write("Entrez le nombre 4 : ");
int nb4 = Convert.ToInt32(Console.ReadLine());
Console.Write("Entrez le nombre 5 : ");
int nb5 = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Le nombre 1 est " + nb1);
Console.WriteLine("Le nombre 2 est " + nb2);
Console.WriteLine("Le nombre 3 est " + nb3);
Console.WriteLine("Le nombre 4 est " + nb4);
Console.WriteLine("Le nombre 5 est " + nb5);

int somme = nb1 + nb2 + nb3 + nb4 + nb5;
Console.WriteLine("Leur somme vaut " + somme);
```

Il existe cependant une manière différente de faire ce programme en utilisant ce que l'on appelle les tableaux :

Un **tableau** regroupe un ensemble d'éléments de même nature. Il permet de stocker plusieurs éléments partageant le même **type**.

On peut alors créer des tableaux d'entiers, de caractères, ...

Déclaration d'un tableau :

```
int[] tab = new int[10]; (déclaration d'un tableau d'entiers de taille 10)
```

```
char [] tab2 = {'a', 'b', 'c', 'd', 'e', 'f'}; (déclaration d'un tableau de caractères de taille 6, avec des valeurs prédéfinies).
```

Accéder aux éléments d'un tableau :

Les éléments d'un tableau sont indexés de 0 jusqu'à la taille du tableau -1 ;

tab[0] permet d'accéder au premier élément du tableau tab créé précédemment et dont la taille est 10, tab[9] permet d'accéder au dernier élément du tableau. (ex : tab[0] = 5 ; tab [1] = 17 ;)

La méthode length permet d'obtenir la taille d'un tableau. Afin d'afficher la taille du tableau tab on écrit : Console.WriteLine(tab.Length);

Remarque :

- En C#, tous les éléments d'un tableau doivent avoir le même type. On ne peut pas mélanger des entiers, des réels ou des chaînes dans un même tableau.
- La taille d'un tableau est fixe.

2.9.1 Exercices

Exercice 0

Ecrire un programme qui permet de remplir un tableau (par lecture au clavier), puis d'afficher le contenu du tableau.

Exercice 1

Ecrire un tableau qui calcule la somme des éléments d'un tableau.

Exercice 2

Ecrire une fonction qui détermine la valeur la plus grande d'un tableau. On renverra le maximum puis la position du maximum dans le tableau.

Exercice 3

Ecrire un programme qui décale les éléments d'un tableau vers le bas (le i-ième passe en (i-1)ième, le premier passe en dernier (Nième)).

Exercice 4

Ecrire un programme qui décale les éléments d'un tableau vers le haut (le i-ième passe en (i+1)ième, le dernier (Nième) passe en premier.

Exercice 5

Ecrire un programme qui inverse un tableau : le premier élément est permuté avec le dernier, le deuxième avec l'avant dernier, etc.

Exercice 6

Ecrire un programme qui recherche la première occurrence d'une valeur dans un tableau.

Exercice 7

Ecrire un programme qui recherche la première occurrence d'une valeur dans un tableau trié en utilisant la méthode de recherche dichotomique.

2.9.2 Les tableaux à deux dimensions (matrices)

2.9.3 Exercices

Exercice 1

Ecrire une fonction qui calcule la moyenne des sommes des lignes et la moyenne des sommes des colonnes d'une matrice.

Exercice 2

Ecrire une fonction qui calcule la moyenne par ligne et par colonne dans une matrice.

Exercice 3

Ecrire une fonction qui multiplie deux matrices.

Avec A matrice n lignes, m colonnes. B matrice m lignes, p colonnes. C matrice n lignes, p colonnes.

$C_{i,j} = \text{somme}(k=0, m) A_{i,k} * B_{k,j}$.

Exercice 4

Ecrire une fonction qui trie une matrice par ordre croissant selon une colonne définie.

2.10 Les sous-programmes

2.10.1 Définition :

En programmation, il est fréquent d'avoir à faire exécuter un nombre important de fois les mêmes blocs d'instructions, ou de faire appel régulièrement aux mêmes fonctions de calcul, de dessin, d'accès aux données, ... Il devient donc vite impératif, afin de ne pas avoir à réécrire ces instructions à chaque fois, d'**isoler des blocs de code réutilisables** dans des sous-programmes **dans le but de les rappeler** chaque fois que nécessaire.

Un sous-programme est donc une suite nommée d'instructions, à laquelle il est possible de faire appel, à chaque fois qu'on en a besoin dans le programme, sans devoir la réécrire.

2.10.2 Les fonctions

Activité 1 :

En analysant le code suivant et le résultat affiché sur la console lors de son exécution, déduisez ce que fait le programme et détaillez comment est-ce qu'il le fait.

Programme :

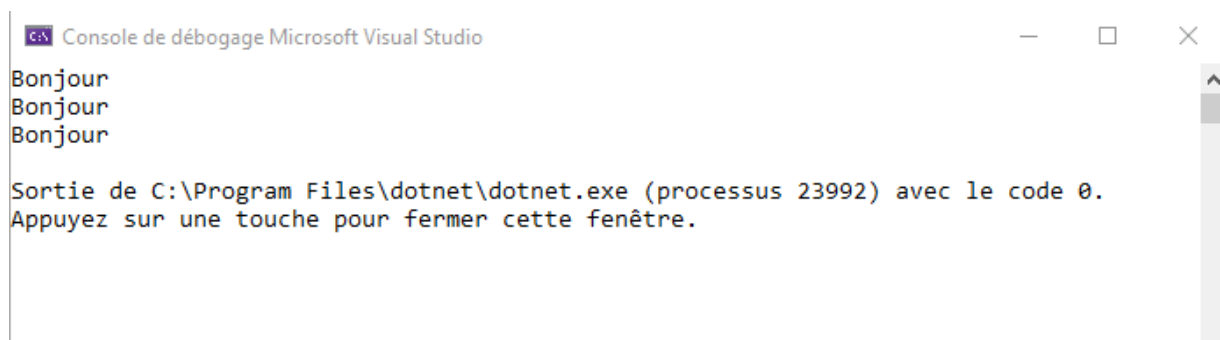
```
class Program
{
    0 références
    static void Main(string[] args)
    {
        afficher();

