Programmation procédurale
Partie 1 : les éléments syntaxiques de base

Contenu

Pour pouvoir apporter des corrections et des évolutions à des applications existantes, il faut être en mesure de comprendre le code, donc de savoir coder.
Vous allez découvrir les bases de la programmation à travers la présentation des éléments syntaxiques et une mise en pratique systématique.
Vous avez pu constater qu'il y a deux aspects à prendre en compte quand on écrit un programme :

- la logique à suivre (à quel moment il faut afficher, saisir, faire un test, une boucle…) qui est la même quel que soit le langage de programmation utilisé ;
- la syntaxe du langage pour écrire le code correspondant à cette logique : cette fois, la syntaxe dépend du langage.

Le plus important et le plus difficile est de trouver la logique du programme. C'est un peu comme résoudre un casse-tête. Avec le temps et l'expérience, vous aurez des facilités à trouver cette logique. Voilà pourquoi il est important de coder, de beaucoup coder. Mais ce qu'il faut retenir, c'est que la programmation ne s'aborde pas du tout comme un cours classique : le but n'est pas d'apprendre puis d'appliquer ce que vous avez appris. Il y a en réalité peu de choses à apprendre : vous avez à connaître quelques briques (vous avez déjà vu les plus importantes), vous devez maintenant apprendre à les assembler correctement. Et là, ce n'est que votre raisonnement qui peut vous aider. Vous ne devez donc pas "apprendre" mais "comprendre" puis exploiter votre raisonnement.
Voici une présentation détaillée des éléments syntaxiques, avec une mise en pratique systématique.

1. Les variables

Un programme est une succession d'instructions qui nécessitent très souvent la manipulation de variables.

1A. La déclaration des variables

But

Il est parfois nécessaire de mémoriser une information pour pouvoir la réutiliser plus loin dans le programme. Pour cela, il faut au préalable déclarer la variable (la case mémoire) qui va stocker l'information.

Fonctionnement

La déclaration d'une variable se fait en lui donnant un nom et en précisant son type.
Rappelons que le **nom de la variable** est libre, choisi par le programmeur, mais en **respectant certaines règles** : — ne pas commencer par un chiffre ; — ne pas contenir d'espace ;
— contenir uniquement des caractères alphabétiques, éventuellement des chiffres et le caractère "_" ; — éviter les caractères accentués (plusieurs langages n'apprécient pas).
Lors de la déclaration en C#, le type doit précéder la ou les noms des variables.

Les **types simples** sont soient **numériques** (entiers, réels…), soient **caractères** (un seul caractère ou une chaîne), soient **booléens** (ne peut prendre que 2 valeurs : true ou false)

Cas d'erreurs

Lors de la déclaration, une erreur peut survenir dans les cas suivants :

- type oublié ;
- type sans variable ;
- nom de variable incorrect.

Ces erreurs seront repérées et signalées par l'IDE.

Syntaxe C#

Voici les types simples que vous pouvez utiliser en C# (vous retrouvez ces types dans la plupart des langages) :

- `byte` : entier entre 0 et 255
- `short` : entier entre -32768 et 32767
- `int` : entier entre -2147483648 et 2147483647
- `long` : entier entre -9223372036854775808 et 9223372036854775807
- `float` : réel (nombre à virgule) entre -3,402823e38 et 3,402823e38
- `double` : réel (nombre à virgule) entre -1,79769313486232e308 et 1,79769313486232e308
- `decimal` : nombre décimal convenant particulièrement aux calculs financiers (en raison de ses nombres significatifs après la virgule)
- `char` : un seul caractère
- `string` : une chaîne de caractères
- `bool` : une valeur booléenne (true ou false)

Exemples

Voici des exemples de déclarations en C# :

EXEMPLES

```
int valeur, essai, nbre; // variables de type entier char
reponse; // variable de type caractère (un seul caractère)
string message; // variable de type chaîne (0, 1 ou plusieurs
caractères)
bool trouve; // variable de type booléenne (true ou false)
float note; // variable de type réel
```

1B. L'affectation dans une variable

But

Une fois une **variable déclarée**, il est possible de lui **affecter une valeur**. Cette affectation peut se faire au moment de la **déclaration** ou à **tout autre moment dans le programme**. Fonctionnement

La **valeur** ou le contenu d'une variable qui se trouve à **droite** de l'affectation est transféré dans la **variable** qui se trouve à **gauche** de l'affectation.

Si la variable qui reçoit la valeur contenait une autre valeur, celle-ci est écrasée (effacée) par la nouvelle valeur.

Cas d'erreurs

Lors de l'affectation, une erreur peut survenir dans les cas suivants :

- la variable qui se trouve à gauche de l'affectation (donc qui reçoit) n'est pas déclarée ;
- ce qui est à gauche de l'affectation n'est pas une variable ;
- la variable qui se trouve à droite de l'affectation n'est pas déclarée ;

— l'information (variable ou valeur) qui se trouve à droite de l'affectation n'est pas du même type que la variable qui se trouve à gauche (donc qui reçoit) et n'est pas transtypée. Ces erreurs seront repérées et signalées par l'IDE.

Syntaxe C#

Voici la syntaxe de l'affectation en C# :

```
variable = valeur_ou_autre_variable ;
```

Exemples

Voici des exemples de déclarations en C# :

EXEMPLES

```
valeur = 25;
essai = 20;
nbre = 3;
reponse = 'N';
message = "Vous avez trouvé !";
message = "Vous avez trouvé en " + nbre + " fois !";
trouve = true;
trouve = (valeur == essai);
note = 15.5f; // f pour signaler que la valeur 15.5 est de type float
```

Il est aussi possible de déclarer et initialiser une variable en une ligne :

EXEMPLE

```
int nbre = 1;
```

Important :

- **Une variable caractère reçoit une valeur entre apostrophes et non entre guillemets.**
- **Une variable chaîne reçoit une chaîne entre guillemets ou une autre variable chaîne ou une concaténation de chaînes (si dans la concaténation, une variable numérique est utilisée, elle est automatiquement transformée en chaîne).**
- **Une variable booléenne reçoit une valeur booléenne (soit true, soit false sans guillemets, soit le résultat d'un test).**
- **Une variable numérique à virgule peut recevoir une valeur entière ou une valeur à virgule (la virgule étant notée sous la forme d'un point).**

Il existe d'autres types plus complexes que vous aurez l'occasion de découvrir plus tard.

Entraînement 1

Voici plusieurs déclarations :

```
string a, b;
char c;
int d;
byte e;
double f;
bool g;
```

Pour chaque instruction, dites si elle est correcte :

correct

```
a = "bonjour";
b = "bonsoir";
c = "z";
d = 300;
e = 300;
f = 300;
g = "false";
a = b;
b = d;
d = e;
```

2. Les entrées/sorties (en mode console)

Le but d'un programme est d'offrir un service à un utilisateur. Pour cela il doit pouvoir recevoir des **instructions** de l'utilisateur (par exemple à travers une **saisie**) et lui communiquer un résultat (par exemple en l'affichant à l'écran).

Au niveau des entrées/sorties, il y a une importante différence entre les **applications en "mode console"**, donc non graphiques, et les **applications graphiques**. Dans un premier temps, nous travaillons uniquement dans la console, donc pour des applications non graphiques.

2A. L'affichage

But À tout moment il est possible d'afficher une information à l'écran.

Fonctionnement

L'information à afficher peut être :

- une variable ;
- une valeur (une chaîne entre guillemets, une valeur numérique sans guillemets) : on parle alors de "valeur en dur", donc qui s'affichera toujours en l'état à l'écran ;
- un calcul (à mettre entre parenthèses) entre des variables et/ou des valeurs ; — une combinaison des 3 points précédents (en concaténant les informations).

Cas d'erreurs

Lors de l'affichage, une erreur peut survenir dans les cas suivants :

- la ou les variables utilisées ne sont pas déclarées ;

- la ou les variables utilisées ne contiennent pas encore de valeur (pas d'initialisation ou pas d'affectation au préalable) ;
- la valeur en dur de type chaîne n'est pas mise entre guillemets ;
- la concaténation entre valeurs en dur et variables n'est pas correctement gérée (avec le signe + en C# et dans la plupart des langages).

Ces erreurs seront repérées et signalées par l'IDE.

Syntaxe C#

En C#, l'affichage se fait avec la commande `Console.Write` (pour afficher sans faire de retour à la ligne) ou `Console.WriteLine` (pour afficher un message et gérer juste après un retour à la ligne). Enfin, `Console.Clear()` permet d'effacer l'écran. Pour pouvoir utiliser Console, il ne faut pas oublier d'intégrer la bibliothèque System (using System).

Exemples

Voici des exemples d'affichage en C# (en reprenant les déclarations et affectations précédentes) avec, en commentaire, ce que l'on obtient à l'affichage :

EXEMPLES
```
using System; ...
Console.Write("Entrez un essai = "); // Entrez un essai =
Console.WriteLine(message); // Vous avez trouvé en 3 fois !
Console.WriteLine();
Console.WriteLine("valeur = " + valeur); // valeur = 25
Console.Write(30); // 30
Console.Write(valeur); // 25
```

Remarque : si vous ne mettez rien dans les parenthèses de `Console.WriteLine()`, alors un simple retour à la ligne sera affiché.

Entraînement 2

Voici plusieurs déclarations et initialisations :

```
int a = 20, b = 5;
float c = 3.5f;
string d = "merci";
bool e = true;
```

Pour chaque instruction, précisez ce qui va être affiché :

```
Console.WriteLine(a);
Console.WriteLine(a + b);
Console.WriteLine(a + c);
Console.WriteLine(a + " " + b);
Console.WriteLine(a + d);
Console.WriteLine(a + " " + d);
Console.WriteLine(e);
Console.WriteLine(a == b);
Console.WriteLine(d + "à tous");
```

Exercice 1

Créez un nouveau projet sous Visual Studio (il est conseillé, à chaque nouveau projet, de lui donner un nom qui permet de se rappeler de l'exercice correspondant, par exemple ici : Exercice1).
Déclarez 3 variables : nom, prenom et age. Initialisez ces 3 variables avec les valeurs vous concernant.
Affichez ensuite un message qui doit ressembler à ceci (imaginons que vous vous appeliez Alain DUPONT et que vous ayez 19 ans) :
`Bonjour Alain DUPONT, vous avez 19 ans.`
(Bien sûr, le prénom, nom et l'âge ne doivent pas être mis en dur dans la chaîne d'affichage, mais doivent provenir du contenu des variables)

2B. La saisie

But Saisie au clavier d'une valeur et possibilité de mémorisation de cette valeur dans une variable.

Fonctionnement

Arrêt momentané de l'exécution du programme en attendant la saisie.
Une fois la saisie faite et validée (ou non si c'est une saisie d'un caractère sans validation), transfert de l'information saisie dans la variable en mémoire si une affectation de la saisie est faite. Si la variable contenait déjà une information, elle est écrasée par la nouvelle information saisie. Reprise de la suite de l'exécution du programme.

Cas d'erreurs

Lors de la saisie, une erreur peut survenir dans les cas suivants :

- dans le cas d'une saisie avec validation (qui permet normalement de saisir une chaîne), si la variable qui reçoit l'information n'est pas de type chaîne (et que la saisie n'est pas transtypée). Par exemple, si la variable est de type numérique ;
- dans le cas d'une saisie sans validation (qui permet normalement de saisir un seul caractère), si la variable qui reçoit le caractère n'est pas de type caractère ;
- la variable n'est pas déclarée (en revanche elle n'a pas besoin d'être initialisée). Ces erreurs seront repérées et signalées par l'IDE.

Certains langages utilisent des ordres différents pour saisir des types différents, voire s'occupent du transtypage sans avoir besoin de le préciser.

Syntaxe C#

En C#, la saisie se fait avec la commande `Console.ReadLine()` (pour lire une chaîne de caractères se terminant par une validation) ou `Console.ReadKey().KeyChar` (pour lire un seul caractère sans avoir besoin de valider).
Pour **pouvoir utiliser Console**, il ne faut pas oublier **d'intégrer la bibliothèque System (`using System`).**

Exemples

Voici des exemples de saisie en C# (en reprenant les déclarations précédentes) :

EXEMPLES

```
message = Console.ReadLine();
essai = int.Parse(Console.ReadLine());
note = float.Parse(Console.ReadLine());
reponse = Console.ReadKey().KeyChar;
```

Entraînement 3

Voici plusieurs déclarations :

```
string a, b;
char c;
int d;
byte e;
double f;
```

Pour chaque instruction, donnez la bonne syntaxe, dans le cas où celle proposée n'est pas correcte :

```
a = Console.ReadLine();
b = int.Parse(Console.ReadLine());
c = char.Parse(Console.ReadLine());
c = Console.ReadKey().KeyChar;
d = Console.ReadLine();
e = int.Parse(Console.ReadLine());
f = double.Parse(Console.ReadLine());
```

Exercice 2

Créez un nouveau projet Exercice2.
Même principe que l'exercice 1, excepté qu'il faut remplacer les initialisations par des saisies (avant chaque saisie, pensez à préciser ce qu'il faut saisir).

