

CÉGEP DU VIEUX MONTRÉAL

Département d'informatique



Notes de cours

420-C11-VM

Marcel L'Italien

TABLE DES MATIÈRES

CHAPITRE 1	4
1.1 La programmation.....	4
1.2 Le diagramme d'action	5
1.3 Le dictionnaire de données.....	6
1.4 La structure d'un programme en langage C.....	8
1.5 Les données	11
1.6 Les opérateurs.....	19
1.7 Exercices	29
 CHAPITRE 2	 33
2.1 Les opérations de sortie.....	33
2.2 Les opérations d'entrée	44
2.2.1 La commande d'entrée dans les diagrammes d'action	44
2.2.2 Le flux d'entrée cin	45
2.2.3 Les fonctions _getche() et _getch()	54
2.2.4 La fonction getline()	55
2.3 Les fonctions gotoxy() et clrscr()	56
2.4 La fonction MessageBoxA()	57
2.5 Exercices	60
 CHAPITRE 3	 69
3.1 La sélection.....	69
3.2 Les sélections dans les diagrammes d'action	69
3.3 La condition simple	70
3.4 La condition composée	71
3.5 Notion de bloc en langage C.....	71
3.6 L'instruction if()	72
3.7 L'instruction if()-else	79
3.8 Les fonctions cin.fail() , cin.clear() et cin.ignore()	88
3.9 La sélection multiple et les if()-else imbriqués	90
3.10 L'instruction switch()	97
3.11 Exercices	108

CHAPITRE 4	116
4.1 La répétition	116
4.2 Les répétitions dans les diagrammes d'action	116
4.3 L'énoncé Pour	117
4.4 L'instruction for()	118
4.5 L'énoncé Tant que	126
4.6 L'instruction while()	127
4.7 L'énoncé Faire...Tant que	138
4.8 L'instruction do...while()	139
4.9 La fonction time()	143
4.10 Exercices	147
 CHAPITRE 5	 150
5.1 Les variables dimensionnées.....	150
5.2 Les éléments d'une variable dimensionnée	151
5.3 La vérification des frontières	152
5.4 Les chaînes de caractères (string) et les variables dimensionnées.....	161
5.5 Les variables multidimensionnelles	170
5.6 Exercices	178
5.7 Exercices sur les vecteurs	183
5.8 Exercices sur les matrices	184
 ANNEXES	
1 Les opérateurs du C++	185
2 Les codes ASCII	188

CHAPITRE 1

LES FONDEMENTS DE LA PROGRAMMATION ET DU LANGAGE C++

1.1 LA PROGRAMMATION

L'objectif principal du cours 420-C11 est de vous introduire à la programmation. Nous avons choisi de centrer votre apprentissage sur les deux facettes suivantes: l'élaboration d'algorithmes simples et le codage de ces algorithmes en langage C++.

L'*informatique* consiste à résoudre un problème en trouvant une solution qui utilise l'ordinateur. Il s'agit donc de construire des programmes. Bien entendu, pour passer du problème à la solution programmée, il y a plusieurs tâches intermédiaires à réaliser. Ces tâches sont nécessaires pour garantir le succès de la solution choisie.

Que les problèmes soient complexes ou très simples, et afin de trouver une bonne solution il n'y a pas d'autre choix que de procéder méthodiquement. Il existe donc différentes méthodologies de développement de système sur le marché, nous guidant dans notre objectif de résoudre un problème. Même s'il existe plusieurs méthodes, elles ont toutes les mêmes grandes lignes directrices:

ÉTAPE 1: ANALYSER LE PROBLÈME

Cette étape consiste premièrement à bien comprendre la problématique et les objectifs que l'on vise. Dans un second temps on regroupera toutes les informations dont on dispose et on définira les grandes lignes de la solution. En étroite collaboration avec l'utilisateur, de qui provient le problème, on retravaillera sans cesse nos conclusions jusqu'au moment où les objectifs et les résultats à obtenir sont bien cernés, et que l'ébauche de la solution proposée est acceptée par l'utilisateur.

ÉTAPE 2: ÉLABORER LA SOLUTION

À partir des résultats de l'étape d'analyse, cette phase consiste à détailler et raffiner la solution, puis à la transcrire sous une forme permettant de la programmer facilement. Le résultat de cette étape est ce qu'on appelle un *algorithme*.

Pour écrire nos algorithmes nous avons choisi une méthode mise au point par James Martin. Cette méthode est le *diagramme d'action*. Nous compléterons les diagrammes d'action par un outil de base en informatique, soit le dictionnaire de données.

ÉTAPE 3: PROGRAMMER ET DOCUMENTER LA SOLUTION

Après avoir choisi le langage informatique approprié, on traduit l'algorithme dans ce langage. On se préoccupera aussi de bien documenter le programme, ainsi que son utilisation.

1.2 LE DIAGRAMME D'ACTION

Le diagramme d'action est la transposition de la logique d'un problème en un langage adapté. Le diagramme d'action permet d'exprimer la méthodologie pour la solution d'un problème. Le diagramme d'action décompose la solution d'un problème en une série d'étapes logiques. Ceci est fait avec des énoncés d'un langage propre à la construction des diagrammes d'action. Il s'agit en fait d'un plan, ou d'une recette à suivre pour résoudre le problème.

Pour construire un diagramme d'action, on utilisera un langage formel composé de six commandes ou mots-clés.

Lire
Écrire
Si... Sinon
Tant que ...
Faire... Tant que
Pour

Voici un exemple de diagramme d'action employant plusieurs mots-clés, ainsi que les éléments graphiques qui doivent les précéder. Cet exemple permet de calculer la moyenne des notes des étudiants d'un groupe, en procédant comme suit:

- . Obtenir le nombre d'étudiants dans le groupe
- . Obtenir la note de chaque étudiant et en faire la somme
- . Calculer la moyenne

```
Écrire "Combien d'étudiants avez-vous?"
Lire NbEtudiant
CompteNbEtudiant = 0
  Tantque CompteNbEtudiant < NbEtudiant
    Écrire "Donnez la note de l'étudiant:"
    Lire NoteEtudiant
    Si NoteEtudiant < 0
      Écrire "Erreur, une note doit être positive ou nulle"
    Sinon
      CompteNbEtudiant = CompteNbEtudiant + 1
      SommeNoteEtudiant = SommeNoteEtudiant + NoteEtudiant
    MoyenneNoteEtudiant = SommeNoteEtudiant / NbEtudiant
  Écrire "La moyenne de ce groupe est de:", MoyenneNoteEtudiant
```

Comme vous le remarquez, en plus des mots-clés, le diagramme d'action utilise des symboles graphiques. Afin de vous faciliter la tâche lors de la rédaction de vos diagrammes d'action avec *Word*, nous avons défini des macros permettant d'insérer automatiquement ces éléments graphiques dans votre texte.

1.3 LE DICTIONNAIRE DE DONNÉES

Le dictionnaire de données est l'élément complémentaire à la fois du diagramme d'action et du programme codé. C'est un bottin qui regroupe la liste de tous les éléments utilisés pour la résolution du problème et le codage de la solution. Nous avons besoin de plusieurs renseignements pour chaque élément. Les plus usuels sont le nom, le type, la valeur de base et les valeurs permises. Le fait de rédiger un document de ce type permet, entre autres, d'utiliser le minimum d'éléments et d'éviter les redondances. Voici la forme que prendra un dictionnaire de données:

IDENTIFICATEUR	SIGNIFICATION	TYPE	VALEUR

1. Les identificateurs.

Le nom d'un identificateur doit être significatif. Il doit indiquer clairement le rôle de la variable, ou de la constante, dans le programme.

Il y a trois catégories d'identificateurs :

1. Les constantes: Par convention, nous les écrirons en lettres majuscules, ce qui permettra de les distinguer des variables.
2. Les variables: Le nom devra être significatif, i.e. que le nom devra clairement identifier l'objet référencé. Par convention, nous débiterons chaque mot constituant de l'identificateur par une lettre majuscule.
3. Les tableaux: On devra toujours indiquer la taille du tableau. Un tableau est une variable composée d'un ensemble d'éléments de même type.

2. La signification.

Cette zone doit contenir une description de l'objet référencé par l'identificateur. Cette description indiquera le rôle et ce que représente l'identificateur.

3. Le type.

Le type indique le genre d'information (on pourrait dire d'objet) qu'un identificateur représente (un nombre, un caractère, etc.). Ce type devra être spécifié au moment de la définition (déclaration) de la variable.

Voici les types disponibles en langage C++, et leur signification:

Type	Grandeur	Valeur représentée
<i>char</i>	1 octet	un seul caractère ou un nombre entier (de -128 à 127)
<i>unsigned char</i>	1 octet	un caractère ou un nombre entier non signé (de 0 à 255)
<i>short</i>	2 octets	un nombre entier (de -32 768 à 32 767)
<i>unsigned short</i>	2 octets	un nombre entier non signé (de 0 à 65535)
<i>long</i> (ou <i>int</i>)	4 octets	un nombre entier (de -2 147 483 648 à 2 147 483 647)
<i>unsigned long</i> (ou <i>unsigned int</i>)	4 octets	un nombre entier non signé (de 0 à 4 294 967 295)
<i>float</i>	4 octets	un nombre réel (7 chiffres significatifs): $\pm 1.2\text{E}-38$ à $3.43\text{E}+38$
<i>double</i>	8 octets	un nombre réel en double précision (15 chiffres significatifs): $\pm 2.2\text{E}-308$ à $1.8\text{E}+308$
<i>long double</i>	10 octets	un nombre réel (19 chiffres significatifs): $\pm 3.4\text{E}-4932$ à $1.2\text{E}+4932$
<i>bool</i>	1 octet	valeur booléenne true ou false
<i>string</i>	variable	une chaîne de caractères (plusieurs caractères)
Type défini		tout type complexe défini par le programmeur

Le type **bool** est propre au langage C++ (il n'existe pas en C standard).

Le type **string** aussi est propre au langage C++ (il n'existe pas en C standard). C'est une classe d'objets qui permet de manipuler facilement un regroupement de caractères, par exemple le nom d'une personne, le numéro d'un cours ou encore une adresse. La définition de cette classe est faite dans le fichier d'en-tête *string*. Donc pour pouvoir utiliser cette classe nous devons utiliser la directive au préprocesseur `#include <string>`.

4. La valeur.

Pour chaque identificateur on indiquera sa valeur (absolue ou calculée).

1.4 LA STRUCTURE D'UN PROGRAMME EN LANGAGE C++

À première vue, un programme écrit en langage C apparaît obscur et complexe. Cette complexité n'est qu'apparente lorsqu'on maîtrise bien la syntaxe du langage. Dans le texte qui suit, nous allons examiner les différentes sections qui constituent un programme complet en C++. Les règles énumérées ici expliquent les standards que nous utiliserons dans nos exemples. Un programme en C++ est constitué de six sections:

1. l'en-tête
2. les directives au préprocesseur
3. les types définis par le programmeur
4. la déclaration des fonctions
5. la fonction main()
6. la définition des autres fonctions

1. L'en-tête

Un programme doit comporter des commentaires donnant des renseignements importants pour documenter et clarifier les différentes parties du programme. Les commentaires sont à l'usage exclusif du programmeur et sont donc ignorés par le compilateur. Un commentaire peut être inséré de deux façons différentes:

Si c'est un texte de plusieurs lignes: on l'encadre d'une paire de caractères:
/* au début, et */ à la fin du commentaire

Si c'est un texte d'une seule ligne: on le précède de //.

Au début du programme nous aurons toujours un commentaire spécial que nous appelons l'en-tête. Un en-tête doit, au minimum, contenir les informations suivantes:

- . le nom de l'auteur;
- . la date du programme;
- . le nom du fichier contenant le code du programme;
- . une brève description de l'objet du programme.

```

/*****
AUTEUR:
DATE:
NOM DU PROGRAMME:
DESCRIPTION:
*****/

```


2. Les directives au préprocesseur

Le langage C++ est composé de peu de choses. Il y a quelques mots-réservés et quelques opérateurs. Pour réaliser des tâches complexes, on doit donc écrire ses propres fonctions ou utiliser celles qui sont fournies avec le compilateur C++.

Les fonctions forment un élément fondamental de la programmation en C++. Les centaines de fonctions fournies sont regroupées par catégorie dans des bibliothèques. Notez que toutes les fonctions *standards* existent avec tous les compilateurs C++, c'est ce qui explique la grande *portabilité* de ce langage.

La directive **#include** permet d'indiquer au préprocesseur les fichiers d'en-tête qui seront utilisés par notre programme. Ces fichiers contiennent la description des fonctions dont nous aurons besoin. Ces dernières pourront ainsi être incorporées au programme lors de sa compilation.

exemple: `#include <iostream>`

On doit inscrire une directive **#include** pour chaque fichier d'en-tête à utiliser. Cette directive, comme tous les éléments du langage C++, doit être en lettres minuscules. Nous laisserons une ligne vide après cette section.

3. Les types définis par le programmeur

Une variable ou constante doit être définie avant d'être utilisée. De plus on lui associera un type lors de sa définition (voir le tableau de la section 1.3).