        afficher();

        afficher();
    }

    3 références
    public static void afficher()
    {
        Console.WriteLine("Bonjour");
    }
}
```

Résultat de l'exécution :



The screenshot shows a console window titled "Console de débogage Microsoft Visual Studio". It displays the output of the program: "Bonjour" printed three times on separate lines. Below the output, a message states: "Sortie de C:\Program Files\dotnet\dotnet.exe (processus 23992) avec le code 0. Appuyez sur une touche pour fermer cette fenêtre."

Solution :

Activité 2 :

En analysant le code suivant et le résultat affiché sur la console lors de son exécution, déduisez ce que fait le programme et détaillez comment est-ce qu'il le fait.

Programme :

```
static void Main(string[] args)
{
    double note11 = 15, note12 = 11, note21 = 10.5 , note22 = 9, moy1, moy2, moyA;

    moy1 = moyenne(note11, note12);
    Console.WriteLine("La moyenne du semestre 1 est : "+moy1);

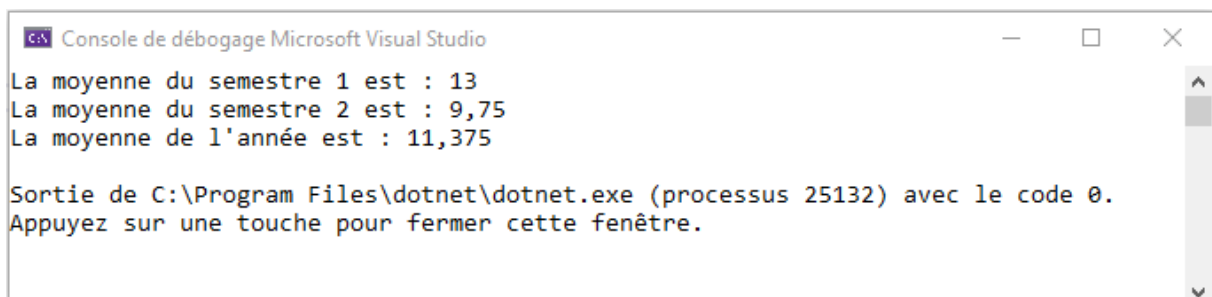
    moy2 = moyenne(note21, note22);
    Console.WriteLine("La moyenne du semestre 2 est : " + moy2);

    moyA = moyenne(moy1, moy2);
    Console.WriteLine("La moyenne de l'année est : " + moyA);
}
```

3 références

```
public static double moyenne(double a, double b)
{
    return (a + b) / 2;
}
```

Résultat de l'exécution :



```
Console de débogage Microsoft Visual Studio
La moyenne du semestre 1 est : 13
La moyenne du semestre 2 est : 9,75
La moyenne de l'année est : 11,375

Sortie de C:\Program Files\dotnet\dotnet.exe (processus 25132) avec le code 0.
Appuyez sur une touche pour fermer cette fenêtre.
```

Solution :

Une fonction est :

Un bloc d'instructions nommé qui peut prendre des

2.10.3 Exercices :

Exercice 1

Le programme suivant permet de calculer le résultat de la division entière de a par b sans utiliser l'opérateur de division « / »

```
class Program
{
    Oréférences
    static void Main(string[] args)
    {
        int a, b, c=0,d;

        Console.WriteLine("Veuillez saisir une valeur entière : ");
        a = int.Parse(Console.ReadLine());
        Console.WriteLine("Veuillez saisir une valeur entière : ");
        b = int.Parse(Console.ReadLine());

        //-----Calcul de la division-----
        d = a;
        while(d>=b)
        {
            d = d - b;
            c++;
        }

        Console.WriteLine(a + "/" + b + "=" + c);
    }
}
```

Modifier le programme pour que le résultat de la division soit calculé et retourné par une fonction qui prend en paramètre deux nombres entiers x et y.

Exercice 2

- a- Ecrire la fonction **addition** qui prend en paramètres deux **nombre réels** (double) **x** et **y** et retourne le résultat de leur addition.