3. Les opérations

Un programme doit pouvoir réaliser différentes opérations arithmétiques et logiques.

3A. Le calcul arithmétique

But

Possibilité de réaliser des calculs arithmétiques entre des variables numériques et/ou des valeurs numériques, et affectation du résultat dans une variable (ou affichage direct du résultat).

Fonctionnement

Dans le cas d'une affectation, le calcul qui se trouve à droite de l'affectation est évalué. Son résultat est transféré dans la variable qui se trouve à gauche de l'affectation.
Si la variable qui reçoit le résultat du calcul contenait une autre valeur, celle-ci est écrasée pour laisser la place à la nouvelle valeur.
Dans le cas d'un affichage direct, le calcul, mis entre parenthèses, est évalué avant l'affichage. Il peut être concaténé à une chaîne.

Cas d'erreurs

Lors du calcul, une erreur peut survenir dans les cas suivants :

— la ou les variables utilisées dans le calcul ne sont pas déclarées et/ou ne contiennent pas de valeur ;
— la ou les variables utilisées dans le calcul ne sont pas numériques ;
— le calcul demandé n'est mathématiquement pas possible (division par 0...).

Dans le cas où le résultat du calcul est affecté dans une variable, il est possible de rencontrer aussi les erreurs citées dans la partie affectation.
Ces erreurs seront repérées et signalées par l'IDE (excepté la division par 0 qui provoquera une erreur lors de l'exécution.

Opérations arithmétiques possibles

Voici les opérations arithmétiques possibles sur des variables et/ou valeurs numériques :

+	Addition
-	Soustraction
*	Multiplication
/	Division
/	Division entière lorsque le résultat est transféré dans une variable type entier Exemple : la division entière de 14 par 3 donne 4
%	Reste de la division (modulo) Exemple : le reste de la division de 14 par 3 donne 2 car 4*3=12 et il manque 2 pour arriver à 14. (14/3 = 4.66666… on ne garde que la partie entière qui est bien 4).
( )	Parenthèses (permet de hiérarchiser les calculs, comme en mathématique).

Raccourcis d'écritures possibles en C# :

a = a + 1	a++
a = a – 1	a--
a = a + b	a += b
a = a – b	a -= b
a = a * b	a *= b
a = a / b	a /= b

Exemples en C#

Voici des exemples de calculs arithmétiques en C# :

EXEMPLES

```
float note1, note2, moyenne;
int total, valeur, diviseur, result, reste;
note1 = 15.5f; /* f pour signaler que la valeur en dur est
float, d pour double, m pour decimal (rien pour les entiers) */
note2 = 8.5f;
moyenne = (note1 + note2) / 2; // moyenne contiendra 12
total = 25 * 2; // total contiendra 50
total = total + 2; // total contiendra alors 52
total += 2; // instruction similaire à la précédente (total vaut
54)
valeur = 14; diviseur = 3;
result = valeur / diviseur; /* result contiendra 4 : la division étant
transférée dans un entier, seule la partie entière est récupérée */
reste = valeur % diviseur; // reste contiendra 2
```

Entraînement 4

Voici plusieurs déclarations et initialisations :

```
int a = 34, b = 5, c = 4, d, e, f;
float g;
```

Voici une succession de traitements. A chaque ligne, précisez la valeur de la variable à gauche de l'affectation :

```
d = a / c;              d =
e = a % c;              e =
b *= c;                 b =
b++;                    b =
f = d * c + e;          f =
g = a / c;              g =
```

Exercice 3

Créez un nouveau projet Exercice3.

Le programme doit permettre la saisie d'une somme de notes (pour le moment on saisit directement la somme, par exemple 125,5), ainsi que le nombre de notes (par exemple 8) et doit afficher la moyenne (si les valeurs précédentes sont saisies, on devrait obtenir 15.6875).

Attention, la saisie d'un nombre à virgule se fait avec le séparateur configuré par votre système d'exploitation (s'il est configuré pour les normes françaises, il ne faut pas utiliser le point mais la virgule, et il est inutile d'ajouter la lettre f lors de la saisie).

Exercice 4

Créez un nouveau projet Exercice4.

Le programme doit permettre la saisie d'un prix HT, d'un taux de TVA et d'afficher le prix TTC équivalent.

3B. Les expressions logiques

But

Possibilité de réaliser des expressions logiques et utilisation du résultat (true ou false) dans un test, ou affectation du résultat dans une variable booléenne (ou affichage direct du résultat, nettement plus rare dans ce cas vu que le résultat sera true ou false).

Fonctionnement

Même principe que pour le calcul.

Cas d'erreurs

Lors de l'expression logique, une erreur peut survenir dans les cas suivants :

- la ou les variables utilisées dans l'expression logique ne sont pas déclarées et/ou ne contiennent pas de valeur ;
- l'expression logique demandée n'est techniquement pas réalisable (incompatibilité de types, opérations impossibles…).

Dans le cas où le résultat de l'expression logique est affecté dans une variable, la variable doit être de type booléen.

Ces erreurs seront repérées et signalées par l'IDE.

Comparateurs mathématiques possibles et syntaxe en C#

Dans une expression logique, voici les comparateurs mathématiques possibles sur des variables et/ou valeurs numériques, chaînes ou booléennes :

Opérateur	C#	Syntaxe	Fonctionnement
=	==	a == b	Vrai si les 2 sont égaux
≠	!=	a != b	Vrai si les 2 sont différents
>	>	a > b	Vrai si a est strictement supérieur à b

<	<	a < b	Vrai si a est strictement inférieur à b
≥	>=	a >= b	Vrai si a est supérieur ou égal à b
≤	<=	a <= b	Vrai si a est inférieur ou égal à b

Opérations logiques possibles et syntaxe en C#

Dans une expression logique, voici les opérations logiques possibles sur des variables et/ou valeurs booléennes (a et b sont booléens ou contiennent une expression logique résultat d'une comparaison mathématique) :

Opérateur	C#	Syntaxe	Fonctionnement
ET	&&	a && b	Vrai si a et b sont tous les 2 vrais Faux dans les autres cas
OU	||	a || b	Vrai si a est vrai ou b est vrai Faux si les 2 sont faux
NON	!	!a	Vrai si a est faux Faux si a est vrai

Exemples en C#

Voici les expressions logiques possibles sur des variables et/ou valeurs numériques ou booléennes (les chaînes peuvent aussi être comparées par rapport à l'ordre alphabétique, mais pas directement avec des signes de comparaisons : on verra plus loin comment faire) :

EXEMPLES

```
bool trouve = (valeur==essai); /* les 2 variables sont comparées, si elles sont égales, true est transféré dans la variable trouve, false dans le cas contraire */
int A = 3, B = 2, C = 5;
bool compare = (A > B && (A + B == C)); /* le test est vrai puisque A qui vaut 3 est supérieur à B qui vaut 2, et A+B est bien égal à C, donc la variable compare va recevoir true */
```

Entraînement 5

Voici plusieurs déclarations :

```
bool a;
int b = 3, c = 2, d = 1;
```

À chaque affectation dans la variable 'a', précisez sa valeur :

```
a = (b - c == d);                a =
a = (b <= c) || (c >= d);        a =
a = !((b <= c) || (c >= d));     a =
a = !(b <= c) || (c >= d);       a =
a = !((b == c) && (b > d));      a =
a = b >= (c + d);                a =
```

4. Les instructions conditionnelles

Un programme doit pouvoir exécuter des instructions sous condition (une ou plusieurs fois). Cela est possible à travers les traitements alternatifs ou itératifs.

4A. L'alternative

But Exécution de traitements sous condition.

Fonctionnement

Une expression logique est évaluée (c'est la condition dans le "si" ou "if").
Si elle est vraie, un lot de traitements est exécuté (on parle de la partie "alors" ou "then" de l'alternative). Si elle est fausse, un autre lot de traitements peut être exécuté (cette seconde partie est optionnelle : on parle de la partie "sinon" ou "else" de l'alternative). **Cas d'erreurs**

Lors d'une alternative, une erreur peut survenir dans les cas suivants :

- la condition n'est pas une expression logique valide (ne retourne pas un booléen) ;
- la partie "then" ne contient pas de traitements ;
- il y a plusieurs parties "then" ou plusieurs parties "else" pour une seule condition ; — Il y a une partie "else" sans partie "then".

Ces erreurs seront repérées et signalées par l'IDE. **Syntaxe**

C#

Voici les 2 syntaxes possibles en C# pour l'alternative simple.

```
// Syntaxe ne contenant que le bloc 'then'
if (condition) {
   // instructions exécutées si la condition est vraie
}
// Syntaxe contenant le bloc 'then' et le bloc 'else'
if (condition)
{
   // instructions exécutées si la condition est vraie
} else
{
   // instructions exécutées si la condition est fausse
}
```

Exemples

Voici des exemples d'alternatives :

EXEMPLES

```
float moyenne = 8.5f;
/* Dans l'exemple ci-dessous, la condition est fausse ('moyenne' n'est pas
supérieure ou égale à 10) donc seul le message "recalé" sera affiché */
 if (moyenne >= 10
{
    Console.WriteLine("accepté");
}
else
{
    Console.WriteLine("recalé");
}
/* Dans l'exemple ci-dessous, la 1ère condition est encore fausse donc le bloc
'else' est exécuté. Ce bloc contient une nouvelle condition qui cette fois est
vraie ('moyenne' est bien supérieure ou égale à 8), donc le bloc 'then' de cette
nouvelle condition est exécuté, ce qui donne le message "rattrapage". Observez
les indentations (décalages du code à l'intérieur des blocs).
Remarquez aussi qu'il est possible d'imbriquer des alternatives. En fait, tout
peut être imbriqué en programmation.
Observez aussi le fait que quand on est dans le premier 'else', c'est que la
première condition est fausse ('moyenne' est inférieure à 10) donc il est
évidemment inutile de retester moyenne avec 10 : il suffit de le tester avec 8.
*/
if (moyenne >= 10)
{
    Console.WriteLine("accepté");
}
else
{
    if (moyenne >= 8)
    {
       Console.WriteLine("rattrapage");
    }
else
    {
        Console.WriteLine("recalé");
    }
}
```

Entraînement 6

Sans tester sur ordinateur, donnez les valeurs de a, b et c à la fin de l'exécution de cette séquence :

```
int a = 3, b = 2, c = 1;
if(a > b || b < c)
{
    b++;
    if(!(a != b))
    {
        c = c * 3;
    }
    else
    {
        a -= 2;
        b = b - 1;
    }
}
else
{
    a += 3;
    b = c + 1;
    c -= 1;
}

a =
b =
c =
```

Exercice 5

Créez un nouveau projet Exercice5.
Le programme doit permettre de saisir un âge et d'afficher "majeur" si la personne a au moins 18 ans, "mineur" si elle n'a pas encore atteint cet âge.
Dans le cas où la personne est mineure, il faudra aussi afficher un message qui précise dans combien d'années la personne sera majeure.

Exercice 6

Créez un nouveau projet Exercice6.

Le programme doit permettre de saisir une moyenne et d'afficher la mention correspondante ("très bien" à partir de 16, "bien" à partir de 14, "assez bien" à partir de 12, "passable" à partir de 10, "recalé" en dessous de 10).

Cas particulier de l'alternative multiple

Il existe une variante de l'alternative qui permet à partir d'une variable, de gérer plusieurs cas différents. Voici la syntaxe :

EXEMPLE

```
switch (variable)
{
case valeur1:
     // traitements exécutés si variable == valeur1
    break;
case valeur2:
    // traitements exécutés si variable == valeur2
    break;
    ...
case valeurN:
    // traitements exécutés si variable == valeurN
    break;
default :
    /* bloc 'else' optionnel : traitements exécutés si 'variable' différent de
toutes les valeurs précédentes */
    break;
}
```

Cette forme d'alternative est intéressante pour tester une variable avec plusieurs valeurs distinctes et non des plages de valeurs (par exemple, pour tester les différents choix d'un menu).

4B. L'itération

But Possibilité de répéter plusieurs fois une séquence de traitements.

Fonctionnement

Une séquence de traitements est placée dans un bloc qui commence ou se termine par un test : suivant le résultat du test, la séquence est répétée. **Cas d'erreurs**

Lors d'une itération, une erreur peut survenir dans les cas suivants :

- la syntaxe de l'itération n'est pas correcte ;
- le test n'est syntaxiquement pas correct (voir les erreurs possibles sur les expressions logiques et mathématiques) ;

Ces erreurs seront repérées et signalées par l'IDE.

Il existe trois itérations possibles : le choix doit se faire suivant les besoins.