Lorsqu'aucun des types prédéfinis du langage ne nous convient, nous pouvons créer nos propres types. C'est le but de la présente section. Par la suite, des variables et constantes pourront être déclarées à l'aide de ces nouveaux types.

exemple :

```
struct Employe_s
{
    string NoEmploye;
    string Nom ;
    float TauxHoraire;
};
```

4. La déclaration des fonctions

Comme pour les types, nous pouvons définir nos propres fonctions. La section 6 servira à la description détaillée de nos fonctions (leur définition), mais pour permettre leur usage nous devons les déclarer ici. C'est ce qu'on appelle le *prototype* de la fonction.

Le prototype d'une fonction comprend:

- . le type de sa valeur de retour (son résultat);
- . son nom;
- . le type et le nom de ses paramètres (entre parenthèses).

exemples:

```
void TracerLigne(void);  
void          indique que cette fonction ne retourne aucun résultat  
TracerLigne   est le nom de la fonction  
(void)        indique que cette fonction n'a pas besoin d'aucune valeur pour faire son  
              travail
```

```
double CalculExposant (float Valeur1, float Valeur2);  
double          indique que cette fonction retourne un résultat de type  
                double  
CalculExposant  est le nom de la fonction  
(float Valeur1, float Valeur2) indique que cette fonction a besoin de deux valeurs  
                              réelles pour faire son travail
```

La déclaration d'une fonction se fait grâce à son **prototype**, et il y aura un prototype pour chaque fonction.

5. La fonction `main()`

Un programme écrit en C++ est divisé en plusieurs unités appelées fonctions. La fonction `main()` est la fonction principale, à laquelle le système d'exploitation passe le contrôle lors de l'exécution d'un programme. La présence de cette fonction est donc **obligatoire**.

À l'intérieur de chaque fonction on retrouvera des énoncés. La fin d'un énoncé est indiquée par un point-virgule. Plusieurs énoncés regroupés ensemble constituent un bloc. Les accolades `{ }` délimitent un bloc.

exemple:

```
void main(void)  
{  
    cout <<"Bonjour et bienvenue";  
    cout <<"au cours de C++";  
}
```

6. LA DÉFINITION DES AUTRES FONCTIONS

Chaque fonction déclarée à la section 4 devra être détaillée ici (c'est la définition). Le corps d'une fonction est composé des mêmes éléments que la fonction `main()`. Une fonction est donc un petit programme complet. Chaque fonction devrait être précédée d'un commentaire expliquant sa tâche et ses particularités.

1.5 LES DONNÉES

Un programme est l'aboutissement, la forme finale et exécutable des algorithmes créés pour résoudre un problème. Or chaque problème met en présence des informations diverses qui, pour nous, deviennent des données.

Dans un programme, toute donnée est représentée soit par une variable, soit par une constante. Dans l'ordinateur, toute donnée est emmagasinée dans une "case de sa mémoire". Donc pour chaque variable ou constante d'un programme, l'ordinateur réserve un espace-mémoire. Ce que nos programmes font, la plupart du temps, c'est de manipuler le contenu des espaces-mémoire de l'ordinateur.

Avec un peu de réflexion, nous pouvons identifier les opérations de base nécessaires pour la manipulation de ces espaces-mémoire. Il faudra, au minimum, pouvoir:

- différencier les espaces-mémoire les uns des autres;
- localiser un espace-mémoire en particulier;
- mettre une donnée dans un espace-mémoire;
- retrouver la valeur d'une donnée d'un espace-mémoire particulier;
- modifier la valeur d'une donnée existante d'un espace-mémoire.

Les concepteurs de compilateurs ont établi une règle assez simple pour régir l'utilisation des espaces-mémoire par nos programmes:

Pour être utilisable par un programme, un espace-mémoire devra posséder:

- . un **nom**, qui sera unique;
- . un **type**, qui permettra de déterminer sa taille (grandeur);
- . une **valeur** qui, dans le cas d'une variable, pourra être modifiée.

Nous suivons cette règle lorsque nous définissons nos constantes et nos variables. C'est au moment de leur définition que les espaces-mémoire sont réservés par le compilateur.

3. Structure des noms des variables et constantes

Un nom de variable ou de constante peut être constitué uniquement des caractères alpha-numériques (lettres et chiffres) et du caractère `_` (soulignement). Toutefois un identificateur ne pourra pas débiter par un chiffre. Quoique permis comme premier caractère, le caractère de soulignement (`_`) ne sera pas utilisé à cet endroit pour réduire la confusion possible avec les identificateurs spéciaux utilisés par les compilateurs.

Le langage C++ distingue les majuscules des minuscules, il est alors bien important d'en tenir compte. On considère donc qu'il y a 52 lettres (et non 26). En fait il y en a plus, car Visual Studio C++ accepte aussi les caractères accentués dans les noms de variables.

Par convention, nous écrivons les noms des variables en minuscules, sauf la première lettre de chaque mot.

exemples: `NbEtudiant`, `TotalFacture`, `NbColonneEcran`

Les mots-réservés du langage et du compilateur utilisé ne pourront pas être employés comme identificateur, sauf s'ils sont inclus dans un nom plus long.

exemples: **char** pas permis
 charade permis

3. Déclaration des variables

Les déclarations des variables en C, doivent se faire obligatoirement, au début des fonctions. En C++ les déclarations peuvent se faire à n'importe quel endroit dans le corps de la fonction.

Par convention, nous exigeons que toutes les déclarations soient faites au début des fonctions.

Par exemple:

```
void main (void)
{
    int Somme;
    short i, Compteur;
    float Moyenne, Note;
    double Distance, Salaire;
    char Lettre;
    ...
    ...
}
//Début de main()
//Déclarations des variables
//Corps de la fonction main()
//Fin de main()
```

Voici plusieurs exemples de déclaration de variables de type «simple»:

unsigned int Compteur;

réserve un espace-mémoire nommé Compteur d'une taille de 4 octets

short Somme;

réserve un espace-mémoire nommé Somme d'une taille de 2 octets

int NbEtudiant, NbCours, NbLab;

réserve un espace-mémoire nommé NbEtudiant d'une taille de 4 octets

réserve un espace-mémoire nommé NbCours d'une taille de 4 octets

réserve un espace-mémoire nommé NbLab d'une taille de 4 octets

unsigned long Population;

réserve un espace-mémoire nommé Population d'une taille de 4 octets

float Moyenne;

réserve un espace-mémoire nommé Moyenne d'une taille de 4 octets

double Distance;

réserve un espace-mémoire nommé Distance d'une taille de 8 octets

char Lettre;

réserve un espace-mémoire nommé Lettre d'une taille de 1 octet

Voici quelques exemples de déclaration de chaînes de caractères:

string Phrase, Nom, Prenom;

réserve l'espace mémoire nécessaire pour emmagasiner tous les caractères de la chaîne lors de l'affectation

N.B. Si nous déclarons des variables de type string, nous devons obligatoirement inscrire la directive au préprocesseur suivante:

```
#include <string>
```

Il est possible d'attribuer une valeur aux variables lors de leur déclaration mais, par souci de cohérence dans nos algorithmes, nous préférons que cela soit fait par un énoncé d'affectation distinct.

3. Les constantes

Une variable dont la valeur ne doit pas changer durant le traitement devrait plutôt être définie comme *constante*. Le nom d'une constante, comme celui d'une variable, doit la décrire clairement.

Une constante est définie comme une variable, mais le mot-clé **const** devra précéder le type de la constante.

```
void main (void)
{
    const int  NOMBRE_DE_LIGNES_ECRAN = 25;
    const int  NOMBRE_DE_COLONNES_ECRAN = 80;
    const char LETTRE_FIN = 'Z';
    .....
}
```

Une même valeur qui est utilisée plus d'une fois dans un programme doit être définie comme une constante.

La valeur d'une constante **doit** lui être affectée dès le moment de sa définition.

Exemples:

const int NOMBRE = 5;

réserve un espace-mémoire nommé Nombre d'une taille de 2 octets et dont le contenu pour la durée du programme est de 5

const double VITESSE = 132.6;

réserve un espace-mémoire nommé Vitesse d'une taille de 8 octets et dont le contenu pour la durée du programme est de 132.6

4. L'affectation des variables

Pour attribuer une valeur à un espace-mémoire nous devons utiliser l'opérateur d'affectation (=).

Par exemple:

```
#include <string>
using namespace std;
void main (void)
{
    int NbEtudiant;
    string NoCours;
    float Moyenne;

    NbEtudiant = 32;
    NoCours = "420-101-90";
    Moyenne = 67.8;
    ...
}
```

Il est aussi possible de modifier le contenu d'un espace-mémoire. Par exemple pour augmenter de 2 la valeur de la variable NbEtudiant nous ferons:

```
NbEtudiant = NbEtudiant + 2;
```

Cet énoncé aura pour effet d'extraire la valeur actuelle de la variable NbEtudiant (32) et de lui ajouter 2. La valeur finale de l'espace-mémoire NbEtudiant sera de 34.

5. Les caractères et les chaînes de caractères

Le langage C++ fait une distinction entre un et plusieurs caractères. Le terme *chaîne de caractères* est employé quand nous avons affaire à plusieurs caractères (il arrive aussi que nous traitions un seul caractère comme étant une chaîne).

Pour spécifier **un** seul caractère on le place entre apostrophes. Par exemple: 'a', 'A', '3', '\n' (\n est ce qu'on appelle un caractère de contrôle). Une variable ou constante qui doit contenir un caractère doit être déclarée de type **char**.

Pour spécifier une **chaîne** de caractères, constituée de 0 à plusieurs caractères, on la place entre guillemets. Par exemple: "Bonjour", "12345", "\0", "". Une variable qui doit contenir une chaîne doit être déclarée de classe **string**.

La classe **STRING**:

La classe **string** (accessible grâce à **#include <string>**) contient plusieurs fonctions et opérateurs nous permettant de faire des manipulations sur les chaînes. Voici les plus utiles:

Fonctions:

c_str()	transforme une <i>string</i> en chaîne de caractères standard (en langage C standard)
length() ou size()	donne la longueur de la chaîne
erase()	vider la chaîne (équivalent à affecter la chaîne vide "")

Opérateurs:

=	affectation
+, +=	addition
==, <, >, <=, >=, !=	opérateurs de relation

EXEMPLE

```
/******
AUTEUR: Marcel L'Italien
NOM DU PROGRAMME: string.cpp
DESCRIPTION: Tests sur des variables de type string
******/
#include <iostream>
#include <string>

using namespace std;

void main(void)
{
    string Texte1, Texte2, Texte3;
    string Nom1, Nom2;
    int Longueur;

    Texte1 = "Bonjour";
    Texte2 = "amis";
    Texte3 = Texte1 + ' ' + "les" + ' ' + Texte2;
    cout << Texte3;

    Longueur = Texte3.length();
    cout << "\n\nLongueur = " << Longueur;

    Texte3 = Texte3 + " du cours";
    cout << "\n" << Texte3;

    Nom1 = "Toto";
    Nom2 = "Tata";
    cout << "\n\nVoici les noms en ordre alphabétique:\n";
    if (Nom1 < Nom2)
        cout << Nom1 << "\n" << Nom2;
    else
        cout << Nom2 << "\n" << Nom1;
}
```

Exécution:

```
Bonjour les amis

Longueur = 16
Bonjour les amis du cours

Voici les noms en ordre alphabétique:
Tata
Toto
```


6. Normes sur les noms des variables et constantes

Outre les règles syntaxiques propres au langage, précisées au point 1, vous devrez faire en sorte que le nom d'une variable décrive complètement et précisément l'entité représentée par la variable. Cette règle sera la plus importante pour le choix de vos noms de variables. La description de l'entité représentée est souvent le meilleur nom possible.

Pourquoi:

- le nom sera facile à lire
- le nom ne pourra être confondu avec quelque chose d'autre
- le nom sera facile à mémoriser car il correspond à l'entité qu'il représente

Exemples:

Entité représentée:	Nombre total de chèques écrits à ce jour
Bons noms:	TotalCourant, TotalCheques, NbCheques
Mauvais noms:	Ecrits, TC, Cheques, TTCHQ, X, X1
Entité représentée:	Indice de fin de programme
Bons noms:	FinDeProgramme, FinProgramme, FinDeProg
Mauvais noms:	FDP, End, Fin, Indice, F, Arret

Un nom de variable doit exprimer un élément du problème et non pas un élément de la solution informatique. Il exprime le «**quoi**» plutôt que le «comment».

Exemples: Comment: ValeurLue	Quoi: Note
Comment: Drapeau	Quoi: ImprimantePrete
Comment: Compteur	Quoi: NbEtudiant

Idéalement, la longueur d'un nom de variable devrait être comprise entre 8 et 18 caractères. Un nom plus court peut être utilisé en autant que la portée de la variable soit très limitée. Une variable servant d'indice de répétition (dans un *for()* par exemple) pourrait s'appeler **i**, et ce serait acceptable.

Pour une meilleure lecture du programme, on pourra utiliser des abréviations, préfixes ou suffixes. Quand vous prendrez une telle décision, établissez une convention puis respectez-la durant tout le programme.

Exemples: des variables de totaux	l'abréviation Ttl sera employée elle sera mise à la fin du nom: VentesTtl, DepensesTtl, NotesTtl
des variables de moyenne	l'abréviation Moy sera employée elle sera mise à la fin du nom: NotesMoy, SalaireMoy, NoteGroupeMoy

des variables de type pointeur

le préfixe **p** sera employé, il sera mis au début du nom:

pNote, pAnnee, pJour

Il faut être prudent lorsqu'on utilise des abréviations. Voici quelques commentaires sur les abréviations tirés du livre Programmation professionnelle de Steve McConnell:

N'abrégez pas en supprimant un seul caractère

Abrégez de façon cohérente

Créez des noms qui soient prononçables

Évitez les combinaisons qui conduisent à des ambiguïtés de prononciation

Documentez les noms courts

Quand on utilise des opposés, il faut le faire avec le plus d'exactitude possible. On devrait toujours employer les conventions comprises par tous. N'essayez pas d'innover.