- b- Ecrire la fonction **Soustraction** qui prend en paramètres deux **nombre réels** (double) **x** et **y** et retourne le résultat de leur soustraction.
- c- Ecrire le programme qui demande à l'utilisateur de saisir **deux nombre réels** qui seront enregistrés dans les variables **a** et **b**.
En utilisant les fonctions préparées précédemment, le programme devra afficher les résultats de l'addition puis de la soustraction de **a** et **b**.

Exercice 3

Dans un magasin qui propose un service de photocopie le tarif de la copie dépend du nombre total de copies demandées :

- Pour un nombre de copies compris entre 1 et 9 (inclus), le tarif est de 0.1 €/copie
- Pour un nombre de copies compris entre 10 et 19 (inclus), le tarif est de 0.08 €/copie
- Pour un nombre de copies supérieur à 19, le tarif est de 0.07 €/copie

A - Ecrire la fonction « **tarif** » qui reçoit en paramètre le nombre entier « **copies** », qui correspond au nombre demandé de copies et retourne comme résultat un nombre réel (double) qui correspond au tarif qui sera appliqué pour calculer le prix total.

B - Ecrire la fonction « **total** » qui prend en paramètre le tarif et le nombre de copies et retourne comme résultat le coût total des copies.

C - Ecrire le programme principal qui permet à l'utilisateur de renseigner le nombre de copies qu'il souhaite et en utilisant les fonctions créées précédemment calcule puis affiche le coût total des copies.

Exercice 4

Un nombre **x** est divisible par un nombre **y** si et seulement si le reste de la division de **x** par **y** ($x \% y$) est égal à zéro.

A - Ecrire la fonction « **divisible** » qui prend en paramètre deux entiers **x** et **y**. la fonction retourne **true** si **x** est divisible par **y** et elle retourne **false** dans le cas contraire.

B - Un nombre entier **x** est dit premier si : **x** n'est divisible que par 1 ou par lui-même. Ecrire la fonction « **premier** » qui prend en paramètre un entier **x** et retourne **true** si **x** est premier et **false** le cas contraire.

C – Ecrire le programme principal (contenu de la méthode main) qui permet à l'utilisateur de saisir un nombre entier **a**, puis affiche la liste des diviseurs de **a** et finalement affiche si **a** est un nombre premier ou pas.

2.10.4 Les procédures

2.10.5 Définition :

Une procédure est un bloc d'instructions nommé qui prend en paramètre des données d'entrée ou des références à des variables qui seront modifiées.

A la différence d'une fonction, une procédure ne retourne pas de résultat.

Syntaxe :

La syntaxe d'une procédure est décrite comme suit :