Itération indéterministe avec test en début de boucle

Syntaxe en C#

```
while (condition_pour_continuer_à_boucler)
{
    // traitements qui seront répétés tant que la condition est vraie
}
```

Fonctionnement

La condition est évaluée une première fois. Si elle est vraie, les traitements sont exécutés et le programme retourne à la condition. À chaque tour de boucle, la condition est à nouveau évaluée et les traitements sont exécutés tant que la condition est vraie. Lorsque la condition devient fausse, l'exécution du programme reprend après la boucle.
La condition étant testée dès le début de la boucle, le nombre d'itérations peut aller de 0 à n fois : 0 fois si la condition est fausse dès la première fois, n fois sans savoir à l'avance la valeur de n car cela dépend de l'évolution des valeurs contenues dans la condition (d'où le fait que c'est une boucle indéterministe : on ne sait pas à l'avance le nombre d'itérations).
Cette boucle est appelée la boucle "universelle" car toute itération peut être représentée par cette boucle 'while'.
Vous verrez plus loin deux autres types de boucles qui sont plus spécifiques. **Exemple**
Voici un exemple :

EXEMPLE

```
int somme = 0, k = 1;
while (k <= 5)
{
    somme = somme + k;
    k++;
}
// en sortie de boucle, somme contient 1+2+3+4+5 = 15
```

Faisons la "trace" du programme (exécution pas-à-pas des instructions et évolution du contenu des variables) comme si nous nous mettions à la place de l'ordinateur :

Ligne de commande		k	somme
`int somme = 0, k = 1;`		1	0
`while (k <= 5)`	vrai		
`somme = somme + k;`			0+1 = 1
`k++;`		1+1 = 2	
`while (k <= 5)`	vrai		
`somme = somme + k;`			1+2 = 3
`k++;`		2+1 = 3	
`while (k <= 5)`	vrai		
`somme = somme + k;`			3+3 = 6

k++;		3+1 = 4	
while (k <= 5)	vrai		
somme = somme + k;			6+4 = 10
k++;		4+1 = 5	
while (k <= 5)	vrai		
somme = somme + k;			10+5 = 15
k++;		5+1 = 6	
while (k <= 5)	faux		
// sortie de boucle…			

Vous remarquez qu'à chaque fois le test de la boucle est évalué et, tant qu'il est vrai, le contenu de la boucle est exécuté. Lorsque 'k' atteint la valeur 6, le test n'est plus correct et l'exécution du programme reprend après la boucle.

Entraînement 7

Voici un extrait de code. Sans tester sur ordinateur, donnez les valeurs de a, b et c en fin d'exécution.

```
int a = 3, b = 2, c = 1;
while (a > b && b >= 0)
{
    if(a == c)
    {
        c++;
        b++;
    }
    else
    {
        a--;
        b--;
    }
}

a =
b =
c =
```

Exercice 7

Créez un nouveau projet Exercice7.
Le programme doit permettre de saisir plusieurs prix en arrêtant la saisie avec 0 (cette information sera d'ailleurs précisée dans le message qui s'affiche à l'écran et qui précède la saisie). Au final, il faudra afficher le total. Attention, on peut très bien saisir 0 dès le départ.

Exercice 8

Créez un nouveau projet Exercice8.
Le programme doit faire la même chose que le précédent (saisie de prix et affichage du total) mais cette fois, avant la saisie d'un prix, une question est posée à l'utilisateur : "Avez-vous un prix à saisir ? (O/N) ". La réponse doit se faire en permettant à l'utilisateur de taper soit O soit N sans valider. S'il répond O, alors on lui demande de saisir un prix. S'il répond N, alors le total des prix précédemment saisis s'affiche.

Itération indéterministe avec test en fin de boucle

Syntaxe en C#

EXEMPLE

```
do
{
    // traitements qui seront répétés tant que la condition est vraie
} while (condition_pour_continuer_à_boucler);
```

Fonctionnement

Cette fois, les traitements sont exécutés une première fois avant que la condition soit évaluée. Si elle est vraie, les traitements sont à nouveau exécutés. Le fonctionnement est donc similaire à l'itération précédente excepté que le test est en fin de boucle. Du coup le nombre d'itérations va de 1 à n fois (les traitements sont au moins exécutés une fois).
Ce type de boucle est idéal quand les traitements doivent être faits au moins une fois. Elle peut bien sûr aussi être transformée en une boucle 'while' universelle.
Remarque : Dans certains langages, et aussi en pseudo-langage (en algorithmique), la condition est inversée et on boucle "jusqu'à" ce que la condition soit vraie (une sorte de "do…until"). Ceci est dit à titre purement informatif, juste pour que vous ayez conscience qu'il peut exister des variantes. **Exemple**
Voici un exemple :

EXEMPLE

```
int somme = 0, k = 1; do
{
  somme = somme +  k;
 k++;
} while (k<=5)
// en sortie de boucle, somme contient 1+2+3+4+5 = 15
```

Remarquez que l'exemple est le même que celui utilisé précédemment pour illustrer la boucle 'while'. C'est aussi pour vous montrer qu'il existe souvent plusieurs écritures différentes, qui fonctionnent.

Entraînement 8

Voici un extrait de code. Sans tester sur ordinateur, donnez les valeurs de a, b et c en fin d'exécution.

```
int a = 3, b = 2, c = 1;
do
{
    if (a > c)
    {
        b--;
        c++;
    }
    else
    {
        b += 3;
    }
} while (a != b || a != c);

a =
b =
c =
```

Exercice 9

Créez un nouveau projet Exercice9.
Le programme doit permettre de saisir une note en contrôlant que la valeur saisie est bien entre 0 et 20. La saisie est redemandée jusqu'à ce qu'une note correcte soit saisie. Au final, il faudra afficher la note correcte saisie.

Exercice 10

Créez un nouveau projet Exercice10.
Le programme doit permettre de saisir le sexe (sous forme H ou F) puis d'afficher "Bonjour madame" ou "Bonjour monsieur". La saisie de H ou F doit être contrôlée (il faut redemander la saisie tant que l'utilisateur n'a pas tapé H ou F).

Itération déterministe Syntaxe en C# :

EXEMPLE

```
for (initialisation ; condition_pour_boucler ; opération_répétée) {
    // traitements qui seront répétés tant que la condition est vraie
}
```

Fonctionnement

La première ligne est composée de 3 parties :

- initialisation : cette partie n'est exécutée qu'une seule fois (on l'utilise généralement pour initialiser une variable qui sert de compteur de boucle ;
- condition_pour_boucler : comme dans les autres itérations, cette condition est évaluée à chaque tour de boucle (la sortie de la boucle se fait quand la condition est fausse) ;
- opération_répétée : à chaque tour de boucle, si la condition est vraie, cette opération est faite après que les traitements dans la boucle soient exécutés.

Compte tenu de la structure particulière de cette itération, elle pourrait très bien être indéterministe. Cependant, elle est classiquement utilisée comme boucle déterministe de la façon suivante :

- initialisation : une variable (compteur) est initialisée (généralement à une valeur de départ type 0 ou 1, ou une valeur d'arrivée) ;
- condition_pour_boucler : le compteur est testé (le but étant de boucler tant que le compteur ne dépasse pas une certaine valeur, ou au contraire ne passe pas en dessous d'une certaine valeur) ; — opération_répétée : le compteur est incrémenté (ou décrémenté).

Ce type de boucle est idéal quand on sait combien de fois il faut boucler. Elle peut bien sûr aussi être transformée en une boucle 'while' universelle.

Exemple

Voici un exemple :

EXEMPLE

```
int k,
somme=0;
for (k = 1; k <= 5; k++)
{
    somme = somme + k;
}
// en sortie de boucle, somme contient 1+2+3+4+5 = 15
```

Remarquez que l'exemple est encore une fois le même. Effectivement, l'exemple précédent se prête parfaitement à ce type de boucle 'for' car on sait combien de fois il faut boucler. Observez aussi cette variante :

EXEMPLE

```
int somme=0;
```

```
for (int k = 1; k <= 5; k++)
{
    somme = somme + k;
}
// en sortie de boucle, somme contient 1+2+3+4+5 = 15
```

Il est tout à fait possible de déclarer une variable au niveau d'un bloc, et non pour tout le programme. Cette fois, 'k' n'est visible qu'au niveau de la boucle, ce qui nous convient parfaitement car k n'est utile que dans la boucle. Cette notation est classiquement utilisée pour les compteurs de boucle 'for' (comme c'est le cas ici pour 'k') mais pas seulement, l'idée étant de toujours déclarer une variable au niveau le plus bas (dans le bloc où elle est utilisée).
On utilise généralement juste une lettre pour nommer un compteur (i, j, k...) mais ce n'est absolument pas une obligation. La variable 'somme', en revanche, n'aurait pas pu être déclarée dans la boucle car elle doit être initialisée à 0 avant la boucle. La variable 'k' n'a pas besoin d'être initialisée avant la boucle car elle est initialisée dès le début de boucle (au niveau du 'for').

Entraînement 9

d'exécution.

```
int a = 3, b = 2, c = 1;
for (int j=0 ; j < 5 ; j++)
{
    if (a > j)
    {
        b++;
    }
    else
    {
        c++;
        b--;
    }
}

a =
b =
c =
```

Voici un extrait de code. Sans tester sur ordinateur, donnez les valeurs de a, b et c en fin

Exercice 11

Créez un nouveau projet Exercice11.

Le programme doit permettre de saisir 5 notes et au final d'afficher la moyenne de ces notes.

solution

Exercice 12

Créez un nouveau projet Exercice12.

Le programme doit permettre d'afficher la table de multiplication de 3, sous la forme :

```
3 x 0 = 0
3 x 1 = 3
3 x 2 = 6
…
3 x 10 = 30
```

Voyons si vous arrivez à gérer plusieurs boucles en même temps avec les exercices suivants :

Exercice 13

Créez un nouveau projet Exercice13.

Le programme doit permettre de saisir un entier entre 1 et 9 (en contrôlant la saisie) et d'afficher la table de multiplication de cet entier.

Exercice 14

Créez un nouveau projet Exercice14.

Améliorer le programme précédent en faisant en sorte que l'on puisse demander d'afficher plusieurs tables de multiplications. Pour cela, une fois la 1ère table affichée, il faut demander "Voulez-vous afficher une nouvelle table de multiplication ? (O/N) ". Cette question doit être reposée après l'affichage de chaque table. Vous pouvez aussi contrôler la réponse (pour qu'elle soit bien 'O' ou 'N'). Pensez à effacer l'écran entre 2 tables de multiplications.

5. Les opérations spécifiques

Certains types permettent des opérations spécifiques. Voyons les plus classiques.

5A. La manipulation des chaînes

Tous les langages offrent différentes possibilités de manipulations des chaînes. **But**

Pouvoir réaliser différentes opérations sur les chaînes. Vous avez déjà vu la concaténation mais il y a beaucoup d'autres possibilités. **Syntaxe C#**

Voici les opérations les plus classiques sur les chaînes, et leur syntaxe en C# (chaine1, chaine2, oldchaine, newchaine sont de type string, debut, longueur sont de type int) :

`chaine1.Length`	Donne la longueur de la chaîne.
`chaine1.Contains(chaine2)`	Vrai si chaine2 est contenu dans chaine1.
`chaine1.StartsWith(chaine2)`	Vrai si chaine1 commence par chaine2.
`chaine1.EndsWith(chaine2)`	Vrai si chaine1 se termine par chaine2.
`chaine1.Equals(chaine2)`	Vrai si les 2 chaînes sont égales (privilégiez Equals pour comparer les chaînes plutôt que ==).
`chaine1.CompareTo(chaine2)`	Comparaison par rapport à l'ordre alphabétique : -1 si chaine1 avant chaine2 1 si chaine1 après chaine2 0 si chaine1 identique à chaine2
`chaine1.IndexOf(chaine2)`	Donne la position de chaine2 dans chaine1 (sachant que le 1er caractère est à la position 0). -1 si chaine2 n'est pas trouvé.
`chaine1.IndexOf(chaine2, debut)`	Donne la position de chaine2 dans chaine1, en cherchant à partir de la position debut (sachant que le 1er caractère est à la position 0).
`chaine1.Insert(debut, chaine2)`	Insère chaine2 dans chaine1 à partir de la position debut.
`chaine1.Replace(oldchaine,newchaine)`	Donne une chaîne identique à chaine1 mais avec toutes les occurrences oldchaine remplacées par newchaine (oldchaine et newchaine sont de type string).
`chaine1.Replace(oldcarac,newcarac)`	Donne une chaîne identique à chaine1 mais avec toutes les occurrences oldcarac remplacées par newcarac (oldcarac et newcarac sont de type char).
`chaine1.Substring(debut)`	Permet de récupérer une sous-chaîne issue de l'extraction dans chaine1 à partir de la position début jusqu'à la fin.
`chaine1.Substring(debut, longueur)`	Permet de récupérer une sous-chaîne issue de l'extraction dans chaine1 à partir de la position début et avec longueur comme nombre de caractères.
`chaine1.ToLower()`	Permet d'obtenir la même chaîne tout en minuscule.
`chaine1.ToUpper()`	Permet d'obtenir la même chaîne tout en majuscule.