Exemples: *Correct:* vrai faux
 oui non

Incorrect: vrai non
 oui faux


1.6 LES OPÉRATEURS

Les opérateurs sont des mots ou des symboles qui indiquent des opérations spécifiques que les programmes devront exécuter. Les opérateurs sont divisés en différentes catégories dont les plus importantes sont:

- les opérateurs arithmétiques;
- les opérateurs d'affectation;
- les opérateurs d'incrément et de décrémentation;
- les opérateurs relationnels;
- les opérateurs booléens.

1. Les opérateurs arithmétiques

Les opérateurs arithmétiques sont:

- + l'addition
- la soustraction
- * la multiplication
- / la division
- % le modulo (pour des entiers seulement) 

L'opérateur du modulo permet de calculer le reste de la division de deux nombres entiers. Par exemple l'énoncé suivant:

`Reponse = 13 % 5;`

affectera la valeur 3 à la variable **Reponse**, parce que 3 est le reste de la division du nombre entier 13 par le nombre entier 5. Notez que le signe du résultat de l'opération modulo est toujours celui du numérateur.

EXEMPLE 1

```

/*****
AUTEUR:  Marcel L'Italien
FICHIER:  lop1.cpp
DESCRIPTION: Utilisation des opérateurs arithmétiques avec des variables
            de type entier.
*****/
#include <iostream>

using namespace std;

void main(void)
{
    int EntierA, EntierB, Resultat;

    EntierA=10;
    EntierB=3;

    Resultat = EntierA + EntierB;
    cout << "\n" << EntierA << " + " << EntierB << " = " << Resultat;

    Resultat = EntierA - EntierB;
    cout << "\n" << EntierA << " - " << EntierB << " = " << Resultat;

    Resultat = EntierA * EntierB;
    cout << "\n" << EntierA << " * " << EntierB << " = " << Resultat;

    Resultat = EntierA / EntierB;
    cout << "\n" << EntierA << " / " << EntierB << " = " << Resultat;

    Resultat = EntierA % EntierB;
    cout << "\n" << EntierA << " % " << EntierB << " = " << Resultat;
}

```

Exécution:

```

10 + 3 = 13
10 - 3 = 7
10 * 3 = 30
10 / 3 = 3
10 % 3 = 1

```

EXEMPLE 2

```

/*****
AUTEUR:  Marcel L'Italien
FICHIER: 1op2.cpp
DESCRIPTION: Utilisation des opérateurs arithmétiques avec des variables
           de type réel.
*****/
#include <iostream>

using namespace std;

void main(void)
{
    float ReelA, ReelB, Resultat;

    ReelA=10.1;
    ReelB=3.25;

    Resultat = ReelA + ReelB;
    cout << "\n" << ReelA << " + " << ReelB << " = " << Resultat;

    Resultat = ReelA - ReelB;
    cout << "\n" << ReelA << " - " << ReelB << " = " << Resultat;

    Resultat = ReelA * ReelB;
    cout << "\n" << ReelA << " * " << ReelB << " = " << Resultat;

    Resultat = ReelA / ReelB;
    cout << "\n" << ReelA << " / " << ReelB << " = " << Resultat;
}

```

Exécution:

```

10.1 + 3.25 = 13.35
10.1 - 3.25 = 6.85
10.1 * 3.25 = 32.825
10.1 / 3.25 = 3.10769

```

2. L'arithmétique avec des variables de différents types.

La précision d'un nombre, en langage C++, est fonction de son type. Nous devons donc porter un attention particulière à cet élément, surtout lorsque nous écrivons des expressions arithmétiques contenant des valeurs (variables) de types différents. Par exemple, qu'arrive-t-il lorsqu'on divise une variable de type **int** avec une variable de type **float**?

Le langage C++ utilise une certaine hiérarchie pour classer les nombres. Voici les types classés en ordre décroissant de priorité:

double	8 octets
float	4 octets
int ou long	4 octets
short	2 octets
char	1 octet

Vous pouvez remarquer que les réels ont priorité sur les entiers, et que les types les plus longs ont priorité sur les plus courts. Lors de l'exécution d'une opération arithmétique, il y aura conversion des opérandes dans le type de l'opérande ayant la plus haute priorité. Le résultat sera aussi de ce type. Par exemple, pour diviser 49 par 12.5 on convertira tout d'abord 49 en **double**, soit 49.0, puis on fera la division. Le résultat sera donc de type **double**.

Malgré ce type de conversion nous ne sommes pas à l'abri de résultats erronés. Si un résultat de type **float** était affecté à une variable de type **int**, par exemple, il y aurait perte des décimales. De même on ne devrait pas affecter un **double** à un **float**, ou un **int** à un **short**.

L'exemple qui suit, illustre ce type d'erreur.

```
int Vente, Unite;
float Prix;

Unite=50;
Prix=1.99;

Vente=Unite*Prix;
cout << "Le total des ventes est:" << Vente;
```

Dans ce programme, le calcul du prix de vente est correct ($\text{Unite} * \text{Prix}$), cependant le résultat étant affecté à la variable **Vente** (de type **int**) le résultat ne sera pas 99.50, comme il se doit, mais bien 99.

EXEMPLE 3

```
/******
AUTEUR:  Marcel L'Italien
FICHIER: lop3.cpp
DESCRIPTION: Utilisation des opérateurs arithmétiques avec des variables
             de types différents.
             Exemple de conversion explicite du type des données à l'aide
             du transtypage.
******/
#include <iostream>

using namespace std;

void main(void)
{
    int ValEnt;
    float ValReel, Resultat;

    ValEnt  = 15;
    ValReel = 5000.25;

    Resultat = ValReel + (float)ValEnt;

    cout << "Le résultat est de " << Resultat;
}
```

Exécution:

Le résultat est de 5015.25

EXEMPLE 4

```

/*****
AUTEUR:  Marcel L'Italien
FICHIER: lop4.cpp
DESCRIPTION: Exemples d'opérations arithmétiques avec des types variés.
            Attention ce programme génère 2 avertissements à l'exécution
*****/
#include <iostream>
using namespace std;
void main(void)
{
    short    Entier, RepEntier;
    long     EntLong, RepLong;
    float    Reel, RepReel;
    double    ReelDouble;
    Entier=50;
    EntLong=100000;
    Reel=10.549;
    ReelDouble=1000.005;

    /* SHORT + FLOAT à FLOAT */
    RepReel=Entier+Reel;
    cout <<"\n\nSHORT + FLOAT à FLOAT: " << Entier << " + " << Reel << " = "
        << RepReel;

    /* SHORT * FLOAT à FLOAT */
    RepReel=Entier*Reel;
    cout <<"\n\nSHORT * FLOAT à FLOAT: " << Entier << " * " << Reel<< " = "
        << RepReel;

    /* LONG + SHORT à LONG */
    RepLong=EntLong+Entier;
    cout <<"\n\nLONG + SHORT à LONG: " << EntLong << " + " << Entier<< " = "
        << RepLong;

    /* DOUBLE * FLOAT à FLOAT */
    RepReel=ReelDouble*Reel;
    cout <<"\n\nDOUBLE*FLOAT à FLOAT: " << ReelDouble << " * " << Reel << " = "
        << RepReel;

    /* SHORT / LONG à SHORT */
    RepEntier=Entier/EntLong; // WARNING
    cout << "\n\nSHORT / LONG à SHORT: " << Entier << " / " << EntLong << " = "
        << RepEntier;

    /* SHORT / LONG à FLOAT */
    RepReel=Entier/EntLong;
    cout << "\n\nSHORT / LONG à FLOAT: " << Entier << " / " << EntLong << " = "
        << RepReel;

    /* SHORT / LONG à FLOAT avec transtypage */
    RepReel=(float)Entier/EntLong;
    cout <<"\n\nSHORT / LONG à FLOAT (transtypage): " << Entier << " / "
        << EntLong << " = " << RepReel;
}

```


Exécution:

```
SHORT + FLOAT à FLOAT: 50 + 10.549 = 60.549
SHORT * FLOAT à FLOAT: 50 * 10.549 = 527.45
LONG + SHORT à LONG: 100000 + 50 = 100050
DOUBLE * FLOAT à FLOAT: 1000 * 10.549 = 10549.1
SHORT / LONG à SHORT: 50 / 100000 = 0
SHORT / LONG à FLOAT: 50 / 100000 = 0
SHORT / LONG à FLOAT (transtypage): 50 / 100000 = 0.0005
```

3. Les opérateurs d'affectation

Vous êtes déjà familiers avec l'opérateur d'affectation =, nous passerons donc immédiatement aux opérateurs d'affectation raccourcis.

Un des aspects intéressants du langage C++ est qu'il offre des opérateurs qui regroupent plusieurs opérations (ces opérateurs sont souvent appelés opérateurs d'affectation arithmétiques ou opérateurs raccourcis). Par exemple:

Accumul = Accumul + 10; *augmenter de 10 la valeur de Accumul*

peut s'écrire aussi de la façon suivante:

Accumul += 10;

donnant ainsi un énoncé plus compact, et surtout plus rapide à exécuter.

Les opérateurs d'affectation sont les suivants:

=	affectation simple	+=	addition
-=	soustraction	*=	multiplication
/=	division	%=	modulo (pour entier seulement)

4. Les opérateurs d'incrémentation et de décrémentation

Le langage C++ utilise un autre type d'opérateurs qui n'est pas commun aux autres langages. Ce type est composé d'opérateurs unaires qui permettent d'écrire des énoncés de type compteur de façon très abrégée.

Au lieu d'écrire **Compteur = Compteur + 1;**
nous écrivons **Compteur++;** ou **++Compteur;**

Ces opérateurs sont: **++** pour l'addition (incrémentation)
 -- pour la soustraction (décrémentation)

Lorsque l'opérateur précède le nom de la variable il est exécuté avant toutes les autres opérations écrites sur la même ligne (dans le même énoncé), sinon son exécution se fait après toutes les autres opérations.

EXEMPLE 5

```
/******
AUTEUR:  Marcel L'Italien
FICHIER:  lop5.cpp
DESCRIPTION: Exemples d'utilisation des opérateurs arithmétiques raccourcis
             (affectation arithmétique: +=, -=, /=, *= et %=)
*****/
#include <iostream>
using namespace std;

void main(void)
{
    int EntierM, EntierN;
    EntierM=10;
    EntierN=5;

    cout    <<"\nLes valeurs de départ sont:\n\tEntierM = " << EntierM
           <<" et EntierN = " << EntierN << "\n";
    cout    <<"\n    EntierM += 2\t --> EntierM = " << (EntierM += 2);
    cout    <<"\n    EntierM -= EntierN\t --> EntierM = "
           << (EntierM -= EntierN);
    cout    <<"\n    EntierM *= EntierN\t --> EntierM = "
           << (EntierM *= EntierN);
    cout    <<"\n    EntierM /= 2\t --> EntierM = " << (EntierM /= 2);
    cout    <<"\n    EntierM %= 2\t --> EntierM = " << (EntierM %= 2);
}
```

Les valeurs de départ sont:

EntierM = 10 et EntierN = 5

EntierM += 2	--> EntierM = 12
EntierM -= EntierN	--> EntierM = 7
EntierM *= EntierN	--> EntierM = 35
EntierM /= 2	--> EntierM = 17
EntierM %= 2	--> EntierM = 1

EXEMPLE 6

```
/******
AUTEUR:  Marcel L'Italien
FICHIER: lop6.cpp
DESCRIPTION: Exemple d'utilisation des opérateurs d'incrémentation et
             de décrémentation
******/
#include <iostream>

using namespace std;

void main(void)
{
    int Entier;

    Entier=100;

    cout << "\nLa valeur de Entier est: " << Entier;
    cout << "\nLa valeur de Entier++ est: " << Entier++;
    cout << "\nLa valeur de Entier est: " << Entier;
    cout << "\nLa valeur de ++Entier est: " << ++Entier;
    cout << "\nLa valeur de --Entier est: " << --Entier;
    cout << "\nLa valeur de Entier-- est: " << Entier--;
    cout << "\nLa valeur de Entier est: " << Entier;
}
```

```
La valeur de Entier est: 100
La valeur de Entier++ est: 100
La valeur de Entier est: 101
La valeur de ++Entier est: 102
La valeur de --Entier est: 101
La valeur de Entier-- est: 101
La valeur de Entier est: 100
```

5. Les opérateurs relationnels et de comparaison

Ces opérateurs sont utilisés principalement dans les énoncés de répétition et dans les énoncés conditionnels (énoncés de sélection ou de branchement). On les utilise dans toutes les *conditions*.

Les opérateurs relationnels et de comparaison sont les suivants:

<	plus petit que
>	plus grand que
<=	plus petit ou égal à
>=	plus grand ou égal à
==	égal à
!=	différent de

Une expression contenant un tel opérateur sera évaluée et retournera la valeur **1** si l'expression est *vrai*, ou **0** si l'expression est *fausse*. Notez qu'en langage C `faux == 0` et `vrai != 0` (le type *booléen* n'existe pas en C et est remplacé par *int*). Nous utilisons souvent les deux énoncés suivants dans les programmes en C:

```
const int VRAI=1;
const int FAUX=0;
```

Cependant, il est conseillé d'utiliser le type **bool** ainsi que les constantes **true** et **false** en langage C++.