```
public static void <nom de la procédure> (paramètre1, paramètre2, ...,paramètre n)
{
    Instructions
}
```

Contrairement à une fonction où l'on donne le type de retour, lors de la déclaration d'une procédure on utilise le mot clé « **void** » qui veut dire vide en français.

Exemple :

```
public static void bonjour (String nom, String prenom)
{
    Console.WriteLine("Bonjour " + prenom + " " + nom );
}
```

Exercice d'application 1 :

1.a - Ecrivez la procédure **afficherTableau**, qui permet d'afficher toutes les valeurs du tableau **tab**, donné en paramètre d'entrée. Le tableau **tab** contient des nombres réels (double).

1.b - Complétez le programme suivant afin qu'il affiche toutes les valeurs tableau **notes** en utilisant la procédure **afficherTableau** de la question a.

```
static void Main(string[] args)
{
    double[] notes = { 10, 12, 13.5, 15.5, 7.5, 8, 9, 11};
}
```

2.10.6 Passage de paramètres par valeur et passage par référence :

Une variable de type valeur contient ses données directement, par opposition à une variable de type référence, qui contient une référence à ses données.

Passer une variable de type valeur à une méthode par valeur signifie passer une copie de la variable à la méthode. Les modifications du paramètre qui interviennent à l'intérieur de la méthode n'ont aucune incidence sur les données d'origine stockées dans la variable d'argument.

Si vous souhaitez que la méthode appelée change la valeur de l'argument, vous devez la passer par référence, à l'aide du mot clé ref ou out.

Exemple :

```
static void Main(string[] args)
{
    int nombreAdoubler1 = 5;
    int nombreAdoubler2 = 5;
    Console.WriteLine("Avant l'appel de la procédure : ");
    Console.WriteLine("nombreAdoubler1 = " + nombreAdoubler1 + " nombreAdoubler2 = " + nombreAdoubler2);
    Console.WriteLine("\n* * * * *");
    doubler(ref nombreAdoubler1, nombreAdoubler2);
    Console.WriteLine("Après l'exécution de la procédure : ");
    Console.WriteLine("nombreAdoubler1 = " + nombreAdoubler1 + " nombreAdoubler2 = " + nombreAdoubler2);
}

1 référence
public static void doubler(ref int nombreAdoubler1, int nombreAdoubler2)
{
    nombreAdoubler1 = nombreAdoubler1 * 2;
    nombreAdoubler2 = nombreAdoubler2 * 2;
    Console.WriteLine("A l'intérieur de la procédure : ");
    Console.WriteLine("nombreAdoubler1 = " + nombreAdoubler1 + " nombreAdoubler2 = " + nombreAdoubler2);
    Console.WriteLine("\n* * * * *");
}
```