Exemples

Voici quelques exemples :

EXEMPLES

```
string phrase = "Bonjour à tous.";
int longueur = phrase.Length; // longueur contient 15
if (phrase.Contains("ou")) // vrai
{
    Console.WriteLine("la chaine contient ou");
}
if (phrase.StartsWith("Bon")) // vrai
{
    Console.WriteLine("la chaine commence par Bon");
}
string newphrase = phrase.Insert(10, "vous "); // newphrase contient "Bonjour à
vous tous"
newphrase = phrase.Replace("Bonjour", "Bonsoir");
// newphrase contient "Bonsoir à tous"
string extrait = phrase.Substring(6, 3);
// extrait contient "r à"
```

Entraînement 10

Voici un extrait de code. Sans tester sur ordinateur, donnez les valeurs de chaque variable.

```
int a, b;
string w, x, y, z = "programmer, c'est facile";
bool e, f;
a = z.Length;                                    a =
b = z.IndexOf(",");                              b =
w = z.Insert(17, " très");                       w =
x = z.Replace('p', 'P');                         x =
y = z.Substring(12, 5);                          y =
e = z.Contains(" ,");                            e =
f = z.EndsWith(z.Substring(z.IndexOf("f"),6));   f =
```

Exercice 15

Créez un nouveau projet Exercice15.

Le programme doit permettre de saisir une phrase puis d'afficher la même phrase mais avec des "_" à la place des espaces.

Exercice 16

Créez un nouveau projet Exercice16.
Le programme doit permettre de saisir une phrase puis un mot. Si ce mot est présent dans la phrase, il faut afficher la phrase mais uniquement à partir de ce mot. Si le mot n'est pas présent, il faut le signaler par un message.
Exemple :
Si la phrase saisie est "N'essaie pas, fais-le, ou ne le fais pas"
Et que le mot saisi est "ne"
Le résultat doit être "ne le fais pas"

5B. Les fonctions mathématiques

Tous les langages offrent des outils mathématiques puissants.

But Pouvoir réaliser des opérations mathématiques évoluées.

Syntaxe C#

En utilisant la classe Math (j'utilise le terme "classe" dès maintenant, mais vous comprendrez vraiment son sens plus tard), vous avez accès à de nombreuses possibilités. Vous pouvez facilement le voir en tapant Math suivi d'un point. Voici quelques exemples :

`Math.PI`	Donne la valeur du nombre PI (type double).
`Math.Pow(nombre, puissance)`	Donne le nombre élevé à la puissance.
`Math.Abs(nombre)`	Donne la valeur absolue d'un nombre (le nombre sans son signe).
`Math.Max(val1, val2)`	Donne la plus grande des 2 valeurs.
`Math.Min(val1, val2)`	Donne la plus petite des 2 valeurs.
`Math.Round(nombre)`	Donne l'entier le plus proche.
`Math.Sqrt(nombre)`	Donne la racine carrée d'un nombre.
`...`	

Faites attention aux types retournés par les différentes fonctions. Les variables qui doivent recevoir le résultat doivent être du bon type. À chaque fois que vous utilisez une fonction, l'IDE précise le type concerné. Certains types sont cependant compatibles. **Exemples**

Voici quelques exemples :

EXEMPLES

```
int val = Math.Min(2, 5); // val contient 2
```

```
double cube = Math.Pow(val, 3); // cube contient 8 (2 puissance 3)
int valabs = Math.Abs(-12); // valabs contient 12
double nombre = Math.Sqrt(valabs); // nombre contient 3,464101615...
double arrondi = Math.Round(nombre); // arrondi contient 3
double arrondi2 = Math.Round(Math.Sqrt(valabs)); // arrondi2 contient 3
```

Remarquez le dernier exemple : il est possible de mettre dans les parenthèses l'appel d'une autre fonction, à condition qu'elle donne une valeur du type attendu dans les parenthèses.

Entraînement 11

Voici un extrait de code. Sans tester sur ordinateur, donnez les valeurs de chaque variable.

```
int a, b, c = 4;
double d, e;
a = Math.Max(3, c);                          a =
b = Math.Abs(c - 10);                        b =
d = Math.Round((double)(c * a) / 2);         d =
e = Math.Pow((double)c - 2, Math.Min(c, 2)); e =
```

Exercice 17

Créez un nouveau projet Exercice17.
Le programme doit permettre de saisir un nombre puis de demander de saisir la racine carrée de ce nombre. Si l'utilisateur a saisi la bonne valeur, "bravo" sera affiché, sinon il faudra lui dire qu'il s'est trompé et afficher la bonne réponse sous la forme : "Erreur, la racine carrée de 9 est 3" (dans le cas où le nombre de départ est 9).

Exercice 18

Créez un nouveau projet Exercice18.
Le programme doit permettre de saisir 3 nombres (type double) et d'afficher le plus petit des trois. Le but ici est de ne pas utiliser de "if".

5C. Les autres fonctions

Sans détailler plus, sachez qu'il existe de nombreux outils mis à votre disposition par les langages pour manipuler différentes notions. Par exemple, il existe des fonctions pour gérer les dates (nombre de jours entre 2 dates, jour de la semaine d'une date...).
Vous aurez l'occasion de découvrir ces possibilités au cours de votre apprentissage et au fur et à mesure des besoins. Pour le moment, l'important est juste de savoir que cela existe.

6. Optimisation

Un programme ne doit pas se contenter de "marcher". Il doit être optimisé pour être le plus performant possible. L'optimisation intervient à plusieurs niveaux.

6A. Optimisation en lisibilité : règles de codage

On parle aussi de charte de code. L'idée est de respecter un certain nombre de règles pour que le code soit le plus propre possible mais aussi et surtout soit facile à lire et relire par l'auteur ou par tout autre développeur connaissant la charte. Voici les règles principales.

Indentations

Les indentations représentent les décalages faits dans le code pour améliorer sa lisibilité en permettant de repérer plus facilement les différents blocs et leurs imbrications. Les décalages utilisés doivent toujours être de même taille. Tous les IDE évolués proposent les indentations et les insèrent automatiquement. Cependant, suivant les modifications que vous allez apporter, vous pouvez parfois "casser" ces indentations. Pensez donc à les contrôler. Sous Visual Studio, si vous sélectionnez un ensemble de lignes et que vous faites Ctrl+K suivi de Ctrl+F, les indentations se refont automatiquement dans la partie sélectionnée. Autre astuce : il suffit de supprimer la dernière accolade fermante et de la réinsérer : toutes les indentations se refont. Ce qu'il faut retenir c'est que chaque contenu de bloc doit être décalé à droite.

EXEMPLE

```
int a = 3, b = 2, c = 3;
while (a > b)
{
  if (a == c)
  {
     a-;
     if(a > 1)
     {
       c++;
       b = Math.Min(a, c);
     }
   } else
   {
     b++;
    }
}
```

Remarquez bien que chaque bloc est entouré d'accolades et le contenu est décalé vers la droite.

Nom des variables et des modules

Les noms des variables et des modules (et, nous verrons plus tard, d'autres choses comme les classes), doivent être choisis judicieusement.

Voici les règles classiquement admises :

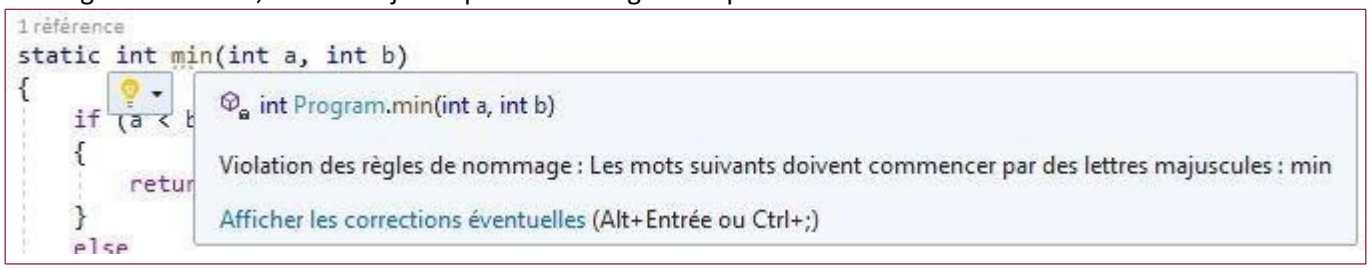
- Le nom doit représenter clairement le contenu.
- Les caractères utilisés dans le nom sont limités : lettres non accentuées, chiffres, tiret bas. Le tiret bas n'est préconisé qu'en premier caractère, dans certaines circonstances particulières.

— Le premier caractère du nom ne peut être qu'une lettre ou, uniquement dans certains cas précis, un tiret bas.
— L'utilisation des majuscules respecte la notation camelCase pour les noms de variables (chaque mot constituant le nom commence par une majuscule, sauf le premier mot) et PascalCase (aussi appelée UpperCamelCase) pour les noms de modules (idem camelCase mais même le premier mot commence par une majuscule).

EXEMPLE

```
float note, moyenneNotes;
int nbNotes, coef;
string matiere, nomEtudiant,
prenomEtudiant;
static int Min(int a, int b){ ...}
```

Dans cet exemple, on a respecté les règles suivantes : nommer de façon parlante, éviter les caractères accentués, utiliser la notation camelCase pour les noms de variables et la notation PascalCase pour les noms des modules : ce sont les notations préconisées sous Visual Studio. Si vous écrivez un module commençant par une minuscule, sans signaler d'erreur, il informe juste que le nommage n'est pas correct :



Suivant les langages, on trouve des variantes.

Commentaires

Les commentaires permettent d'apporter des informations en clair pour aider à la compréhension du code. Certains commentaires sont ignorés par l'ordinateur (donc ne servent qu'à informer celui qui lit le code), d'autres commentaires apportent des informations interprétées par l'ordinateur (par exemple pour générer automatiquement la documentation technique). **Les différents types de commentaires** Il existe 3 types de commentaires sous C#:

```
/*
* commentaire informatif de plusieurs
* lignes
*/
// commentaire informatif d'une ligne
/**
* commentaire pouvant être interprété
* de plusieurs lignes
 */
/// commentaire pouvant être interprété, d'une ligne
```

Remarquez la différence de couleur (des étoiles et des barres) entre les commentaires interprétés et ceux non interprétés (vert pour "informatif", gris pour "interprété"). Autre remarque : les étoiles sur chaque ligne intermédiaire ne sont pas obligatoires

Important

Inutile de mettre beaucoup de commentaires non interprétés. Ne mettez pas un commentaire par ligne de code, mais plutôt un commentaire en tête d'un ensemble de lignes pour présenter le but du bloc concerné. Le code lui-même doit être clair à comprendre. Si ce n'est pas le cas, donc si le code est obscur, il doit être repensé.