6. Les opérateurs booléens

Les opérateurs booléens (logiques) sont utilisés lorsqu'on veut vérifier plus d'une relation dans la même expression conditionnelle. Les conditions multiples doivent être liées par un opérateur logique.

Les opérateurs logiques sont les suivants:

&&	et
 	ou
!	négation



1.7 EXERCICES

1. Quels sont les trois principaux types de variables?
2. Dans le nom d'une variable, majuscules et minuscules sont-elles équivalentes?
3. Citer, parmi les identificateurs suivants, ceux qui sont valides. Pour ceux qui ne le sont pas, donner la raison.

a) Nom1	e) Nom	i) Nom-Et-Prenom
b) 1nom	f) Prénom	j) 123
c) Variable_2	g) Nom et prenom	k) 123-45-67
d) Float	h) NomEtPrenom	
4. Trouver les caractères valides parmi les valeurs suivantes.

a) 'a'	e) 'T'	i) "123"
b) '\$'	f) "	j) ''
c) '\n'	g) 'x yz'	k) '9'
d) '\t'	h) "x"	l) "9"
5. Trouver les chaînes de caractères valides parmi les valeurs suivantes.
 - a) '8:15 PM'
 - b) "Rouge, Blanc, Bleu"
 - c) "Nom:
 - d) "Chapitre 3 (suite)"
 - e) "1.6e-78"
 - f) "Montréal, H4N 8K9"
 - g) "Le professeur a dit "Le C C- facile"
6. Indiquez les énoncés dont la syntaxe est valide.
 - a) int Valeur;
 - b) int ValeurA, ValeurB;
 - c) float ValeurC;
 - d) double float ValeurD;
 - e) unsigned char ValeurE;
7. Écrire les définitions (en langage C++) adéquates pour les variables de la liste suivante:
 - a) variables entières: ValP, ValQ
variables réelles: ValX, ValY, ValZ
variables caractères: ValA, ValB, ValC
 - b) variable entière: Compteur
variables réelles: Cout, Taxe
variables chaîne de caractères: Nom, Prenom

- c) variables réelles en double précision: Impot, Net, Brut
variable entière non-signée: NoClient
variable entière: Indice
- d) variable réelle: Taux
variables entières longues: Avant, Apres
variables caractères non-signées: Lettre, Code

8. Écrire les définitions et les énoncés d'affectation adéquats pour les variables et constantes de la liste suivante:

- a) variables réelles: ValA=-8.2, ValB=0.005
constante réelle: REEL=8.2
variables entières: ValX=129, ValY=32677, ValZ=-22
constante entière: ENTIER=123
variables caractères: ValC=w, ValD=&, ValE=Q
- b) variables en double précision: Val1=2.88x10⁻⁸, Val2=-8.03x10⁵
constante réelle: VAL=32.45
constante double: DOUBLE=3468.678
constante réelle: PI=3.1416
variables entières non-signées: Code1=126, Code2=A
- c) variable chaîne de caractères: Nom=Arthur Laframboise
variable caractère: Lettre=C
constante caractère: DEBUT=a
constante chaîne de caractères: TEXTE=Bonjour les amis
- d) variable réelle: Taux=2.12
variables entières longues: Avant=2x10⁵, Apres=700000
variables caractères non-signées: Lettre=250, Code=é
- e) constante caractère: LETTRE = a
constante entière non-signé: VALEUR = 255
constante entière longue non-signée: VALEUX= 1 234 567
constante réelle: PI = 3.14159

9. Réécrire les expressions suivantes en employant le bon opérateur (la forme la plus courte):

- a) Nombre=Nombre+1;
- b) Somme=Somme+5;
- c) Taux=Taux*10;
- d) Temp=Temp-1;
- e) Mode=Mode+Nombre;

10. Un opérateur relationnel est utilisé pour

- a) concaténer des valeurs
- b) comparer des valeurs
- c) distinguer différents types de variable
- d) changer la valeur d'une variable en une valeur logique

11. Soient EntA, EntB et EntC des variables entières auxquelles ont été affectées les valeurs 10, 2 et -4. Déterminer la valeur des expressions arithmétiques suivantes:

- | | | |
|-------------------------|--------------|---------------------|
| a) EntA+EntB+EntC | d) EntA%EntB | g) EntA*EntB/EntC |
| b) 2*EntB+3*(EntA-EntC) | e) EntA/EntC | h) EntA*(EntB/EntC) |
| c) EntA/ EntB | f) EntA%EntC | i) (EntA*EntC)%EntB |
| | | j) EntA*(EntC%EntB) |

12. À partir des définitions suivantes, déterminer le type du résultat de chacune des expressions:

```
int    EntI,EntJ;
long   EntLong;
float   Reel;
double ReelDbl;
char    Carac;
```

- | | | |
|-----------------|---------------------------|------------------|
| a) EntI+Reel | d) (int(ReelDbl))+EntLong | g) EntLong+EntJ |
| b) Reel+Carac | e) EntI+Reel | h) EntI+Carac |
| c) ReelDbl+Reel | f) EntLong+EntJ | i) EntLong+Carac |

13. Déterminer la valeur de chacune des expressions suivantes:

```
int i,j;
float x,y;
char c,d;
i=8;
j=5;
x=0.005;
y=-0.01;
c='c';
d='d';
```

- | | | |
|-----------|----------------|----------------------------|
| a) -(i+j) | g) c>d | m) 2*x+y==0 |
| b) ++i | h) x>=0 | n) !(i<j) |
| c) i++ | i) j!=6 | o) (i>0) && (j<5) |
| d) --j | j) c==99 | p) (i>0) (j<5) |
| e) y-- | k) 5*(i+j)>'c' | q) (x>y) && (i>0) (j<5) |
| f) i<=j | l) (2*x+y)==0 | r) (x>y) && (i>0) && (j<5) |

14. À partir des définitions et initialisations qui suivent, déterminer la valeur de chacune des expressions suivantes:

```
int i,j,k;  
float x,y,z;  
char a,b,c,d;
```

```
i=8;  
j=5;  
x=0.005;  
y=-0.01;  
c='c';  
d='d';
```

- a) $k = (i + j)$
- b) $z = (x + y)$
- c) $k = (x + y)$
- d) $k = c$
- e) $z = i / j$
- f) $i += 2$
- g) $y -= x$
- h) $x *= 2$
- i) $i /= j$
- j) $i \% = j$
- k) $i += (j - 2)$

CHAPITRE 2

2.1 LES OPÉRATIONS DE SORTIE

Ce qui intéresse vraiment l'utilisateur d'un programme, c'est d'obtenir des résultats. Les résultats informatiques prennent plusieurs formes. Ils peuvent être des affichages à l'écran, des rapports imprimés, des chèques de paie ou encore des factures. Dans d'autres domaines non-administratifs les résultats peuvent être des plans, des films ou encore le mouvement d'une machine-outil.

Dans ce chapitre nous ne verrons que l'affichage de résultats à l'écran.

Le flux qui permet d'afficher des résultats est *cout* (unité de sortie standard, i.e. l'écran). L'opérateur qui permet de transmettre les résultats au flux est `<<`.

1. LA COMMANDE DE SORTIE DANS LES DIAGRAMMES D'ACTION

Quand nous faisons l'analyse d'un problème à l'aide des diagrammes d'action et que nous désirons transmettre un ou des résultats à notre usager, nous n'utilisons qu'un seul terme, le mot **ÉCRIRE** suivi de la liste des résultats à transmettre.

exemple: On désire afficher à l'écran la date (Jour, Mois, Année):

ÉCRIRE Jour, Mois, Année

On désire imprimer un chèque de paie (Date, Bénéficiaire, Montant):

ÉCRIRE Jour, Mois, Année, Bénéficiaire, Montant

On désire afficher à l'écran le nombre d'étudiants dans la classe:

ÉCRIRE NbEtudiant

Voici un exemple de diagramme d'action permettant de diviser 144 par 8. Notez la dernière ligne du diagramme.

```
┌ Nombre=144
├ Diviseur=8
├ Quotient=Nombre/Diviseur
└ ÉCRIRE Nombre, Diviseur, Quotient
```

2. LE CONTRÔLE DE LA MISE EN PAGE

Lorsque nous écrivons nos diagrammes d'action nous ne nous préoccupons aucunement de la mise en page. C'est au moment du codage que nous devons ajouter les énoncés qui permettront de «**formater**» nos sorties.

Pour la mise en page à l'intérieur d'un écran, le langage C offre des séquences d'échappement que nous pourrions insérer dans toute chaîne à afficher. Une séquence d'échappement est composée d'un ou plusieurs caractères et est précédée d'une barre oblique inversée (\). La barre oblique inversée est considérée comme le caractère d'échappement, car elle permet d'échapper à l'interprétation habituelle des caractères contenus dans une chaîne. Le caractère suivant ce symbole aura donc une signification particulière. Les séquences d'échappement que nous utiliserons sont les suivantes:

Séquences d'échappement:

<code>\n</code>	passer à la ligne suivante;
<code>\t</code>	aller à la prochaine tabulation;
<code>\"</code>	un guillemet;
<code>\\</code>	une barre oblique inversée;
<code>\r</code>	retour au début de la ligne (touche <enter>)
<code>\b</code>	retour arrière (une colonne vers la gauche)
<code>\a</code>	faire un <i>bip</i> !
<code>\xhhh</code>	le caractère ASCII représenté par le nombre hhh (hexadécimal).
<code>\ooo</code>	le caractère ASCII représenté par le nombre ooo (octal).
<code>\0</code>	le caractère nul

Nous verrons plusieurs exemples de ces séquences dans les programmes présentant le flux de sortie *cout*. Aussi, on peut utiliser **endl** à la place de la séquence `\n` dans le flux *cout*.

3. LE FLUX COUT

Le flux *cout* permet d'afficher (à l'écran) des valeurs de n'importe quel type du langage C++, soient des chaînes de caractères (string), des caractères (char) et des valeurs numériques. Dans une chaîne nous pouvons mettre:

- . du texte;
- . des séquences d'échappement.

Notez qu'on peut afficher aussi bien des constantes que des variables. Il est aussi possible d'afficher plusieurs valeurs avec une seule utilisation du flux *cout*, toutefois chaque valeur sera précédée de l'opérateur << (écrire).

Exemples avec des chaînes de caractères:

Descendre le curseur de deux lignes puis afficher le texte «Bonjour»:

```
cout << "\n\nBonjour";
```

Afficher les mots «Nom», «Prénom», et «Note» à partir de la position courante du curseur. Espacer chaque titre de deux tabulations. Déplacer le curseur à la ligne suivante après l'affichage:

```
cout << "Nom\t\tPrénom\t\tNote\n";
```

Afficher, à partir de la position courante du curseur, les mots Nord, Sud , Est et Ouest. Chaque mot devra apparaître sur une ligne différente:

```
cout << "Nord" << endl << "Sud" << endl << "Est" << endl << "Ouest";
```

Exemples avec des variables:

Afficher le résultat d'une division sous la forme $a/b=c$. Les variables utilisées sont Nombre, Diviseur et Quotient (toutes sont de type float).

```
cout << Nombre << '/' << Diviseur << '=' << Quotient;
```

Afficher le nombre d'étudiant dans un groupe sous la forme suivante:


Il y a xx étudiants dans le groupe $xxxx$

```
cout << "Il y a " << NbEtudiant << " dans le groupe " << NoGroupe;
```

4. LES MANIPULATEURS

Pour permettre une plus grande souplesse lors de la mise en page, nous utiliserons les *manipulateurs* du flux de sortie *cout*. Pour les utiliser nous devons inclure le fichier d'en-tête *iomanip* (avec la directive **#include <iomanip>**).

Les manipulateurs:

setw(<i>n</i>)		pour spécifier une largeur minimale de <i>n</i> caractères. On complète avec des espaces.
oct :		écriture en octal (pour les entiers)
hex :		écriture en hexadécimal (pour les entiers)
dec :		écriture en décimal (par <i>défaut</i>)
setprecision(<i>n</i>) :		pour spécifier une précision de <i>n</i> décimales (pour les réels) pour spécifier le nombre de chiffres à afficher (pour les entiers)
fixed :		pour activer la notation en virgule flottante (pour les réels seulement)
scientific :		pour activer la notation scientifique (pour les réels seulement)
left :		pour activer le cadrage à gauche
right :		pour activer le cadrage à droite (par défaut)
endl :		changer de ligne (équivalent à '\n')

N.B. À l'exception de *setw()*, les manipulateurs modifient les caractéristiques de toutes les prochaines valeurs affichées. Les réglages durent donc jusqu'à un nouveau réglage fait par ces mêmes manipulateurs.