```

Microsoft Visual Studio
Avant l'appel de la procédure :
nombreAdoubler1 = 5 nombreAdoubler2 = 5

* * * * *

A l'intérieur de la procédure :
nombreAdoubler1 = 10 nombreAdoubler2 = 10

* * * * *

Après l'exécution de la procédure :
nombreAdoubler1 = 10 nombreAdoubler2 = 5
```

On remarque dans cet exemple que la valeur de **nombreAdoubler1** a été modifiée après l'appel de la procédure, alors que la valeur de **nombreAdoubler2** est restée la même.

Ceci est dû au fait que dans les paramètres de la méthode, la variable **nombreAdoubler1** est passée par référence alors que la variable **nombreAdoubler2** est passée par valeur.

La méthode a donc modifié la valeur enregistrée dans l'espace mémoire réservé à la variable **nombreAdoubler1** car elle dispose de son adresse mais n'a pas modifié la valeur de **nombreAdoubler2** car elle n'a reçu que la copie de la valeur enregistrée dans la variable.

Exercice d'application 2 :

2a - Ecrivez la procédure **saisirTableau**, qui permet à l'utilisateur de saisir des nombres réels (**double**) et de les enregistrer dans le tableau **notes** de taille 3, donné en paramètre d'entrée.

Attention à ce que la procédure permette de saisir autant de valeurs qu'il y a de cases dans le tableau.

2b - En utilisant les procédures **saisirTableau** et **afficherTableau**, complétez le programme suivant afin que les tableaux, **notes1** et **notes2** soient remplis et ensuite affichés.

```
static void Main(string[] args)
{
    double[] notes1 = new double[10];
    double[] notes2 = new double[10];
}
```

2.10.7 Exercices :

Exercice 1 :

A – Ecrivez la procédure « **moyenne** » qui prend en paramètre la référence de trois tableaux « semestre1 », « semestre2 » et « annee » de type « **double** ». La procédure doit calculer la moyenne des notes des tableaux « semestre1 » et « semestre2 » et l'enregistrer dans le tableau « annee »

B – En faisant **appel** aux procédures des questions **1a** et **2a** du cours, écrivez le programme principal qui permet à l'utilisateur de saisir les notes du 1^{er} et du 2nd semestres et les enregistre respectivement dans les tableaux **semestre1** et **semestre2**.

C – Le programme doit maintenant calculer les moyennes des notes des deux semestres en utilisant la procédure **moyenne** et de les enregistrer dans le tableau **moy**. Il faudra ensuite que le programme affiche les moyennes sur la console.

Exercice 2 :

A – Ecrivez la procédure **tableDeMultiplication** qui prend en paramètre un nombre entier **nombre** dont elle affichera la table de multiplication.

B – Ecrivez le programme principal qui permet d'afficher les tables de multiplication de **1** à **9**, en faisant **appel** à la procédure de la question A.

Exercice 3 :

A - Ecrivez la procédure **ttc**, qui prend en paramètre les deux tableaux **prixHorsTaxes** et **prixTTC** de type double. A partir des prix hors taxes du tableau **prixHorsTaxes**, la procédure doit calculer les prix toute taxes comprises (TTC) et les enregistrer dans le tableau **prixTTC**.

Notes :

- **prixHorsTaxes.Length** permet de connaître la taille du tableau **prixHorsTaxes**.
- **Le $\text{prixTTC} = \text{prixHorsTaxes} * 1.2$**

B – Ecrivez maintenant le programme principal qui permet à l'utilisateur de saisir les prix hors taxes puis en faisant appel à la procédure **ttc** calcul puis affiche les prix TTC.

Exercice 4 :

A - Ecrivez la procédure **maxiMin**, qui prend en paramètre le tableau d'entiers **nombresAcomparer** et les deux variables de type int **max** et **min**. La procédure doit trouver la plus grande et la plus petite valeur, enregistrées dans le tableau **nombresAcomparer** et les enregistrer dans les variable **max** et **min**.