Les commentaires interprétés

En revanche, les commentaires interprétés en tête de modules sont indispensables et doivent préciser le rôle du module, des paramètres et, dans le cas d'une fonction qui retourne une valeur, le rôle du retour. Visual Studio apporte une aide à la création de commentaires de modules, dans le but de la génération automatique de la documentation technique au format XML. Il suffit de placer le curseur une ligne au-dessus du module, de valider pour que le curseur se place au bon niveau d'indentation, et de taper "///". On obtient automatiquement la structure suivante :

EXEMPLE DE STRUCTURE DE COMMENTAIRE D'UN MODULE SOUS C#

```
/// <summary>
///
/// </summary>
/// <param name="a"></param>
/// <param name="b"></param>
/// <returns></returns> static
int Min(int a, int b)
{
    if (a < b)
    {
        return a;
    } else
    {
        return b;
    }
}
```

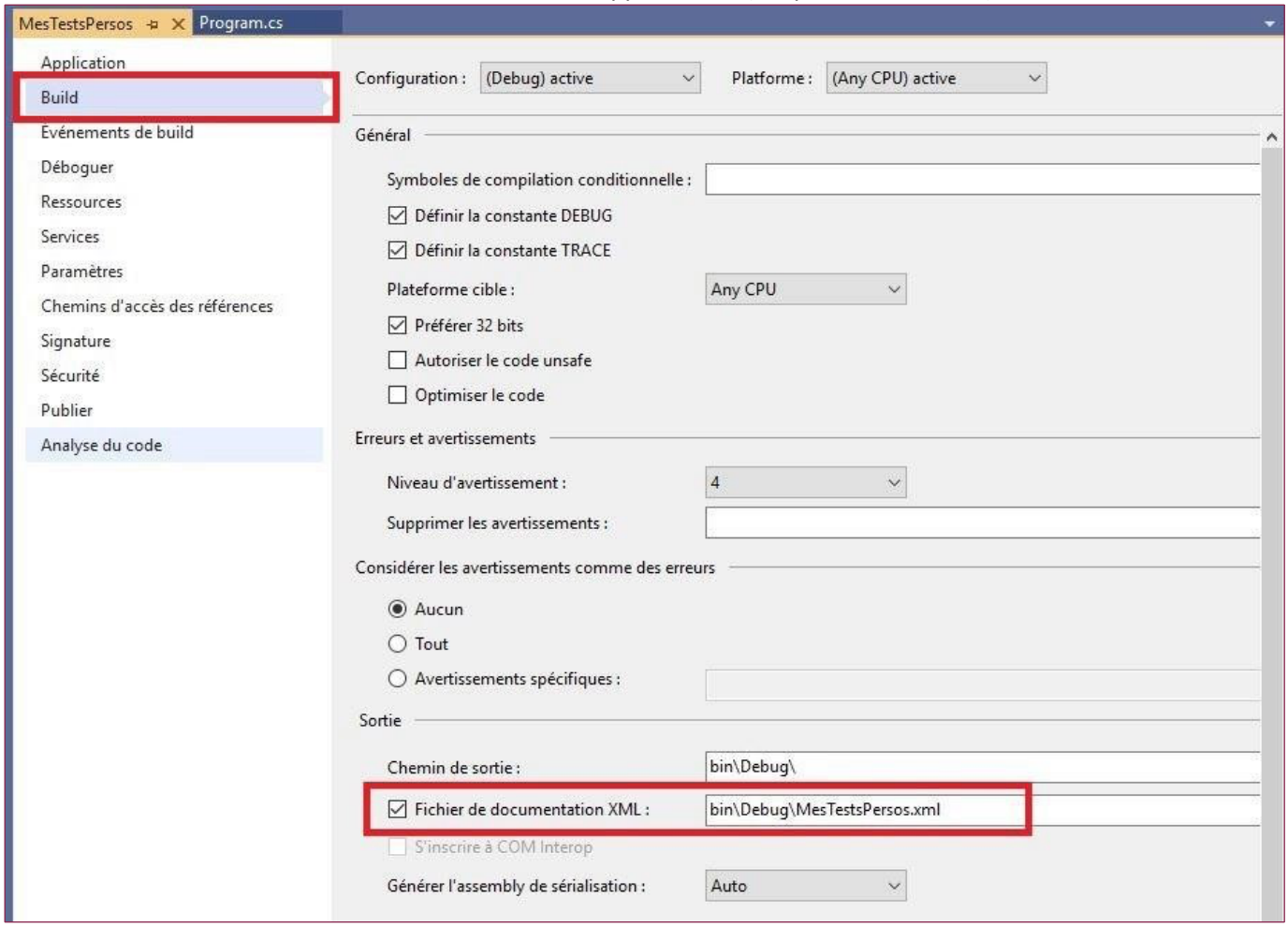
Il ne reste alors plus qu'à remplir avec les informations que l'on désire retrouver dans la documentation technique.

EXEMPLE DE COMMENTAIRE REMPLI D'UN MODULE SOUS C#

```
/// <summary>
/// recherche le minimum entre 2 entiers reçus en paramètre ///
</summary>
/// <param name="a">premier entier</param>
/// <param name="b">second entier</param>
/// <returns>le plus petit entre les 2 paramètres</returns> static
int Min(int a, int b)
{
    if (a < b)
    {
      return a;
    } else
    {
      return b;
    }
}
```

Génération de la documentation technique

Pour que la documentation technique soit automatiquement générée, il faut modifier le paramétrage du projet. Cela se fait dans le menu "Projet > Propriétés". Dans la partie "Build" (à gauche), il faut cocher la case "Fichier de documentation XML". Le chemin et le nom du fichier apparaissent (ce qui est modifiable).



Par défaut, le chemin est "bin\Debug\" et le nom du fichier de la documentation est le nom du projet avec l'extension "xml".

Voici à quoi ressemble le contenu du fichier, à partir de l'exemple de commentaire précédent :

EXEMPLE DE DOCUMENTATION TECHNIQUE SOUS C#

```
<?xml version="1.0"?>
<doc>
    <assembly>
        <name>MesTestsPersos</name>
    </assembly>
<members>
        <member name="M:MesTestsPersos.Program.Min(System.Int32,System.Int32)">
            <summary>
            recherche le minimum entre 2 entiers reçus en paramètre
            </summary>
            <param name="a">premier entier</param>
            <param name="b">second entier</param>
            <returns>le plus petit entre les 2 paramètres</returns>
</member>
    </members>
</doc>
```

Remarquez la balise "<member name" qui a été automatiquement remplie avec le nom du projet, suivi de "Program", du nom de la fonction "Min" et des types des paramètres attendus. La suite correspond aux informations qui ont été saisies dans la zone de commentaire.

Il faut ensuite utiliser des outils permettant d'exploiter le fichier xml afin de présenter son contenu proprement, par exemple sous forme d'un site. Cet aspect sera vu plus tard.

6B. Optimisation en temps

Le but est parfois d'optimiser le temps d'exécution d'un programme. Un même programme peut être écrit de plusieurs façons différentes : toutes les solutions peuvent donner un résultat identique à la fin, mais pas avec le même temps d'exécution.

La différence est parfois minime mais lorsqu'on se place à un autre niveau (par exemple des centaines d'internautes qui essayent d'exécuter la même fonctionnalité), on peut obtenir des différences de temps très significatives.

Il faut donc essayer de supprimer au maximum les redondances, les traitements inutiles et optimiser ceux qui sont gourmands en ressources.

Le principe consiste à compter le nombre d'instructions qui s'exécutent (qui peut être variable dès qu'il y a des alternatives et surtout des itérations). Le nombre d'instructions peut vite exploser s'il y a des itérations imbriquées. Il faut chercher à minimiser le nombre d'instructions exécutées, en particulier en analysant les itérations utilisées et leur pertinence.

Par exemple, méfiez-vous des boucles imbriquées : si vous avez une itération qui va boucler 20 fois et qui contient une autre itération qui va boucler 30 fois, l'air de rien le nombre de boucles s'élève à 20x30 donc 600 fois !

EXEMPLE

```
// première version : 400 boucles, 400 tests, 401 affectations
int somme = 0;
for(int x = 0; x < 20; x++)
{
    for(int y = 0; y < 20; y++)
    {
        if (y > x)
        {
            somme += y;
        }
    }
}
// deuxième version : 200 boucles, 0 tests, 201
affectations somme = 0;
for (int x = 0; x < 19; x++)
{
    for (int y = x+1; y < 20; y++)
    {
        somme += y;
    }
}
```

Vous pouvez tester sous Visual Studio : vous obtiendrez dans les 2 cas 2470 dans somme. Pourtant, la seconde solution est 3 fois plus rapide. Cela vient d'une meilleure analyse des traitements effectués.
Dans la première solution, les 2 boucles vont de 0 à 19 mais l'ajout dans somme ne se fait que quand y est supérieur à x. Donc, autant faire démarrer la seconde boucle à partir de x+1 (la première valeur supérieure à x). Cela permet de diminuer le nombre de boucles et d'éliminer le test.

6C. Optimisation en espace

L'espace représente la place de stockage. Pour le moment, vous n'avez abordé que le stockage en mémoire centrale (des différentes variables). Vous verrez plus tard d'autres types de stockage, sur supports permanents (disque dur…).
Actuellement, suivant les types de supports, l'optimisation en espace peut être plus ou moins importante : sur un ordinateur classique, les espaces de stockage sont assez importants, mais ce n'est pas le cas pour tous les supports, dont certains objets connectés.

6D. Optimisation fonctionnelle

Le but est d'éviter les redondances au niveau du code. Pour cela, il faut repérer les répétitions identiques ou similaires et, lorsque cela est possible, créer des modules paramétrés pour les remplacer. La notion de "module" sera abordée dans la séance suivante.
Pour le moment, vous pouvez juste faire en sorte de minimiser les redondances dans le code. Par exemple, si vous devez afficher plusieurs fois la même phrase avec juste une valeur qui change, autant afficher la phrase qu'une seule fois et utiliser une variable pour la valeur.

Exercice 19

Créez un nouveau projet Exercice19.

Le programme doit permettre de saisir un montant. Si le montant est supérieur à 40€, la remise est de 10%. De 20 à 40€, elle est de 5%. Le taux de remise et le montant à payer doivent être affichés sous la forme :

"montant = 45,5 avec une remise de 10%"

Voici le code proposé pour résoudre ce problème (le programme fonctionne correctement) :

```
Console.Write("Entrez un montant = ");
float montant = float.Parse(Console.ReadLine());
if (montant > 40)
{
    montant = montant * 90 / 100;
    Console.WriteLine("montant = " + montant + " avec une remise de 10%");
}
if (montant >= 20 && montant <= 40)
{
    montant = montant * 95 / 100;
    Console.WriteLine("montant = " + montant + " avec une remise de 5%");
}
if (montant < 20)
{
    Console.WriteLine("montant = " + montant + " avec une remise de 0%");
}
Console.ReadLine();
```

Proposez une version optimisée au niveau des tests et des répétitions de code.

7. Exercices récapitulatifs

Dans le domaine de la programmation, plus vous coderez, plus vous serez à l'aise. Surtout ne regardez pas les corrections directement. Vous devez tenter de coder par vous-même en utilisant les outils mis à votre disposition pour corriger le programme (l'IDE vous aide pour trouver les erreurs de compilation, ainsi que les erreurs d'exécution et de logique avec le débogueur).
Les exercices sont progressifs.

Exercice 20

Créez un nouveau projet Exercice20.
Le programme doit permettre de saisir un nom de ville et un nombre d'habitants, pour afficher au final un message du type :
`"Grasse possède 35000 habitants."`

Exercice 21

Créez un nouveau projet Exercice21.
Le programme doit permettre de récupérer une couleur de feu tricolore (officiellement par un capteur, mais ce sera ici sous forme de saisie de R pour rouge, O pour orange et V pour vert) et d'afficher l'ordre qui doit être exécuté : "s'arrêter", "ralentir", "passer". La saisie de la couleur doit être contrôlée. Essayez de réaliser 2 versions du programme : une avec des "if" imbriqués et une avec un switch.

Exercice 22

Créez un nouveau projet Exercice22.
Le programme doit permettre de saisir 10 notes puis d'afficher le nombre de notes supérieures ou égales à la moyenne (10) et le nombre de notes inférieures à la moyenne. Il n'est pas demandé de contrôler la saisie.

Exercice 23

Créez un nouveau projet Exercice23.
Le programme doit permettre de saisir plusieurs notes (entre 0 et 20) puis d'afficher la plus petite et la plus grande. Pour savoir combien de notes doivent être saisies, on posera d'abord la question à l'utilisateur. Il n'est pas demandé de contrôler les saisies.

Exercice 24

Créez un nouveau projet Exercice24.
Le programme doit permettre de saisir 10 températures (négatives ou positives, et pas de bornes fixées) puis d'afficher la plus petite et la plus grande.

Exercice 25

Créez un nouveau projet Exercice25.
Le programme doit permettre de saisir plusieurs valeurs numériques (on arrêtera la saisie en posant la question à l'utilisateur entre chaque saisie) puis au final de dire si la suite de valeurs saisies est strictement croissante (les valeurs ont été saisies de la plus petite à la plus grande). Les contrôles de saisie ne sont pas demandés.
Aide : pour voir si la suite est croissante, il faut comparer chaque valeur saisie avec la précédente. Donc il faut en permanence mémoriser 2 valeurs (la valeur actuelle et la précédente). Dès que 2 valeurs ne sont pas dans le bon ordre, c'est toute la suite qui n'est pas croissante (pensez à utiliser une variable booléenne pour mémoriser le fait que la suite est croissante ou non).