EXEMPLE 1

```

/*****
AUTEUR:  Marcel L'Italien
FICHIER: 2cout1.cpp
DESCRIPTION: Utilisation des manipulateurs setw() et left avec
             une variable entière.
*****/
#include <iostream>  // pour utiliser cout
#include <iomanip>    // pour utiliser les manipulateurs

using namespace std;

void main(void)
{
    int Val;

    Val=12345;

    cout << "\nAfficher la valeur dans 3 positions:" << setw(3) << Val;
    cout << "\nAfficher la valeur dans 5 positions:" << setw(5) << Val;
    cout << "\nAfficher la valeur dans 6 positions:" << setw(6) << Val;
    cout << "\nAfficher la valeur dans 8 positions:" << setw(8) << Val;

    cout << "\n\nMêmes affichages que ci-dessus, mais avec un "
         << "alignement à gauche\n";

    cout << left; // Activer la cadrage à gauche
    cout << "\nAfficher la valeur dans 3 positions:" << setw(3) << Val;
    cout << "\nAfficher la valeur dans 5 positions:" << setw(5) << Val;
    cout << "\nAfficher la valeur dans 6 positions:" << setw(6) << Val;
    cout << "\nAfficher la valeur dans 8 positions:" << setw(8) << Val;

}

```

ÉCRAN DE SORTIE

```

Afficher la valeur dans 3 positions:12345
Afficher la valeur dans 5 positions:12345
Afficher la valeur dans 6 positions: 12345
Afficher la valeur dans 8 positions:  12345

Mêmes affichages que ci-dessus, mais avec un alignement à gauche

Afficher la valeur dans 3 positions:12345
Afficher la valeur dans 5 positions:12345
Afficher la valeur dans 6 positions:12345
Afficher la valeur dans 8 positions:12345

```

Dans l'exemple précédent on remarque que le corps du programme est précédé par les directives au préprocesseur **#include <iostream>** et **#include <iomanip>**. Un groupe de fichiers est emmagasiné dans le répertoire **include** du compilateur. Dans ces fichiers de texte sont conservés les prototypes de toutes les fonctions offertes par le compilateur C++ (fonctions qui sont donc disponibles à vos programmes), ainsi que certaines constantes prédéfinies et certains types particuliers (*string* par exemple). Ces informations sont regroupées par catégories. Les définitions nécessaires au flux de sortie *cout* sont regroupées dans le fichier **iostream**. De même, les manipulateurs peuvent être utilisés grâce au fichier **iomanip**.

Vous avez peut-être noté la présence de l'instruction **using namespace std;** devant le programme. En fait le flux *cout* porte le nom complet *std::cout*. Cette instruction indique que nous utiliserons l'espace de noms **std** par défaut, nous évitant l'écriture au long.

Pour utiliser une fonction, un flux ou la classe *string* :

À chaque fois que vous utiliserez une fonction, un flux ou la classe *string*, vous devrez vous assurer que le fichier contenant sa déclaration est inclus devant le corps du programme, et ceci à l'aide de la directive **#include** du préprocesseur.

Puisque chacun de nos programmes affiche des résultats grâce au flux *cout*, nous placerons l'énoncé **using namespace std;** devant chaque programme.

EXEMPLE 2

```
/******
AUTEUR:  Marcel L'Italien
FICHIER: 2cout2.cpp
DESCRIPTION: Utilisation des manipulateurs setw() et setprecision() avec une
            variable réelle.
******/
#include <iostream>
#include <iomanip>

using namespace std;

void main(void)
{
    float Valf;
    Valf=123.4560;

    cout << "\nAffichage du nombre réel:" << Valf;
    cout << fixed; //Activer la notation virgule flottante

    cout << "\n\nAfficher un nombre réel avec une largeur et une précision:";
    cout << "\nLargeur de 12 et précision de 6 -->" << setw(12) << setprecision(6) << Valf;
    cout << "\nLargeur de 8 et précision de 4 -->" << setw(8) << setprecision(4) << Valf;
    cout << "\nLargeur de 8 et précision de 3 -->" << setw(8) << setprecision(3) << Valf;
    cout << "\nLargeur de 8 et précision de 2 -->" << setw(8) << setprecision(2) << Valf;

    cout << "\n\nMêmes affichages que ci-dessus, mais avec cadrage à gauche:";
    cout << left; //Activer le cadrage à gauche
    cout << "\nLargeur de 12 et précision de 6 -->" << setw(12) << setprecision(6) << Valf;
    cout << "\nLargeur de 8 et précision de 4 -->" << setw(8) << setprecision(4) << Valf;
    cout << "\nLargeur de 8 et précision de 3 -->" << setw(8) << setprecision(3) << Valf;
    cout << "\nLargeur de 8 et précision de 2 -->" << setw(8) << setprecision(2) << Valf;
}
```

ÉCRAN DE SORTIE

```
Affichage du nombre réel:123.456

Afficher un nombre réel avec une largeur et une précision:
Largeur de 12 et précision de 6 --> 123.456001
Largeur de 8 et précision de 4 -->123.4560
Largeur de 8 et précision de 3 --> 123.456
Largeur de 8 et précision de 2 --> 123.46

Mêmes affichages que ci-dessus, mais avec cadrage à gauche:
Largeur de 12 et précision de 6 -->123.456001
Largeur de 8 et précision de 4 -->123.4560
Largeur de 8 et précision de 3 -->123.456
Largeur de 8 et précision de 2 -->123.46
```

EXEMPLE 3

```
/******  
AUTEUR:  Marcel L'Italien  
FICHIER: 2cout3.cpp  
DESCRIPTION: Affichage d'un caractère et de son code ASCII  
*****/  
#include <iostream>  
  
using namespace std;  
  
void main(void)  
{  
    char Ch1,  
        Ch2;  
  
    Ch1='A';  
    Ch2='a';  
  
    cout << "\n Le caractère '" << Ch1 << "' a le code ASCII " << (int)Ch1;  
    cout << "\n Le code ASCII du caractère '" << Ch2 << "' est " << (int)Ch2;  
}
```

ÉCRAN DE SORTIE

```
Le caractère 'A' a le code ASCII 65  
Le code ASCII du caractère 'a' est 97
```


EXEMPLE 4

```
/******
AUTEUR: Marcel L'Italien
FICHIER: 2cout4.cpp
DESCRIPTION: Affichage de nombres entiers en décimal, en octal
             et en hexadécimal.
******/
#include <iostream>
#include <iomanip>

using namespace std;

void main(void)
{
    int Nb1,
        Nb2,
        Nb3,
        Nb4;

    Nb1=1;
    Nb2=9;
    Nb3=12;
    Nb4=17;

    cout << "\n\nImpression des 4 nombres en décimal:\t\n";
    cout << dec << Nb1 << '\t' << Nb2 << '\t' << Nb3 << '\t' << Nb4;

    cout << "\n\nImpression des 4 nombres en octal:\t\n";
    cout << oct << Nb1 << '\t' << Nb2 << '\t' << Nb3 << '\t' << Nb4;

    cout << "\n\nImpression des 4 nombres en hexadécimal:\t\n";
    cout << hex << Nb1 << '\t' << Nb2 << '\t' << Nb3 << '\t' << Nb4;

}
```

Écran de sortie

```
Impression des 4 nombres en décimal:
1      9      12      17

Impression des 4 nombres en octal:
1      11      14      21

Impression des 4 nombres en hexadécimal:
1      9      c      11
```

EXEMPLE 5

```

/*****
AUTEUR:  Marcel L'Italien
FICHIER: 2cout5.cpp
DESCRIPTION: Affichage de diverses variables, de leur taille ( sizeof() )
              et de leur adresse ( & ).
*****/

#include <iostream>

using namespace std;

void main(void)
{
    char    Carac;
    int     Entier;
    long    EntLong;
    float   Reel;
    double  ReelLong;

    cout << "\nLa taille de char est de: " << sizeof(char) << " octet";
    cout << "\nLa taille de int est de: " << sizeof(int) << " octets";
    cout << "\nLa taille de long est de: " << sizeof(long) << " octets";
    cout << "\nLa taille de float est de: " << sizeof(float) << " octets";
    cout << "\nLa taille de double est de: " << sizeof(double) << " octets";

    cout << "\n\n\nAdresse des variables (en hexadécimal):\n";
    cout << "\nL'adresse de Entier est: " << &Entier;
    cout << "\nL'adresse de EntLong est: " << &EntLong;
    cout << "\nL'adresse de Reel est: " << &Reel;
    cout << "\nL'adresse de ReelLong est: " << &ReelLong;
    cout << "\nL'adresse de Carac est: " << (int *)&Carac;
                                     /* Pour ne pas considérer &Carac
                                     comme une chaîne C standard */

    cout << "\n\n\nAdresse des variables (en décimal):\n";
    cout << "\nL'adresse de Entier est: " << (int)&Entier;
    cout << "\nL'adresse de EntLong est: " << (int)&EntLong;
    cout << "\nL'adresse de Reel est: " << (int)&Reel;
    cout << "\nL'adresse de ReelLong est: " << (int)&ReelLong;
    cout << "\nL'adresse de Carac est: " << (int)&Carac;
}

```

ÉCRAN DE SORTIE

```
La taille de char est de: 1 octet  
La taille de int est de: 4 octets  
La taille de long est de: 4 octets  
La taille de float est de: 4 octets  
La taille de double est de: 8 octets
```

Adresse des variables (en hexadécimal):

```
L'adresse de Entier est: 0x0c9f2814  
L'adresse de EntLong est: 0x0c9f2810  
L'adresse de Reel est: 0x0c9f280c  
L'adresse de ReelLong est: 0x0c9f2804  
L'adresse de Carac est: 0x0c9f2817
```

Adresse des variables (en décimal):

```
L'adresse de Entier est: 10260  
L'adresse de EntLong est: 10256  
L'adresse de Reel est: 10252  
L'adresse de ReelLong est: 10244  
L'adresse de Carac est: 10263
```

2.2 LES OPÉRATIONS D'ENTRÉE

Nous avons vu que nos programmes peuvent transmettre des résultats et des informations à l'utilisateur, via les flux de sortie. Mais qu'arrive-t-il lorsque l'utilisateur doit transmettre des données au programme.

Il est très important d'être conscient que l'utilisateur ne peut décider du moment où il doit transmettre ses données à nos programmes. Pour un programme, par exemple, calculant le prix d'une facture, il faudra que l'utilisateur fournisse le nombre d'articles achetés ainsi que le prix unitaire de chacun. Mais c'est seulement *après* la réception de ces données par le programme qu'on pourra faire les calculs et transmettre les résultats à l'utilisateur. Le programmeur décidera donc, au moment où il créera le diagramme d'action, du moment précis où l'utilisateur devra entrer les données.

Pour ce faire le programmeur utilisera la *lecture* de données à partir d'un média quelconque. Ça peut être le clavier, un fichier, une bande magnétique, un lecteur optique ou encore une caméra. Dans ce chapitre nous ferons la saisie des données via le clavier.

1. LA COMMANDE D'ENTRÉE DANS LES DIAGRAMMES D'ACTION

Lorsqu'on voudra capter des données externes au programme, on utilisera toujours le terme **LIRE**, suivi de la liste des variables qui serviront à mémoriser ces données.

Par exemple: On désire demander à l'utilisateur le prix et la quantité d'un article qu'il a acheté:
 LIRE Quantité, Prix

 On désire demander à l'utilisateur le numéro du mois présent:
 LIRE Mois

Voici un exemple de diagramme qui permet de lire deux nombres puis de les diviser:

```
┌ LIRE Nombre, Diviseur  
├ Quotient = Nombre/Diviseur  
└ ÉCRIRE Nombre, Diviseur, Quotient
```

AVIS TECHNIQUE:

Dans le cadre d'une bonne programmation, l'entrée des données devra être précédée par l'affichage d'un message expliquant ce que nous attendons de l'utilisateur.

2. LE FLUX D'ENTRÉE CIN

La lecture à partir du flux *cin* est dite «bufferisée» car elle se sert d'un **tampon** où les données sont stockées avant d'être affectées aux variables. Les données ne seront affectées aux variables qu'après avoir pressé la touche <ENTER>.

Le flux *cin* permet de lire des données de **tous** les types prédéfinis (int, float, char, string, etc.). C'est l'opérateur >> (opérateur *lire*) qui sera utilisé pour transmettre les données vers les variables.

La lecture à partir du flux *cin* se fait toujours:

- en supprimant les espaces initiaux,
- avec l'espace (un ou plusieurs) comme séparateur entre les valeurs à lire
- en pressant la touche <ENTER> pour terminer.

Donc pour lire plusieurs valeurs il faudra les séparer d'au moins un espace.

Tous les caractères suivants sont considérés comme un espace:

espace
\t (tabulation)
\n (enter)

Par exemple:

```
int Valeur1, Valeur2;
```

ce sont les deux valeurs à lire

```
cin >> Valeur1 >> Valeur2;
```

l'utilisateur devra taper les deux valeurs entières en les séparant par un ou plusieurs espaces.

Exemple: **4 6<ENTER>**

```
int Jour, Mois, Annee;
```

les trois valeurs à lire

```
cin >> Jour >> Mois >> Annee;
```

l'utilisateur tape les trois valeurs séparées par un ou plusieurs espaces

Exemple: **28 8 97<ENTER>**

```
string Nom;  
int Age;
```

les deux valeurs à lire

```
cin >> Nom >> Age;
```

Exemple: **Roger 25<ENTER>**

EXEMPLE 1

LE PROBLÈME

Écrire un programme qui permet de lire l'heure de départ et l'heure d'arrivée (heure et minutes) d'un train. Ensuite on calculera et affichera la différence entre les deux temps (en minutes).

QUE VEUT-ON À LA SORTIE?

La différence entre les deux temps, en minutes.

QUELLES SONT LES DONNÉES CONNUES?

L'heure et les minutes de début;
L'heure et les minutes de fin.