Exercice 26

Créez un nouveau projet Exercice26.
Le programme doit permettre de saisir un nombre de secondes (entre 0 et 86400) et d'afficher la conversion sous forme HH:MM:SS. Faites en sorte qu'il y ait bien 2 positions pour chaque valeur, donc éventuellement en ajoutant 0 quand cela est nécessaire. Exemple : 13205 secondes donnent 03:40:05 (et non pas 3:40:5)

CORRECTIONS

1. Correction des entraînements

Entraînement 1

Voici plusieurs déclarations :

```
string a, b;
char c;
int d;
byte e;
double f;
bool g;
```

Pour chaque instruction, dites si elle est correcte :

	correct	
`a = "bonjour";`	☑	
`b = "bonsoir";`	☑	
`c = "z";`	☐	`c = 'z';` (le caractère doit être mis entre apostrophes)
`d = 300;`	☑	
`e = 300;`	☐	le type byte va de 0 à 255
`f = 300;`	☑	
`g = "false";`	☐	`g = false;` (pas de guillemets)
`a = b;`	☑	
`b = d;`	☐	une chaîne ne peut pas recevoir une valeur numérique
`d = e;`	☑	

Entraînement 2

Voici plusieurs déclarations et initialisations :

```
int a = 20, b = 5;
float c = 3.5f;
string d = "merci";
bool e = true;
```

Pour chaque instruction, précisez ce qui va être affiché :

`Console.WriteLine(a);`	20
`Console.WriteLine(a + b);`	25
`Console.WriteLine(a + c);`	23,5
`Console.WriteLine(a + " " + b);`	20 5
`Console.WriteLine(a + d);`	20merci
`Console.WriteLine(a + " " + d);`	20 merci
`Console.WriteLine(e);`	True
`Console.WriteLine(a == b);`	False
`Console.WriteLine(d + "à tous");`	merci à tous

Entraînement 3

Voici plusieurs déclarations : string

```
a, b;
char c; int d;
byte e;
double f;
```

Pour chaque instruction, donnez la bonne syntaxe, dans le cas où celle proposée n'est pas correcte : a

```
= Console.ReadLine();
b = int.Parse(Console.ReadLine());
b = Console.ReadLine();
c  c = char.Parse(Console.ReadLine());
d = Console.ReadKey().KeyChar;
e = Console.ReadLine();   d = int.Parse(Console.ReadLine()); e =
int.Parse(Console.ReadLine());  e = byte.Parse(Console.ReadLine()); f =
double.Parse(Console.ReadLine());
```

Il suffit de bien regarder les types des variables. Par défaut, la saisie retourne un type string. Donc il ne faut pas de conversion si la variable est de type string. Pour toutes les autres variables, il faut convertir en respectant le type.

Entraînement 4

Voici plusieurs déclarations et initialisations :

```
int a = 34, b = 5, c = 4, d, e, f;
float g;
```

Voici une succession de traitements. A chaque ligne, précisez la valeur de la variable à gauche de l'affectation :

```
d = a / c;  d = 34 / 8 // division entière
e = a % c;  e = 34 % 2 = 2 // reste de la division
b *= c;                    b = 5 * 4 = 20
b++;     b = 20 + 1 = 21 f = d * c + e;
f = 8 * 4 + 2 = 34 g = a / c;
g = 34 / 4 = 8 !!! // et non pas 8,5
```

Si vous vous êtes trompé sur le dernier calcul, c'est normal. Le résultat de cette division est bien 8,5 et on affecte ce résultat dans une variable de type float, donc on pourrait s'attendre à ce que la variable contienne 8,5. En fait, C# part garde les types des variables du calcul : puisque vous divisez un entier par un entier, il retourne un entier. Dans le calcul, il suffit qu'une des valeurs soit de type float pour que le résultat soit bien de type float. Pour convertir la variable de type int en type float, vous ne pouvez pas utiliser float.Parse qui sert à convertir un string en float. Il faut utiliser une autre méthode qui fonctionne sur des types dits "compatibles" : `g = (float)a / c;`

Cette fois, g contiendra bien 8,5.

Entraînement 5

Voici plusieurs déclarations :

```
bool a;
int b = 3, c = 2, d = 1;
```

À chaque affectation dans la variable 'a', précisez sa valeur :

```
a = (b - c == d);                  a = (3 - 2 == 1) = (1 == 1) = true
a = (b <= c) || (c >= d);          a = (3 <= 2) || (2 >= 1) = true
a = !((b <= c) || (c >= d));       a = !((3 <= 2) || (2 >= 1)) = false
a = !(b <= c) || (c >= d);         a = !(3 <= 2) || (2 >= 1) = true
a = !((b == c) && (b > d));        a = !((3 == 2) && (3 > 1)) = true
a = b >= (c + d);                  a = 3 >= (2 + 1) = 3 >= 3 = true
```

Les tests vrais sont surlignés en vert, les tests faux sont surlignés en rouge. Attention, il faut ensuite tenir compte du reste (par exemple, s'il y a 2 tests séparés par '||', donc 'ou', alors il suffit qu'un seul test soit vrai pour que le test complet soit vrai).

Attention à la position des parenthèses (observez la 3ème est 4ème ligne où le '!' ne porte pas sur la même partie du test).

Entraînement 6

Sans tester sur ordinateur, donnez les valeurs de a, b et c à la fin de l'exécution de cette séquence :

```
int a = 3, b = 2, c = 1;
if(a > b || b < c) // vrai
{
    b++;
    if(!(a != b)) // vrai
    {
        c = c * 3;
    }
    else
    {
        a -= 2;
        b = b - 1;
    }
}
else
{
    a += 3;
    b = c + 1;
    c -= 1;
}

a = 3
b = 3
c = 3
```

Attention au second test : à cet endroit, les variables 'a' et 'b' sont égaux, donc le test (a != b) est faux, mais le '!' devant le test permet d'inverser le résultat, voilà pourquoi le test est vrai.

Entraînement 7

Voici un extrait de code. Sans tester sur ordinateur, donnez les valeurs de a, b et c en fin d'exécution.

```
int a = 3, b = 2, c = 1;
while (a > b && b >= 0)
{
    if(a == c)
    {
        c++;
        b++;
    }
    else
    {
        a--;
        b--;
    }
}

a = 1
b = 1
c = 2
```

Si vous n'avez pas trouvé ce résultat, testez directement sur Visual Studio en faisant une exécution pas à pas pour voir par où passe le programme et surveiller l'évolution des variables.

Entraînement 8

Voici un extrait de code. Sans tester sur ordinateur, donnez les valeurs de a, b et c en fin d'exécution.

```
int a = 3, b = 2, c = 1;
do
{
    if (a > c)
    {
        b--;
        c++;
    }
    else
    {
        b += 3;
    }
} while (a != b || a != c);

a = 3
b = 3
c = 3
```

Si vous n'avez pas trouvé ce résultat, testez directement sur Visual Studio en faisant une exécution pas à pas pour voir par où passe le programme et surveiller l'évolution des variables.

Entraînement 9

Voici un extrait de code. Sans tester sur ordinateur, donnez les valeurs de a, b et c en fin d'exécution.

```
int a = 3, b = 2, c = 1;
for (int j=0 ; j < 5 ; j++)
{
    if (a > j)
    {
        b++;
    }
    else
    {
        c++;
        b--;
    }
}

a = 3
b = 3
c = 3
```

Si vous n'avez pas trouvé ce résultat, testez directement sur Visual Studio en faisant une exécution pas à pas pour voir par où passe le programme et surveiller l'évolution des variables.

Entraînement 10

Voici un extrait de code. Sans tester sur ordinateur, donnez les valeurs de chaque variable.

```
int a, b;
string w, x, y, z = "programmer, c'est facile";
bool e, f;
a = z.Length;                                   a = 24
b = z.IndexOf(",");                             b = 10
w = z.Insert(17, "très");                       w = "programmer, c'est très facile"
x = z.Replace('p', 'P');                        x = "Programmer, c'est facile"
y = z.Substring(12, 5);                         y = "c'est"
e = z.Contains(" ,");                           e = false
f = z.EndsWith(z.Substring(z.IndexOf("f"),6)); f = true
```

Si vous n'avez pas trouvé ce résultat, testez directement sur Visual Studio en faisant une exécution pas à pas pour voir par où passe le programme et surveiller l'évolution des variables.

Entraînement 11

Voici un extrait de code. Sans tester sur ordinateur, donnez les valeurs de chaque variable.

```
int a, b, c = 4;
double d, e;
a = Math.Max(3, c);    a = 4 b = Math.Abs(c - 10);  b = 6
d = Math.Round((double)(c * a) / 2);    d = 8 e =
Math.Pow((double)c - 2, Math.Min(c, 2));      e = 4
```

Si vous n'avez pas trouvé ce résultat, testez directement sur Visual Studio en faisant une exécution pas à pas pour voir par où passe le programme et surveiller l'évolution des variables.

2. Correction des exercices

Exercice 1

Code de la correction :

```
static void Main(string[] args)
{
    // déclarations et initialisations
string prenom = "Alain";     string
nom = "DUPONT";     int age = 19;
    // affichage personnalisé
    Console.WriteLine("Bonjour" + prenom + " " + nom + ", vous avez " + age + "
ans.");
    Console.ReadLine();
}
```

Explications :

L'affichage peut se faire comme ci -dessus en une seule ligne, en combinant les chaînes en dur (mi ses entre guillemets et qui apparaissent en rouge dans le code) et les variables. Le signe + permet de concaténer les chaînes (les coller les unes à la suite des autres). Tout ce qui est entre guillemets est affiché en l'état. Tout ce qui n'est pas entre guillemets sont des variables : c'est leur contenu qui est affiché.

Il est bien sûr possible aussi de faire l'affichage avec plusieurs `Console.WriteLine`. Il est possible aussi de séparer les déclarations des initialisations.

Exercice 2

Code de la correction :

```
static void Main(string[] args)
{
    // déclarations
string prenom, nom;
int age;
    // saisie des 3 informations
    Console.Write("Entrez votre prénom : ");
prenom = Console.ReadLine();
    Console.Write("Entrez votre nom : ");
nom = Console.ReadLine();
    Console.Write("Entrez votre âge : ");
age = int.Parse(Console.ReadLine());
    // affichage personnalisé
    Console.WriteLine("Bonjour " + prenom + " " + nom + ", vous avez " + age + "
ans.");     Console.ReadLine();
```

Explications :

Le principe est le même excepté que les variables sont remplies avec des saisies et non directement initialisées. Remarquez le Parse pour l'âge car ce qui est saisi est toujours de type string et il faut changer son type pour l'affecter dans la vari able age qui est de type int.

Exercice 3

Code de la correction :

```
static void Main(string[] args)
{
    // déclarations
    float totalNotes, moyenne;
    int nbNotes;

    // saisie des informations
    Console.Write("Entrez la somme des notes = ");
    totalNotes = float.Parse(Console.ReadLine());
    Console.Write("Entrez le nombre de notes = ");
    nbNotes = int.Parse(Console.ReadLine());

    // calcul de la moyenne
    moyenne = totalNotes / nbNotes;

    // affichage de la moyenne
    Console.WriteLine("Moyenne = " + moyenne);
    Console.ReadLine();
}
```

Explications :
Par rapport aux notions précédentes, il y a juste un calcul supplémentaire. À noter que le programme peut provoquer une erreur : dans le cas où le nombre de notes saisi est 0.
Cependant, il peut aussi y avoir une erreur dès la saisie, si elle n'est pas conforme au type attendu (si l'utilisateur tape un caractère, par exemple).
On apprendra plus tard à éviter ce genre d'erreur. En effet, un programme correctement écrit ne doit jamais provoquer d'erreur, quelles que soient les manipulations de l'utilisateur.

Exercice 4

Code de la correction :

```
static void Main(string[] args)
{
    // Déclarations
    float prixht, tva;

    // Saisie du prix HT et de la TVA
    Console.Write("Entrez le prix HT (ex : 25,43) = ");
    prixht = float.Parse(Console.ReadLine());
    Console.Write("Entrez le taux de TVA (ex : 19,6) = ");
    tva = float.Parse(Console.ReadLine());

    // Affichage du prix TTC
    Console.WriteLine("prix TTC = " + (prixht * (1 + (tva / 100))));
    Console.ReadLine();
}
```

Explications :
Ce programme présente encore un calcul. La solution ci-dessus montre que le calcul peut être directement intégré dans l'affichage. Vous avez peut-être séparé le calcul en l'affectant dans un premier temps dans une variable, avant de l'afficher, ce qui est tout à fait correct.

{

Exercice 5

Code de la correction :

```
static void Main(string[] args)
{
    // Déclaration
    int age;
    // Saisie de l'âge
    Console.Write("Entrez votre âge = ");
     age = int.Parse(Console.ReadLine());
    // Test sur l'âge
    if (age >= 18)
    {
        Console.WriteLine("Vous êtes majeur");
    }     else
    {
        Console.WriteLine("Vous êtes mineur");
        Console.WriteLine("Vous serez majeur dans " + (18 - age) + " an(s)");
    }
    Console.ReadLine();
}
```

Explications :

Ce programme est le premier qui contient une alternative. Dans la partie "else", puisque la personne est mineure, il est possible de facilement calculer le nombre d'années manquantes avant la majorité. Le calcul a été directement intégré dans l'affichage (remarquez les parenthèses pour ne pas confondre avec la concaténation des chaînes). Si vous avez réalisé le calcul dans une variable, c'est tout à fait juste.

En revanche, ce qui fonctionne mais n'est pas correct, c'est refaire un test dans le "else" pour contrôler que l'âge est inférieur à 18. C'est une erreur très classique. Prenez conscience que si vous êtes dans le "else", c'est que la condition du "if" est fausse, donc que age est bien strictement inférieur à 18. Il est donc inutile de le retester. Donc voilà ce qu'il ne fallait pas faire :

```
if (age >= 18)
{
    Console.WriteLine("Vous êtes majeur");
} else
{
    if (age < 18) // test totalement inutile
    {
        Console.WriteLine("Vous êtes mineur");
        Console.WriteLine("Vous serez majeur dans " + (18 - age) + " an(s)");
}
}
```

Autre cas : si vous avez fait deux tests séparés, le programme va fonctionner mais il n'est pas optimisé. En effet, les deux tests vont être évalués alors qu'un seul suffit. Voilà donc aussi ce qu'il ne fallait pas faire :

```
{
    Console.WriteLine("Vous êtes majeur");
}
if (age < 18) // test inutile puisqu'on peut utiliser le précédent
    Console.WriteLine("Vous êtes mineur");
    Console.WriteLine("Vous serez majeur dans " + (18 - age) + " an(s)");
```

Surtout ne vous dites pas "mon programme marche c'est le plus important". Si vous voulez devenir un bon programmeur, vous devez faire des programmes qui marchent et qui sont efficaces.

Exercice 6

Code de la correction :

```
static void Main(string[] args)
{
    // Déclaration
    float moyenne;

    // Saisie de la moyenne
    Console.Write("Entrez la moyenne = ");
    moyenne = float.Parse(Console.ReadLine());

    // Test de la moyenne et affichage de la mention
    if (moyenne < 10)
    {
        Console.WriteLine("recallé");
    }
    else
    {
        if (moyenne < 12)
        {
            Console.WriteLine("passable");
        }
        else
        {
            if (moyenne < 14)
            {
                Console.WriteLine("assez bien");
            }
            else
            {
                if (moyenne < 16)
                {
                    Console.WriteLine("bien");
                }
                else
                {
                    Console.WriteLine("très bien");
                }
            }
        }
    }
    Console.ReadLine();
}
```

Explications :
Voilà un exemple classique d'imbrications de "if". À chaque fois qu'on est dans le "else", on est dans le contraire de la condition du "if", c'est donc inutile de retester (comme cela a été expliqué dans l'exercice précédent).
Quand, comme ici, ce sont des plages de valeurs à tester, vous pouvez tester du plus petit au plus grand, ou inversement.

— Exercice 7

Code de la correction :

```
static void Main(string[] args)
{
    // Déclarations
     float prix, total = 0;
    // Saisie d'un premier prix
    Console.Write("Entrez un prix (0 pour terminer) = ");
    prix = float.Parse(Console.ReadLine());
    // Boucle sur la saisie des prix et le cumul
    while (prix != 0)
    {
       total = total + prix;
       // saisie d'un nouveau prix
       Console.Write("Entrez un prix (0 pour terminer) = ");
       prix = float.Parse(Console.ReadLine());
    }
    // Affichage du total
    Console.WriteLine("total des prix = " + total);
    Console.ReadLine();
}
```

Explications :

C'est le premier exemple d'itération. Remarquez la première saisie de prix avant la boucle, puis l'autre saisie en fin de boucle. C'est la saisie dans la boucle qui va se répéter plusieurs fois. Après chaque saisie, on teste si 0 a été entré : c'est la condition pour sortir de la boucle.

Cet exercice utilise aussi une variable de cumul, nommée total. Elle est obligatoirement initialisée avant la boucle (ici, 0 lui est affecté) car elle est utilisée par la suite dans la boucle. Si vous n'initialisez pas à 0 la variable total, lors du premier passage dans la boucle, il y a un souci car total+prix ne peut être effectué (puisque total ne contient rien).

— Exercice 8

Code de la correction :

```
static void Main(string[] args)
{
    // Déclarations
     float prix, total = 0;
     char reponse;
    // demande si un prix est à saisir
    Console.Write("Avez-vous un prix à saisir ? (O/N) ");
    reponse = Console.ReadKey().KeyChar;
    // Boucle sur la saisie des prix et le cumul
     while (reponse=='O')
    {
        // saisie d'un nouveau prix
        Console.Write("   Entrez un prix = ");
        prix = float.Parse(Console.ReadLine());
        // cumul
        total = total + prix;
        // demande si un nouveau prix est à saisir
        Console.Write("Avez-vous un prix à saisir ? (O/N) ");
         reponse = Console.ReadKey().KeyChar;
    }
    // Affichage du total
```

```
    Console.ReadLine();
}
```

Explications :

La saisie de la réponse se fait sur un caractère : c'est plus pratique. Du coup, faites attention, la comparaison se fait avec 'O' entre 2 apostrophes et non entre guillemets. C# fait la différence entre 'O' et "O" : le premier est un caractère, le second est une chaîne ne contenant qu'un caractère.

Ne soyez pas étonné des espaces laissés en début de chaîne " Entrez un prix = " : c'est juste pour la présentation pour éviter que ce soit collé au 'O' qui est saisi juste avant. Vous avez peut-être plutôt géré un retour à la ligne juste avant.

Exercice 9

Code de la correction :

```
static void Main(string[] args)
{
    // Déclarations
int note;
    // Boucle sur le contrôle de la saisie de la note
do
    {
        Console.Write("Entrez une note entre 0 et 20 = ");
note = int.Parse(Console.ReadLine());     } while (note <
0 || note > 20);
    // Affichage de la note
    Console.WriteLine("note saisie : " + note);
    Console.ReadLine();
}
```

Explications :

Ce genre de boucle est idéal pour tester une saisie, car justement il faut d'abord saisir avant de tester.

Dans le test de la boucle, vous avez peut-être d'abord fait l'erreur de mettre && (et) au lieu de || (ou). C'est bien || (ou) car on ne peut pas avoir une note qui est à la fois inférieure à 0 et supérieure à 20…

Vous avez peut-être utilisé des types différents de ceux utilisés dans la correction, et parfois même mieux adapté. D'ailleurs ici, le type limite aux notes entières, ce qui ne reflète pas forcément la réalité.

```
    Console.WriteLine("   total des prix = " + total);
```

Exercice 10

Code de la correction :

```
static void Main(string[] args)
{
    // Déclaration
    char sexe;

    // Boucle sur la saisie correcte du sexe
    do
    {
        Console.WriteLine();
        Console.Write("Quel est votre sexe ? (H/F) ");
        sexe = Console.ReadKey().KeyChar;
    } while (sexe != 'H' && sexe != 'F');

    // Affichage du message personnalisé
    Console.WriteLine();
    if (sexe == 'H')
    {
        Console.WriteLine("Bonjour monsieur");
    }
    else
    {
        Console.WriteLine("Bonjour madame");
    }
    Console.ReadLine();
}
```

Explications :
Là aussi, un test de saisie est demandé, d'où la boucle "do" la plus adaptée.
Si vous avez utilisé un type string plutôt que char pour saisir le sexe, ce n'est pas faux mais moins pratique pour la saisie.

Exercice 11

Code de la correction :

```
static void Main(string[] args)
{
    // Déclarations
    float note, somme = 0;

    // Saisie des 5 notes et cumul
    for (int k = 1; k <= 5; k++)
    {
        Console.Write("Entrez une note = ");
        note = float.Parse(Console.ReadLine());
        somme += note;
    }

    // Affichage de la moyenne
    Console.WriteLine("moyenne = " + (somme / 5));
    Console.ReadLine();
}
```

Explications :
Sachant le nombre d'itérations, la boucle "for" est la plus adaptée (même si les autres boucles fonctionnent).
La correction utilise un raccourci d'écriture pour faire la somme des notes. Vous pouvez aussi écrire :

```
        somme = somme + note;
```

Encore une fois, le calcul qui est directement fait dans l'affichage, peut être d'abord transféré dans une variable.

Exercice 12

Code de la correction :

```
static void Main(string[] args)
{
    // boucle sur la table de multiplication
    for(int k = 0; k <= 10; k++)
    {
        Console.WriteLine("3 x " + k + " = " + (3 * k));
    }
    Console.ReadLine();
}
```

Explications :
L'affichage peut se faire en une seule ligne comme ci-dessus, ou en plusieurs instructions (en utilisant `Console.Write` pour éviter les retours à la ligne, sauf pour le dernier affichage).

Exercice 13

Code de la correction :

```
static void Main(string[] args)
{
       // Déclaration
    int entier;

    // Saisie d'un entier entre 1 et 9 avec contrôle de saisie
    do
    {
        Console.Write("Entrez un entier entre 1 et 9 = ");
        entier = int.Parse(Console.ReadLine());
    } while (entier < 1 || entier > 9);

    // boucle sur la table de multiplication de cet entier
    for (int k = 0; k <= 10; k++)
    {
        Console.WriteLine(entier+" x " + k + " = " + (entier * k));
    }
    Console.ReadLine();
}
```

Explications :
Il faut utiliser 2 boucles indépendantes l'une de l'autre. La première permet de contrôler la saisie, la seconde permet l'affichage de la table de multiplication.

— Exercice 14

Code de la correction :

```
static void Main(string[] args)
{
    // Déclaration
     int entier;
     char reponse;
    // Boucle sur plusieurs tables de multiplications
     do
    {
        // Saisie d'un entier entre 1 et 9 avec contrôle de saisie
         do
        {
            Console.Write("Entrez un entier entre 1 et 9  = ");
            entier = int.Parse(Console.ReadLine());
         } while (entier < 1 || entier > 9);
        // boucle sur la table de multiplication de cet entier
         for (int k = 0; k <= 10; k++)
        {
            Console.WriteLine(entier + " x " + k + " = " + (entier * k));
         }
        // Demande pour l'affichage d'une table de multiplication
        do
        {
            Console.WriteLine();
            Console.Write("Voulez-vous afficher une nouvelle table de
multiplication ? (O/N) ");
            reponse = Console.ReadKey().KeyChar;
         } while (reponse!='O' && reponse!='N');
        // Effacer l'écran         Console.Clear();
    } while (reponse=='O');
}
```

Explications :

Le programme est plus complexe car il faut arriver à gérer la grande boucle qui contient les autres traitements. Si vous n'êtes pas arrivé à le faire, prenez le temps de bien le comprendre. La grande boucle générale permet de répéter plusieurs fois l'ensemble des traitements, donc de pouvoir obtenir l'affichage de plusieurs tables de multiplications. C'est une boucle "do" car on veut afficher au moins une table. Dans cette grande boucle, on retrouve les traitements du programme précédent, avec en plus la question pour continuer ou non.

Exercice 15

Code de la correction :

```
static void Main(string[] args)
{
    // Saisie de la phrase
    Console.WriteLine("Entrez une phrase : ");
    string phrase = Console.ReadLine();

    // Remplacement des espaces par _
    string newphrase = phrase.Replace(' ', '_');

    // Affichage de la nouvelle phrase
    Console.WriteLine(newphrase);
    Console.ReadLine();
}
```

Explications :
Pour changer, cette correction ne présente pas de déclarations séparées mais au moment de la première utilisation des variables. C'est pour vous montrer que c'est possible de faire ainsi.

Exercice 16

Code de la correction :

```
static void Main(string[] args)
{
    // Saisie d'une phrase
    Console.Write("Entrez une phrase = ");
    string phrase = Console.ReadLine();

    // Saisie d'un mot
    Console.Write("Entrez un mot présent dans la phrase = ");
    string mot = Console.ReadLine();

    // Recherche de la position mot dans la phrase
    int position = phrase.IndexOf(mot);

    // Teste si le mot a été trouvé
    if (position != -1)
    {
        // affichage de la phrase à partir de ce mot
        Console.WriteLine(phrase.Substring(position));
    }
    else
    {
        Console.WriteLine("le mot n'existe pas");
    }
    Console.ReadLine();
}
```

Explications :
Il est préférable de mémoriser la position du mot dans une variable, car elle va servir 2 fois : pour le test (pour savoir si le mot a été trouvé) et aussi pour extraire la sous-chaîne.
Vous avez peut-être fait l'extraction dans une variable avant l'affichage : c'est tout à fait juste.

Exercice 17

Code de la correction :

```
static void Main(string[] args)
{
    // saisie du nombre
    Console.Write("Entrez un nombre = ");
    int nombre = int.Parse(Console.ReadLine());

    // demande de la racine carrée
    Console.Write("Entrez la racine carrée = ");
    double racine = double.Parse(Console.ReadLine());

    // contrôle si la rasine carrée est correcte
    if (racine == Math.Sqrt(nombre))
    {
        Console.WriteLine("Bravo !");
    }
    else
    {
       Console.WriteLine("Erreur, la racine carrée de " + nombre + " est " + Math.
Sqrt(nombre));
    }
    Console.ReadLine();
}
```

Explications :
Vous avez peut-être utilisé une variable intermédiaire pour mémoriser la racine carrée afin d'éviter d'appeler 2 fois la fonction : c'est une bonne idée.
Vous n'avez peut-être même pas utilisé `Math.Sqrt` : effectivement, il est possible de comparer nombre avec racine*racine. C'est tout à fait juste aussi.