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
Heure	l'heure de départ ou d'arrivée	int	Lue à l'entrée
Minutes	les minutes de départ ou d'arrivée	int	Lue à l'entrée
TempsDebut	l'heure de départ en minutes	int	Heure*60 +Minutes
TempsFin	l'heure d'arrivée en minutes	int	Heure*60 +Minutes

LE DIAGRAMME D'ACTION:

```
— Lire Heure  
— Lire Minutes  
— TempsDebut = Heures*60 + Minutes  
— Lire Heure  
— Lire Minutes  
— TempsFin = Heure*60 + Minutes  
— Écrire TempsFin-TempsDebut
```

LE PROGRAMME

```

/*****
AUTEUR:  Marcel L'Italien
FICHIER: 2cin1.cpp
DESCRIPTION: Calcul de la différence entre deux temps
*****/
#include <iostream>

using namespace std;

void main(void)
{
    int TempsDebut,
        TempsFin,
        Heure,
        Minutes;

    cout << "\nEntrez l'heure de départ:\n\tHeure: ";
    cin >> Heure;
    cout << "\tMinutes: ";
    cin >> Minutes;

    TempsDebut=(Heure*60)+Minutes;

    cout << "\nEntrez l'heure d'arrivée (doit être plus tard):\n\tHeure: ";
    cin >> Heure;
    cout << "\tMinutes: ";
    cin >> Minutes;

    TempsFin=(Heure*60)+Minutes;

    cout << "\n\nLa durée du voyage est de "
        << TempsFin-TempsDebut << " minutes.";
}

```

ÉCRAN DE SORTIE

```

Entrez l'heure de départ:
    Heure: 12
    Minutes: 30

Entrez l'heure d'arrivée (doit être plus tard):
    Heure: 13
    Minutes: 45

La durée du voyage est de 75 minutes.

```

EXEMPLE 2

LE PROBLÈME

Écrire un programme qui permet d'additionner deux nombres réels quelconques et d'afficher le résultat de la somme.

QUE VEUT-ON À LA SORTIE?

La somme de deux nombres réels.

QUELLES SONT LES DONNÉES CONNUES?

Deux nombres réels.

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
Nba	premier nombre à additionner	float	lue à l'entrée
Nbb	deuxième nombre à additionner	float	lue à l'entrée
Somme	somme des nombres	float	Nba+Nbb

LE DIAGRAMME D'ACTION:

```
┌ Lire Nba, Nbb
│ Somme = Nba + Nbb
│ Écrire Nba, Nbb
└ Écrire Somme
```


LE PROGRAMME

```

/*****
AUTEUR:  Marcel L'Italien
FICHIER: 2cin2.cpp
DESCRIPTION: Calcul de la somme de deux nombres réels
*****/
#include <iostream>
#include <iomanip>    // Pour les manipulateurs de cout

using namespace std;

void main(void)
{
    float Nba,
          Nbb,
          Somme;

    // Lecture des variables avec un espace comme séparateur
    cout << "\nEntrez deux nombres réels séparés par un espace: ";
    cin >> Nba >> Nbb;

    // Calcul de la somme
    Somme = Nba+Nbb;

    // Impression de la somme
    cout << "\n\n\tLes nombres Nba et Nbb sont: "
         << fixed // Activer la notation virgule flottante
         << setprecision(2) // Toutes les valeurs affichées avec 2 décimales
         << setw(6) << Nba
         << setw(6) << Nbb
         << "\n\tLa Somme est: "
         << setw(6) << Somme;
}

```

ÉCRAN DE SORTIE

```
Entrez deux nombres réels séparés par un espace: 23.2 4.1
```

```

    Les nombres Nba et Nbb sont:  23.20  4.10
    La Somme est:  27.30

```

EXEMPLE 3

LE PROBLÈME

Écrire un programme qui calcule la facture totale d'un repas au restaurant. Le programme doit, à partir du montant du repas et du montant des consommations, calculer les différentes taxes applicables. Les taxes sont les suivantes:

- 7% sur le prix du repas (TPS)
- 7.5% sur le prix du repas et la TPS, (taxe de vente provinciale TVQ)
- 15% sur le repas (pourboire)

On désire afficher une facture détaillant chaque item.

QUE VEUT-ON À LA SORTIE?

Calculer le total de la facture d'un repas au restaurant.

QUELLES SONT LES DONNÉES CONNUES?

- Le montant du repas;
- Les différents pourcentages (taxes et service).

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
TVQ	pourcentage de la taxe provinciale	const float	0.075
POURBOIRE	pourcentage du pourboire	const float	0.15
TPS	pourcentage de la TPS	const float	0.07
PrixRepas	prix du repas	float	Lue à l'entrée
MontantPourboire	pourboire	float	$\text{PrixRepas} * \text{Service}$
MontantTPS	montant de la TPS	float	$\text{PrixRepas} * \text{TPS}$
MontantTVQ	montant de la TVQ	float	$(\text{PrixRepas} + \text{MontantTPS}) * \text{TVQ}$
TotalFacture	prix total de la facture	float	$\text{PrixRepas} + \text{MontantTPS} + \text{MontantTVQ} + \text{MontantServicer}$

LE DIAGRAMME D'ACTION:

```
— Lire PrixRepas
— MontantTPS = PrixRepas*TPS
— MontantTVQ = (PrixRepas+MontantTPS)*TVQ
— MontantPourboire = PrixRepas * POURBOIRE
— TotalFacture = PrixRepas + MontantTPS + MontantTVQ + MontantPourboire
— Écrire PrixRepas
— Écrire MontantTPS
— Écrire MontantTVQ
— Écrire MontantPourboire
— Écrire TotalFacture
```


EXEMPLE 4

```

/*****
AUTEUR:  Marcel L'Italien
FICHIER: 2cin4.cpp
DESCRIPTION: Lecture d'une chaîne de caractères.
            Utilisation de la classe string.
*****/
#include <iostream>
#include <string>          //pour la classe string

using namespace std;

void main(void)
{
    /* Pour manipuler plusieurs caractères on doit définir une chaîne de car.
       Pour cela nous devons utiliser la classe string */
    string Prenom;

    cout << "\nEntrez votre prénom: ";
    cin >> Prenom;

    cout << "\n\n\t\t\tBonjour, " << Prenom;
}

```

ÉCRAN DE SORTIE

Entrez votre prénom: Marcel

Bonjour, Marcel

EXEMPLE 5

```
/******  
AUTEUR:  Marcel L'Italien  
FICHIER: 2cin5.cpp  
DESCRIPTION: Affectation et concaténation de chaînes de caractères.  
*****/  
#include <iostream>  
#include <string>  
  
using namespace std;  
  
void main(void)  
{  
    string Nom, Prenom, NomFamille;  
  
    // Saisi du nom de l'utilisateur au clavier  
    cout << "\nEntrez votre prénom: ";  
    cin >> Prenom;  
    cout << "Entrez votre nom: ";  
    cin >> NomFamille;  
  
    // Concaténation du prénom et du nom,  
    Nom = Prenom + " " + NomFamille;  
  
    // Impression du nom complet  
    cout << "\n\n\tBonjour, " << Nom;  
}
```

ÉCRAN DE SORTIE

```
Entrez votre prénom: Guy  
Entrez votre nom: Tare  
  
        Bonjour, Guy Tare
```

3. LES FONCTIONS `_GETCHE()` ET `_GETCH()`

Ces deux fonctions permettent de lire un seul caractère à partir du clavier. Le flux *cin* permet lui aussi de lire un seul caractère, alors pourquoi vous présenter ces fonctions?

Voici. Nous avons vu précédemment qu'il faut toujours faire **<retour>** pour terminer une lecture à partir du flux *cin*. Il y a toutefois des situations où on voudrait l'éviter. Par exemple, considérez les énoncés suivants:

```
char Reponse;  
...  
cout << "Pressez sur une touche pour continuer";  
cin >> Reponse;
```

Ici l'utilisateur **doit** presser une touche quelconque suivie de **<retour>**. On ne peut faire **<retour>** seul, car les tous les *espaces initiaux* sont ignorés (voir page 42)

Pour régler cette situation nous utiliserons la fonction `_getche()`: Par exemple:

```
char Reponse;  
...  
cout << "Pressez sur une touche pour continuer";  
Reponse = _getche();
```

La fonction `_getch()` est identique à `_getche()`, sauf que le caractère lu n'est pas affiché (utile pour la lecture d'un mot de passe...).

Ces fonctions font partie de la bibliothèque **conio.h**.

4. LA FONCTION GETLINE()

Considérez les énoncés suivants, et leur exécution:

```
cout << "Entrez votre prénom: ";
cin >> Prenom;
cout << "Entrez votre nom: ";
cin >> Nom;
cout << "Bonjour " << Prenom << ' ' << Nom;
```

EXÉCUTION 1 (l'utilisateur se nomme *Jean Tie*)

```
Entrez votre prénom: Jean
Entrez votre nom: Tie
Bonjour Jean Tie
```

EXÉCUTION 2 (l'utilisateur se nomme *Jean Bon Eau*)

```
Entrez votre prénom: Jean Bon
Entrez votre nom:
Bonjour Jean Bon
```

En observant la seconde exécution nous constatons que le nom ne semble pas être lu, et donc qu'il n'est pas affiché. En fait, il y a deux valeurs à lire (*Prenom* et *Nom*) et deux valeurs ont été entrées (*Jean* et *Bon*). Ici on considère donc que le prénom est *Jean*, et le nom *Bon*. Le problème provient du fait que lors d'une lecture à partir de *cin*, l'espace sert toujours à délimiter deux valeurs différentes à lire.

Ce que nous recherchons ici c'est donc un moyen de lire des chaînes de caractères incluant des espaces. La lecture de la chaîne se terminera par un **<enter>**. La solution consistera à utiliser la fonction *getline()*. Voici la correction au problème précédent:

```
cout << "Entrez votre prénom: ";
getline(cin, Prenom); /* fonction qui lit tous les caractères d'une chaîne
                       (incluant les espaces) et ce jusqu'à ce qu'on termine
                       par <entrée> */
cout << "Entrez votre nom: ";
getline(cin, Nom);
cout << "Bonjour " << Prenom << ' ' << Nom;
```

EXÉCUTION 3 (l'utilisateur se nomme *Jean Bon Eau*)

```
Entrez votre prénom: Jean Bon
Entrez votre nom: Eau
Bonjour Jean Bon Eau
```

ATTENTION : La fonction *getline()* lira une chaîne vide dans le cas où il reste un **\n** dans le tampon de lecture.

2.3. LES FONCTIONS **GOTOXY()** ET **CLRSCR()**

Voici deux fonctions permettant un plus grand contrôle de la mise en page à l'écran. Ces fonctions font partie de la bibliothèque maison **cvm.h**.

La première (*gotoxy()*) permet de déplacer le curseur à l'écran, et ce à n'importe quelle position. Pour utiliser cette fonction il faudra spécifier deux valeurs (deux paramètres). Il s'agit des coordonnées où le curseur devra se positionner, i.e. le numéro de la colonne (0 à 79) ainsi que celui de la ligne (0 à 24).

Exemples: `gotoxy(5,15)` → place le curseur à la colonne 6 et à la ligne 16

```
ligne=0;
```

```
col=0;
```

```
gotoxy(col,ligne) → place le curseur en haut et à gauche de l'écran  
                    (première colonne et première ligne)
```

La seconde fonction (*clrscr()*) permet d'effacer tout le contenu de l'écran de sortie.

Exemple: `clrscr();`

Notez que l'écran de sortie est vide au début de l'exécution de chaque programme, donc qu'il n'y a pas à l'effacer avec *clrscr()* à ce moment.

2.4 LA FONCTION MESSAGEBOXA()

Considérez les énoncés suivants:

```
cout << "Ce produit n'existe pas"
cout << "Pressez sur une touche pour continuer."
_getch();
```

Ces énoncés indiquent à l'utilisateur qu'il y a une erreur (produit qui n'existe pas). L'utilisateur doit ensuite presser une touche pour continuer, laquelle sera lue par la fonction `_getch()`.

Au lieu de la méthode précédente, que diriez-vous d'informer l'utilisateur de son erreur par la fenêtre suivante:



C'est très facile, et ça ne nécessite même pas de connaître la programmation *Windows*. Il s'agit d'utiliser la fonction `MessageBoxA()`. Pour faire apparaître cette fenêtre on a fait:

```
MessageBoxA(NULL, "Ce produit n'existe pas", "Erreur", MB_OK|MB_ICONSTOP);
```

L'utilisateur pourra continuer en cliquant sur le bouton *OK*, en pressant la touche<Enter> ou en fermant la fenêtre.

➔Voici la description complète de la fonction **MessageBoxA()**:

DESCRIPTION:

La fonction `MessageBoxA()` affiche et gère une fenêtre de message. La fenêtre de message contient: un **message**, un **titre**, un ou des **boutons**, et une **icône** (facultatif).

FICHER D'EN-TÊTE:

```
#include <windows.h>
```

PROTOTYPE:

```
int MessageBoxA(Fenêtre Parent, Texte, Titre, Style);
```

Description des paramètres:

Fenêtre Parent Identifie la fenêtre parent de la fenêtre message. Si ce paramètre est **NULL**, c'est qu'il n'y a pas de parent.

Texte Le message à afficher.

Titre Le titre de la fenêtre.

Style Spécifie le contenu de la fenêtre de message. On peut y mettre toute combinaison des valeurs suivantes:

nombre de bouton:

MB_ABORTRETRYIGNORE	affiche trois boutons: Abandonner, Réessayer, et Ignorer.
MB_OK	affiche un bouton: OK
MB_OKCANCEL	affiche deux boutons: OK, Annuler
MB_RETRYCANCEL	affiche deux boutons: Réessayer, Annuler.
MB_YESNO	affiche deux boutons: Oui, Non.
MB_YESNOCANCEL	affiche trois boutons: Oui, Non, Annuler.

bouton actif:

MB_DEFBUTTON1	définit le premier bouton comme celui qui est actif.
MB_DEFBUTTON2	définit le deuxième bouton comme celui qui est actif.
MB_DEFBUTTON3	définit le troisième bouton comme celui qui est actif.

icône:

MB_ICONEXCLAMATION	l'icône «!» (point d'exclamation)
MB_ICONINFORMATION	l'icône «i» minuscule
MB_ICONQUESTION	l'icône «?» (point d'interrogation)
MB_ICONSTOP	l'icône «X» (stop)

comportement:

MB_SYSTEMMODAL	la fenêtre restera visible jusqu'à réponse de l'utilisateur
-----------------------	---

Description de la valeur de retour:

IDABORT	le bouton <i>Abandonner</i> a été sélectionné
IDCANCEL	le bouton <i>Annuler</i> a été sélectionné
IDIGNORE	le bouton <i>Ignorer</i> a été sélectionné
IDNO	aucun bouton n'a été sélectionné
IDOK	le bouton <i>OK</i> a été sélectionné
IDRETRY	le bouton <i>Réessayer</i> a été sélectionné
IDYES	le bouton <i>Oui</i> a été sélectionné

Considérez les énoncés suivants:

```
do
{
    cout << "Voulez-vous continuer (O/N)?: "
    Reponse = _getche();
    Reponse = toupper(Reponse);
}
while(Reponse != 'O' && Reponse != 'N');

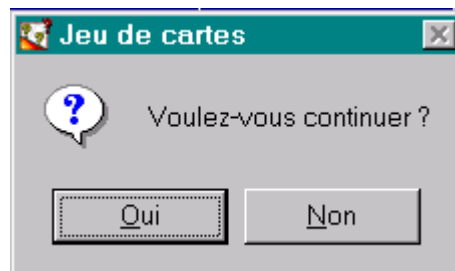
if(Reponse == 'O')
{
    ...
}
else
{
    ...
}
```

Ici on demande à l'utilisateur s'il veut continuer. L'utilisateur doit répondre par une lettre (**o**, **n**, **O** ou **N**) sinon on lui repose la question. On pourrait faire la même chose, mais avec la fonction *MessageBoxA*().

```
int Reponse;
Reponse = MessageBoxA( NULL, "Voulez-vous continuer ?", "Jeu de cartes",
                        MB_YESNO|MB_ICONQUESTION|MB_SYSTEMMODAL);

if (Reponse == IDYES)
{
    ...
}
else
{
    ...
}
```

La fenêtre affichée sera:



La fonction *MessageBoxA*() est une fonction d'entrée et de sortie. Elle peut donc, dans certains cas, remplacer les flux *cin* et *cout*.

2.5 EXERCICES

1. Nommez tous les types de valeurs acceptés par *cout*.
2. Comment la précision d'un nombre est-elle spécifiée lors de l'affichage d'un réel?
3. Donnez deux séquences d'échappement et expliquez leur rôle.

4. Quel sera le résultat du programme suivant:

```
#include <iostream>
void main(void)
{
    cout<< "Bienvenue aux cours de \"101\"\\net \"131\"";
}
```

5. Soit le programme suivant:

```
#include <iostream>
using namespace std;
void main(void)
{
    int ValI, ValJ, ValK;
    ValI=2;
    ValJ=3;
    ValK=4;
}
```

Compléter le programme avec un seul énoncé permettant d'afficher chacune des expressions suivantes:

- a) ValI, ValJ, et ValK
- b)(ValI+ValJ) et (ValI-ValK)

6. Soit un programme contenant les énoncés suivants:

```
#include <iostream>
#include <string>
using namespace std;
void main(void)
{
    int EntierI, EntierJ;
    long EntierLong;
    unsigned int SansSigne;
    float Reel;
    double ReelDouble;
    char Carac;
    string Chaine;
}
```

Écrire un seul énoncé permettant d'afficher chacun des groupes de variables ci-après:

- a) EntierI, EntierJ, Reel, ReelDouble
- b) EntierI, EntierLong, EntierJ, Reel, SansSigne
- c) EntierI, SansSigne, Carac, Chaine
- d) Carac, Chaine, Reel, ReelDouble, EntierLong

7. Reprendre l'exercice précédent en faisant afficher, en plus de la valeur de la variable, son nom. Séparez chaque variable de la suivante par une tabulation.

8. Soit un programme contenant les énoncés suivants:

```
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;
void main(void)
{
    int EntierI, EntierJ;
    long EntierLong;
    unsigned int SansSigne;
    float Reel;
    double ReelDouble;
    char Carac;
    string Chaine;
}
```

Écrire un énoncé permettant d'afficher chacun des groupes de variables ci-après, en respectant les spécifications demandées.

a) EntierI, EntierJ, Reel, ReelDouble

chaque entier dans un champ de 4 caractères;

chaque réel dans un champ de 14 caractères, avec une précision de 8;

avec une tabulation entre chaque valeur.

b) EntierI, EntierLong, Reel, SansSigne

chaque entier dans un champ de 5 caractères;

chaque entier long dans un champ de 10 caractères;

chaque réel dans un champ de 12 caractères, avec une précision de deux et en notation scientifique;

une valeur par ligne.

c) EntierI, SansSigne, Carac, Chaine

chaque valeur numérique dans un champ de 6 caractères;

trois espaces séparant chaque valeur;

deux valeurs par ligne;

aligner les entiers à gauche.

d) Carac, Chaine, Reel, ReelDouble, EntierLong

chaque Chaîne de caractères dans un champ de 40 caractères;

chaque caractère dans un champ de 10 caractères

chaque réel dans un champ de 20 caractères, avec une précision de 5;

chaque entier long dans un champ de 10 caractères.

9. Soit un programme contenant les énoncés suivants:

```
#include <iostream>
#include <iomanip>
using namespace std;
void main(void)
{
    int EntI, EntJ, EntK;
    long EntLg;
    unsigned SansSigne;
    EntI=12345;
    EntJ=-13579;
    EntK=-24680;
    EntLg=123456789;
    SansSigne=5555;
}
```

Décrire l'affichage généré par chacun des énoncés suivants:

- a) `cout << EntI << EntJ << EntK << EntLg << SansSigne;`
- b) `cout << setw(3) << EntI << setw(3) << EntJ << setw(3) << EntK << setw(3) << EntLg << setw(3) << SansSigne;`
- c) `cout << setw(8) << EntI << setw(8) << EntJ << setw(8) << EntK << setw(8) << EntLg << setw(8) << SansSigne;`
- d) `cout << left << setw(8) << EntI << setw(8) << EntJ << setw(8) << EntK << setw(8) << EntLg << setw(8) << SansSigne;`
- e) `cout << oct << EntI << oct << EntJ << oct << EntK << oct << EntLg << oct << SansSigne;`
- f) `cout << hex << EntI << hex << EntJ << hex << EntK << hex << EntLg << hex << SansSigne;`

10. Soit un programme contenant les énoncés suivants:

```
#include <iostream>
#include <iomanip>
using namespace std;
void main(void)
{
    float ReelA, ReelB, ReelC;
    double ReelDouble;
    ReelA=2.5;
    ReelB=0.0005;
    ReelC=3000;
    ReelDouble=12345.6789;
}
```

Décrire l'affichage généré par chacun des énoncés suivants:

- a) `cout << ReelA << ReelB << ReelC << ReelDouble;`
- b) `cout << fixed << setw(8) << ReelA << setw(8) << ReelB << setw(8) << ReelC << setw(8) << ReelDouble;`

c) `cout << fixed << setw(8) << setprecision(4) << ReelA << setw(8)`
`<< setprecision(4) << ReelB << setw(8) << setprecision(4) << ReelC`
`<< setw(8) << setprecision(4) << ReelDouble;`

d) `cout << fixed << setw(8) << setprecision(2) << ReelA << setw(8)`
`<< ReelB << setw(8) << ReelC << setw(8) << ReelDouble;`

e) `cout << fixed << setw(12) << setprecision(2) << ReelA << setw(12)`
`<< ReelB << setw(12) << ReelC << setw(12) << ReelDouble;`

11. Complétez le programme suivant. Les nombres entiers doivent être placés dans un champ de 10 positions. Les nombres réels doivent être justifiés à gauche dans un champ de 12 positions avec une précision de 3 chiffres.

```
#include <iostream>
#include <iomanip>

using namespace std;

void main(void)
{
    char Carac;
    int Entier1,
        Entier2;
    float Reel1,
        Reel2;

    Carac = 'd';
    Entier1 = 345;
    Entier2 = 78;
    Reel1 = 123456.567;
    Reel2 = 0.87654;

    cout << "Carac = " << Carac << '\n';

    cout << "\nEntier1 = " << Entier1;
        << fixed
        << "Reel1 = " << Reel1;

    cout << "\nEntier1 = " << Entier2;
        << "Reel1 = " << Reel2;
}
```

12. Indiquer quelles seront les valeurs **affichées** des variables Reel1 et Reel2 du problème précédent.

13. Nommer les fichiers (#include) fréquemment utilisés pour les entrées-sorties.

14. Quel sera le résultat du programme suivant:

```
#include <iostream>

using namespace std;

void main(void)
{
    float Base, Hauteur, Surface;

    cout << "Base: ";
    cin >> Base;

    cout << "Hauteur: ";
    cin >> Hauteur;

    Surface=(Base*Hauteur)/2;
    cout << "Surface: " << Surface;
}
```

15. Soit le programme suivant:

```
#include <iostream>
#include <string>

using namespace std;

void main(void)
{
    string    Chaine;
    int       EntierI,EntierJ;
    long      EntierLong;
    unsigned  SansSigne;
    float     Reel;
    double    ReelDouble;
    char      Carac;
}
```

Écrire un seul énoncé utilisant le flux *cin*, permettant de saisir chacun des groupes de variables ci-après.

- a) EntierI, EntierJ, Reel, ReelDouble
- b) EntierI, EntierLong, EntierJ, Reel, SansSigne
- c) EntierI, SansSigne, Carac, Chaine
- d) Carac, Chaine, Reel, ReelDouble, EntierLong

16. Écrire le code (programme en C++) de ce problème:

LE PROBLÈME

On lit au clavier une température en degrés centigrades et on veut calculer son équivalent en fahrenheit.

QUE VEUT-ON À LA SORTIE?

Une température en fahrenheit

QUELLES SONT LES DONNÉES CONNUES?

Une valeur représentant une température en degrés centigrades.

L'ÉCRAN DE SAISIE ET RÉSULTATS:

Donner la température en centigrade: xxx.xx

L'équivalent en fahrenheit est de xxx.xx

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
CentiTemp	température en centigrade	float	lue à l'entrée
FahrenTemp	température en fahrenheit	float	$(\text{CentiTemp} * 9/5) + 32$

LE DIAGRAMME D'ACTION:

```
┌ Lire CentiTemp
├  $\text{FahrenTemp} = (\text{CentiTemp} * 9/5) + 32$ 
└ Écrire FahrenTemp
```

17. Écrire le code (programme en C++) de ce problème

LE PROBLÈME

À partir de la mesure du rayon d'une sphère on veut calculer l'aire de cette sphère.

QUE VEUT-ON À LA SORTIE?

L'aire de la sphère.

QUELLES SONT LES DONNÉES CONNUES?

Le rayon de la sphère.

L'ÉCRAN DE SAISIE ET RÉSULTATS:

Donner le rayon de la sphère: xxx.xx

L'aire de la sphère est de: xxx.xx

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
PI	la constante mathématique pi	const float	3.14159
Rayon	rayon de la sphère	float	lue à l'entrée
Aire	aire de la sphère	float	$4*PI*Rayon*Rayon$

LE DIAGRAMME D'ACTION:

```
┌ Lire Rayon  
├ Aire = 4*PI*Rayon*Rayon  
└ Écrire Aire
```

18. Compléter l'analyse de ce problème, puis écrire le code.

LE PROBLÈME

Écrire un programme qui lit un poids en livres et le convertit en kilogrammes et en grammes. (Une livre vaut 0.45359237 kilogramme ou 453.59237 grammes)

QUE VEUT-ON À LA SORTIE?

Un poids en grammes et un poids en kilogrammes.

QUELLES SONT LES DONNÉES CONNUES?

Un poids en livres.

L'ÉCRAN DE SAISIE ET RÉSULTATS:

Donner le poids en livres: xxxx.xx

Le poids en grammes est de: xxxx.xx

Le poids en kilogrammes est de: xxxx.xx

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
PoidsGramme	poids en gramme	float	
PoidsKilo	poids en kilogramme	float	
PoidsLivre	poids en livre	float	lue à l'entrée

LE DIAGRAMME D'ACTION:

à compléter

19. Compléter l'analyse de ce problème, puis écrire le code.

LE PROBLÈME

Écrire un programme qui permet de calculer la consommation moyenne d'une voiture (litres/100km).

QUE VEUT-ON À LA SORTIE?

La consommation moyenne d'une voiture

QUELLES SONT LES DONNÉES CONNUES?