Exercice 18

Code de la correction :

```
static void Main(string[] args)
{
    // saisie des 3 nombres
    Console.Write("Entrez le nombre 1 = ");
double nb1 = double.Parse(Console.ReadLine());
Console.Write("Entrez le nombre 2 = ");     double
nb2 = double.Parse(Console.ReadLine());
Console.Write("Entrez le nombre 3 = ");     double
nb3 = double.Parse(Console.ReadLine());
    // affichage du plus petit des 3 nombres
    Console.WriteLine("Le plus petit est " + Math.Min(nb1, Math.Min(nb2, nb3)));
    Console.ReadLine();
}
```

Explications :
Vous avez peut-être d'abord stocké le plus petit de 2 nombres dans une variable, avant de chercher le plus petit du 3e nombre avec cette variable. C'est correct. Ici, tout est fait en une seule fois : on cherche le plus petit entre nb1 et le résultat de la recherche du plus petit entre nb1 et nb2. Cela vous permet de voir que ce que l'on met dans les parenthèses d'une fonction peut être le résultat d'une autre fonction.

L'erreur à ne pas faire est de penser qu'on peut mettre directement 3 valeurs dans les parenthèses : la fonction Min n'attend que 2 valeurs.

Exercice 19

Code de la correction :

```
static void Main(string[] args)
{
    // Saisie du montant
    Console.Write("Entrez un montant = ");
    float montant = float.Parse(Console.ReadLine());

    // recherche de la remise
    float remise = 0;
    if(montant > 40)
    {
        remise = 10;
    }
    else
    {
        if (montant >= 20)
        {
            remise = 5;
        }
    }

    // affichage de la remise et du montant final
    Console.WriteLine("montant = " + (montant * (1 - remise / 100)) + " avec une
remise de " + remise + "%");
    Console.ReadLine();
}
```

Explications :
L'affichage n'est fait qu'une seule fois en exploitant entre autres la variable remise qui est initialisée par les tests. Les tests imbriqués permettent de trouver la remise, ce qui évite le triple test précédent. L'initialisation à 0 de remise avant les tests permet d'éviter le test inférieur à 20.

Exercice 20

Code de la correction :

```
static void Main(string[] args)
{
    // saisie de la ville et du nombre d'habitants
    Console.Write("Entrez la ville = ");
    string ville = Console.ReadLine();
    Console.Write("Entrez le nombre d'habitants = ");
    int habitants = int.Parse(Console.ReadLine());

    // affichage du message
    Console.WriteLine(ville + " possède " + habitants + " habitants.");
    Console.ReadLine();
}
```

Explications :
Vous avez peut-être géré l'affichage en plusieurs `Console.Write`. C'est correct mais l'écriture ci-dessus est plus courte et surtout plus lisible.

Exercice 21

Code de la correction :

```
static void Main(string[] args)
{
    // saisie contrôlée du feu
    char feu;
    do
    {
        Console.WriteLine();
        Console.Write("feu (R/O/V) = ");
        feu = Console.ReadKey().KeyChar;
    } while (feu!='R' && feu!='O' && feu!='V');

    // Affichage de l'ordre
    Console.WriteLine();
    if (feu == 'R')
    {
        Console.WriteLine("s'arrêter");
    }
    else
    {
        if (feu == 'O')
        {
            Console.WriteLine("ralentir");
        }
        else
        {
            Console.WriteLine("passer");
        }
    }
    Console.ReadLine();
}
```

Explications :
Une boucle permet de tester la saisie du caractère. Il suffit ensuite d'imbriquer des tests. Remarquez que seuls 'R' et 'O' sont testés : en effet, si on est dans le dernier 'else', on est forcément dans le cas 'V' puisqu'on ne peut pas avoir saisi autre chose.

Voici le code de la version avec switch (exercice21bis) :

```
static void Main(string[] args)
{
    // saisie contrôlée du feu
    char feu;
    do
    {
        Console.WriteLine();
        Console.Write("feu (R/O/V) = ");
        feu = Console.ReadKey().KeyChar;
    } while (feu != 'R' && feu != 'O' && feu != 'V');

    // Affichage de l'ordre
    Console.WriteLine();
    switch (feu)
    {
        case 'R':
            Console.WriteLine("s'arrêter");
            break;
        case 'O':
            Console.WriteLine("ralentir");
            break;
        case 'V':
            Console.WriteLine("passer");
            break;
    }
    Console.ReadLine();
}
```

Explication :
Voici un cas assez rare où le switch se prête bien au traitement à effectuer. La variable doit être comparée à des valeurs précises et qui n'ont pas de lien logique les unes avec les autres (pas d'ordre alphabétique, par exemple).
N'oubliez pas le break qui permet de sortir du switch une fois le cas traité.
N'utilisez pas le switch si vous devez gérer des plages de valeur.

Exercice 22

Code de la correction :

```
static void Main(string[] args)
{
    // déclaration
    int note, nbsup = 0;

    // boucle sur les 10 notes
    for(int k = 1; k <= 10; k++)
    {
        // saisie d'une note
        Console.Write("note n°" + k + " = ");
        note = int.Parse(Console.ReadLine());
        // compte si la note est supérieure à la moyenne
        if (note >= 10)
        {
            nbsup++;
        }
    }

    // affichage du nombre de notes supérieures et inférieures à 10
    Console.WriteLine("Nombre de notes >= 10 : " + nbsup);
    Console.WriteLine("Nombre de notes < 10  : " + (10 - nbsup));
    Console.ReadLine();
}
```

Explications :
Il est bien sûr inutile de mémoriser les 10 notes : à chaque saisie, on contrôle si la note est supérieure ou égale à 10 pour compter combien de notes sont dans ce cas. La saisie suivante peut alors écraser la note précédemment saisie.
Remarquez aussi qu'un seul compteur est utilisé : une fois que l'on a le nombre de notes >= 10, un simple calcul donne le nombre de notes inférieures à 10.

Exercice 23

Code de la correction :

```
static void Main(string[] args)
{
    // déclaration
     int note;
    // saisie du nombre de notes
    Console.Write("Nombre de notes à saisir = ");
     int nbnotes = int.Parse(Console.ReadLine());
    // initialisation du min et max
     int min = 20, max = 0;
    // saisie des notes et recherche des min et max
     for (int k = 1; k <= nbnotes; k++)
    {
        // saisie de la note
```

```
            Console.Write("note n°" + k + " = ");
             note = int.Parse(Console.ReadLine());
```

```
// est-ce la nouvelle plus petite note ?
if (note < min)
        {
            min = note;
}
        // est-ce la nouvelle plus grande note ?
if (note > max)
        {
            max = note;
        }
    }
    // affichage des extrema
if (nbnotes > 0)
    {
        Console.WriteLine("La plus petite note est = " + min);
Console.WriteLine("La plus grande note est = " + max);
    }
    Console.ReadLine();
}
```

Explications :
La boucle "for" reste la plus adaptée car, mêmesi le nombre de notes n'est pas fixe, il est connu dès le début du programme après la 1ère saisie.
La recherche de la plus petite et de la plus grande valeur dans une série de valeurs se fait en comparant chaque valeur avec une valeur de référence. Par exemple, ici, pour la plus petite note, la variable min a été initialisée à 20 (car il ne peut pas y avoir de notes au-dessus de 20) et elle est comparée à chaque note saisie. Dès qu'on saisit une nouvelle note, il suffit de la comparer à min : si elle est plus petite, alors on est face à la nouvelle plus petite note, voilà pourquoi elle est transférée dans min. Même logique pour max.
Lorsqu'on recherche la plus petite et/ou la plus grande valeur d'une série, il peut y avoir 2 cas possibles :

- soit la série est bornée (comme ici, des valeurs entre 0 et 20) alors il suffit d'initialiser min et max avec les bornes ;
- soit la série est non bornée, alors il faut initialiser min et max avec une des valeurs de la série (généralement la 1ère valeur).

Encore une fois, on n'a pas besoin de mémoriser toutes les valeurs d'une série pour trouver la plus petite ou la plus grande valeur.
Enfin, remarquez le test final. En effet, si vous saisissez 0 notes, il ne faut pas afficher le min et le max.

— Exercice 24

Code de la correction :

```
static void Main(string[] args)
{
    // déclaration
    double temperature, min, max;
    // saisie de la 1ère température pour initialiser min et max
     Console.Write("température n°1 = ");
     min = max = double.Parse(Console.ReadLine());
    // saisie des autres températures et recherche des min et max
     for (int k = 2; k <= 10; k++)
    {
        // saisie de la temperature
        Console.Write("temperature n°" + k + " = ");
```

```
            temperature = double.Parse(Console.ReadLine());
            // est-ce la nouvelle plus petite température ?
            if (temperature < min)
            {
                min = temperature;
            }
            // est-ce la nouvelle plus grande température ?
            if (temperature > max)
            {
                max = temperature;
            }
        }
        // affichage des extrema
        Console.WriteLine("La plus petite température est = " + min);
        Console.WriteLine("La plus grande température est = " + max);
        Console.ReadLine();
    }
```

Explications :

Comme cela a été expliqué dans l'exercice précédent, lorsque la série n'est pas bornée, il faut alors initialiser min et max avec une des valeurs de la série. Généralement on isole la première valeur. Vous remarquerez d'ailleurs que je n'ai même pas transféré la saisie dans la variable temperature, mais directement dans min et dans max, puisqu'il n'y a pas de test à faire.

Exercice 25

Code de la correction :

```
static void Main(string[] args)
{
    // déclarations
    char reponse;
    bool croissant = true;
    double valeur, ancvaleur;

    // saisie d'une première valeur
    Console.Write("valeur = ");
    ancvaleur = double.Parse(Console.ReadLine());

    // question pour continuer
    Console.Write("Voulez-vous continuer ? (O/N) ");
    reponse = Console.ReadKey().KeyChar;

    // boucle sur la saisie des valeurs
    while (reponse == 'O')
    {
        // saisie d'une valeur
        Console.WriteLine();
        Console.Write("valeur = ");
        valeur = double.Parse(Console.ReadLine());

        // test de la valeur par rapport à la précédente
        if (valeur <= ancvaleur)
        {
            croissant = false;
        }

        // mémorisation de la valeur dans ancvaleur pour la comparaison suivante
        ancvaleur = valeur;

        // question pour continuer
        Console.Write("Voulez-vous continuer ? (O/N) ");
        reponse = Console.ReadKey().KeyChar;
    };

    // affichage du message final (suite croissante ou non)
    Console.WriteLine();
    if (croissant)
    {
        Console.WriteLine("OUI, la suite est strictement croissante");
    }
    else
    {
        Console.WriteLine("NON, la suite n'est pas strictement croissante");
    }
    Console.ReadLine();
}
```

Explications :
Le programme est plus difficile mais une aide est donnée dans l'énoncé et permet de savoir comment gérer le problème. Remarquez la première valeur saisie avant la boucle pour avoir une valeur d'avance afin de la comparer avec la nouvelle valeur saisie. Si l'utilisateur dit 'N' dès le début, une seule valeur sera saisie et du coup la suite sera croissante, ce qui est correct. Pour les autres cas, donc à partir de 2 valeurs saisies, il y a bien une comparaison des valeurs 2 à 2.
La variable croissant est à vrai au départ, et passe à faux dès que 2 valeurs ne sont pas dans l'ordre. Remarquez bien que cette variable ne peut jamais repasser à vrai, une fois qu'elle est à faux. Le test final peut aussi s'écrire :
L'écriture de la correction vous montre comment on peut tester un booléen. C'est une écriture

```
if (croissant == true)
```

raccourcie.

Exercice 26

Code de la correction :

```
static void Main(string[] args)
{
    // déclarations
    int secondes; // nombre de secondes à convertir
    string heure; // heure finale correspondante, sous forme "HH:MM:SS"

    // saisie des secondes avec conrôle de saisie
    do
    {
        Console.Write("Entrez le nombre de secondes (entre 0 et 86400) = ");
        secondes = int.Parse(Console.ReadLine());
    } while (secondes < 0 || secondes > 86400);

    // calcul des heures, minutes, secondes correspondantes
    int h = secondes / 3600;
    int m = (secondes - (h * 3600)) / 60;
    int s = secondes - (h * 3600) - (m * 60);

    // formatage de la chaîne à afficher (avec ajout de "0" si nécessaire
    heure = "";
    if (h<10)
    {
        heure += "0";
    }
    heure += h + ":";
    if (m < 10)
    {
        heure += "0";
    }
    heure += m + ":";
    if (s < 10)
    {
        heure += "0";
    }
    heure += s;

    // affichage final de l'heure
    Console.WriteLine(heure);
    Console.ReadLine();
}
```

Explications :
Il faut d'abord convertir les secondes en heures, minutes, secondes numériques (dans les variables h, m, s). Ensuite, pour le formatage de l'affichage, une seule variable chaîne est construite étape par étape. Remarquez les tests pour savoir s'il faut rajouter un "0" ou non.