Le nombre de litres consommés et le nombre de kilomètres parcourus.

L'ÉCRAN DE SAISIE ET RÉSULTATS:

Donner le nombre de litres: xxx.xx

Donner le nombre de kilomètres: xxx.xx

La consommation moyenne est de: xx.xx/100km

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
NbLitres	nombre de litres consommés	float	lue à l'entrée
NbKilo	nombre de kilomètres parcourus	float	lue à l'entrée
Consommation	consommation litre/100km	float	

LE DIAGRAMME D'ACTION:

à compléter

CHAPITRE 3

3.1 LA SÉLECTION

L'activité humaine est, souvent, une suite d'actions dans des situations où il y a plusieurs possibilités. Cela nous amène à prendre des décisions, donc à sélectionner une action plutôt qu'une autre. Par exemple, avant de quitter la maison on peut se demander: pleut-il? S'il pleut, on quitte alors la maison avec un parapluie, sinon on quitte la maison sans parapluie. Autre exemple, si on devait se rendre au cégep, on pourrait prendre une décision de la façon suivante: si le cégep est à moins d'un kilomètre, je marche, sinon je prend l'autobus.

En informatique, quand nous sommes placés devant une situation offrant plusieurs possibilités, nous la représentons à l'aide d'une instruction de sélection. La question qui permettra de sélectionner l'action à faire sera la condition.

3.2 LES SÉLECTIONS DANS LES DIAGRAMMES D'ACTION

Les instructions de sélection, dans les diagrammes d'action, s'écrivent selon les règles qui suivent:

règle 1: la condition menant à la sélection doit être exprimée de manière à n'offrir que deux réponses possibles: **vrai** ou **faux**;

règle 2: si des actions doivent être effectuées dans les deux cas possibles (réponse vraie et réponse fausse), la sélection s'exprimera par le diagramme qui suit:

```
[ Si condition
  bloc-instructions lorsque la condition est vraie
[ Sinon
  bloc-instructions lorsque la condition est fausse
```

règle 3: si pour l'une des deux réponses possibles il n'y a aucune action à effectuer, la sélection s'exprimera par le diagramme qui suit:

```
[ Si condition
  bloc-instructions lorsque vrai
```

➔ nous n'utiliserons jamais la forme suivante:

```
[ Si condition
[ Sinon
  bloc-instructions lorsque la condition est fausse
```

Dans l'exemple de la pluie, nous pourrions représenter notre choix par l'un des trois diagrammes suivants:

```

[ [ Si il pleut
  [ je prends un parapluie
  [ je quitte la maison

```

```

[ [ Si il pleut
  [ je prends un parapluie
  [ je quitte la maison
  [ Sinon
  [ je quitte la maison

```

```

[ [ Si il ne pleut pas
  [ je quitte la maison
  [ Sinon
  [ je prends un parapluie
  [ je quitte la maison

```

Quoiqu'on arrive toujours au même résultat, la première solution est la mieux structurée, car elle est la seule à ne pas utiliser le même énoncé *je quitte la maison* dans les deux blocs (vrai et faux).

Observez que la condition du troisième exemple est la négation de celle des deux exemples précédents. Il arrivera souvent, pour faciliter la compréhension, qu'on veuille inverser les énoncés du bloc *vrai* avec ceux du bloc *faux*. Dans un tel cas il s'agira d'écrire la condition inverse (négation). Nous le ferons aussi pour respecter la règle 3 énoncée plus haut, et éviter que le bloc *vrai* soit vide.

3.3 LA CONDITION SIMPLE

Une condition simple est constituée des deux membres liés par un opérateur relationnel. Chaque membre peut être une constante, une variable ou une expression.

Les opérateurs relationnels sont les suivants:

- ==** vérifie si les deux membres de la condition sont égaux;
- <** vérifie si le premier membre de la condition est plus petit que le deuxième;
- >** vérifie si le premier membre de la condition est plus grand que le deuxième;
- <=** vérifie si le premier membre de la condition est plus petit ou égal au deuxième;
- >=** vérifie si le premier membre de la condition est plus grand ou égal au deuxième;
- !=** vérifie si les deux membres de la condition sont différents.

3.4 LA CONDITION COMPOSÉE

Une condition composée est constituée de deux ou plusieurs conditions simples reliées par les opérateurs logiques (booléens) **et** ainsi que **ou**. L'opérateur logique **non**, s'appliquera à une condition simple ou à une condition composée.

L'évaluation des opérateurs logiques est comme suit:

- &&** (et) donne un résultat vrai si, et seulement si, toutes les conditions simples sont vraies;
- ||** (ou) donne un résultat vrai, si au moins une des conditions simples est vraie;
- !** (non) donne un résultat vrai si la condition est fausse, donne un résultat faux si la condition est vraie.

3.5 NOTION DE BLOC EN LANGAGE C++

Avant de présenter les instructions du langage C++ correspondant aux énoncés de sélection des diagrammes d'action, nous devons nous familiariser avec la notion de *bloc d'instructions*. En C++, un bloc est une ou plusieurs instructions encadrées par une paire d'accolades ({ }). Lorsqu'il n'y a qu'une seule instruction, les accolades sont facultatives.

Un bloc est donc un regroupement d'instructions. Il peut s'agir de toute instruction, y compris une instruction incluant elle-même un ou plusieurs autres blocs (comme *if()-else*). Il est donc question, ici, d'imbrication des instructions. Cette notion est fondamentale car nous l'utiliserons dans les instructions de sélections, de même que dans les instructions de répétitions (chapitre 4).

Bloc d'instructions:

En langage C++, un *bloc d'instructions* aura une des deux formes suivantes:

```
énoncé;            ou            {  
                               énoncé 1;  
                               énoncé 2;  
                               ...  
                               énoncé n;  
                               }
```

N.B. Un énoncé peut être un *if()*, un *if()-else*, ou toute autre instruction comportant elle-même des blocs. Une telle instruction est considérée comme **un seul énoncé**.

3.6 L'INSTRUCTION IF()

En langage C++ une sélection respectant la règle 3 (énoncée au point 3.2) s'écrira avec l'instruction *if()*.

L'instruction *if()* sert à déterminer si un bloc d'instructions sera exécuté ou non à l'aide de l'évaluation d'une condition. Si la condition est vraie, alors le bloc qui suit immédiatement l'instruction *if()* est exécuté. Si la condition est fausse, ce bloc d'instructions est ignoré.

syntaxe: `if(condition)`
 `bloc-instructions`

exemples de *if()*:

```
if (condition)
    énoncé;
```

```
if (condition)
{
    énoncé 1;
    énoncé 2;
    ...
    énoncé n;
}
```

```
if (n>10)
    cout << "n est plus grand que 10";
```

```
if (n>10)
{
    cout << "n est un nombre entier";
    cout << "\nplus grand que 10";
}
```

diagramme d'action correspondant:

```
┌─ Si condition
└─ énoncé
```

```
┌─ Si condition
├─ énoncé 1
├─ énoncé 2
├─ ...
└─ énoncé n
```

```
┌─ Si n>10
└─ Écrire "n est plus grand que 10"
```

```
┌─ Si n>10
├─ Écrire "n est un nombre entier"
└─ Écrire "plus grand que 10"
```


EXEMPLE 1

LE PROBLÈME:

Soient deux nombres **VarA** et **VarB** lus au clavier. On veut ranger le plus petit nombre dans la variable **VarA** et le plus grand nombre dans la variable **VarB**.

QUE VEUT-ON À LA SORTIE?

Les valeurs des variables **VarA** et **VarB**.

QUELLES SONT LES DONNÉES CONNUES?

Deux nombres différents, lus au clavier.

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
VarA	le premier nombre lu le plus petit après le traitement	int	lue à l'entrée
VarB	le deuxième nombre lu le plus grand après le traitement	int	lue à l'entrée
Echange	variable de travail permettant l'échange des valeurs entre VarA et VarB	int	VarA

LE DIAGRAMME D'ACTION:

```
┌ Lire VarA, VarB
├   ┌ Si VarA > VarB
│   │   Echange = VarA
│   │   VarA = VarB
│   └─ VarB = Echange
└ Écrire VarA, VarB
```

PROGRAMME

```

/*****
AUTEUR:  Marcel L'Italien
FICHIER: 3sil.cpp
DESCRIPTION: Ranger le plus petit de deux nombres dans la variable VarA et
             le plus grand dans la variable VarB.
*****/
#include <iostream>

using namespace std;

void main(void)
{
    int VarA,
        VarB,
        Echange;

    cout << "Donner le premier nombre: ";
    cin >> VarA;
    cout << "Donner le deuxième nombre: ";
    cin >> VarB;

    /* Vérification que a est bien le plus grand des deux nombres */
    if (VarA>VarB)
    {
        Echange=VarA;
        VarA=VarB;
        VarB=Echange;
    }

    cout << "\nLe plus petit nombre (variable VarA) est: " << VarA;
    cout << "\nLe plus grand nombre (variable VarB) est: " << VarB;
}

```

ÉCRAN DE SORTIE

```

Donner le premier nombre: 7
Donner le deuxième nombre: 5

Le plus petit nombre (variable VarA) est: 5
Le plus grand nombre (variable VarB) est: 7

```

EXEMPLE 2

LE PROBLÈME:

On doit lire au clavier trois nombres: Nb1, Nb2 et Nb3. Ces nombres doivent être placés en ordre croissant. Le plus petit doit être rangé dans la variable Nb1 et le plus grand dans la variable Nb3.

QUE VEUT-ON À LA SORTIE?

Afficher les trois nombres en ordre croissant.

QUELLES SONT LES DONNÉES CONNUES?

Trois nombres lus au clavier.

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
Echange	variable de travail, permettant l'échange des valeurs entre deux variables	int	Nb1 ou Nb2
Nb1	le premier nombre	int	lue à l'entrée
Nb2	le deuxième nombre	int	lue à l'entrée
Nb3	le troisième nombre	int	lue à l'entrée

LE DIAGRAMME D'ACTION:

```
┌ Lire Nb1, Nb2, Nb3
├   ┌ Si Nb1 > Nb2
│   │ Echange = Nb1
│   │ Nb1 = Nb2
│   └ Nb2 = Echange
├   ┌ Si Nb1 > Nb3
│   │ Echange = Nb1
│   │ Nb1 = Nb3
│   └ Nb3 = Echange
├   ┌ Si Nb2 > Nb3
│   │ Echange = Nb2
│   │ Nb2 = Nb3
│   └ Nb3 = Echange
└ Écrire Nb1, Nb2, Nb3
```

PROGRAMME

```

/*****
AUTEUR:  Marcel L'Italien
FICHIER: 3si2.cpp
DESCRIPTION: Lire trois nombres et les placer en ordre croissant.
*****/
#include <iostream>
using namespace std;

void main(void)
{
    int Nb1,
        Nb2,
        Nb3,
        Echange;
    cout << "Donner le premier nombre (Nb1): ";
    cin >> Nb1;
    cout << "Donner le deuxième nombre (Nb2): ";
    cin >> Nb2;
    cout << "Donner le troisième nombre (Nb3): ";
    cin >> Nb3;

    /* Comparaison de la valeur de Nb1 par rapport à Nb2 et Nb3) */
    if (Nb1>Nb2)
    {
        Echange=Nb1;
        Nb1=Nb2;
        Nb2=Echange;
    }
    if (Nb1>Nb3)
    {
        Echange=Nb1;
        Nb1=Nb3;
        Nb3=Echange;
    }

    /* Comparaison de la valeur de Nb2 par rapport à Nb3) */
    if (Nb2>Nb3)
    {
        Echange=Nb2;
        Nb2=Nb3;
        Nb3=Echange;
    }
    cout << "\nLes nombres en ordre croissant sont:\n"
         << "\t\t\t" << Nb1 << "\t" << Nb2 << "\t" << Nb3;
}

```

ÉCRAN DE SORTIE

```

Donner le premier nombre (Nb1): 8
Donner le deuxième nombre (Nb2): 1
Donner le troisième nombre (Nb3): 4

Les nombres en ordre croissant sont:
                1         4         8

```

EXEMPLE 3

LE PROBLÈME:

Soit un nombre lu au clavier. On veut savoir si ce nombre est compris entre 0 et 100.

QUE VEUT-ON À LA SORTIE?

Un message indiquant si le nombre est entre 0 et 100.

QUELLES SONT LES DONNÉES CONNUES?

Un nombre.

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
MINIMUM	la borne inférieure de l'intervalle	const int	0
MAXIMUM	la borne supérieure de l'intervalle	const int	100
Nombre	nombre lu	int	lue à l'entrée

LE DIAGRAMME D'ACTION:

```
┌ Lire Nombre
└ ┌ Si MINIMUM<=Nombre && Nombre<=MAXIMUM
  └ Écrire Le nombre est dans l'intervalle permis
```

PROGRAMME

```
/* *****  
AUTEUR: Marcel L'Italien  
FICHIER: 3si3.cpp  
DESCRIPTION: Valider qu'un nombre est bien dans un intervalle donné  
***** */  
#include <iostream>  
  
using namespace std;  
  
void main(void)  
{  
    const int MINIMUM=0,  
             MAXIMUM=100;  
    int Nombre;  
  
    cout << "Entrer votre nombre: ";  
    cin >> Nombre;  
  
    /* Vérification que le nombre est dans l'intervalle */  
    if (MINIMUM<=Nombre && Nombre<=MAXIMUM)  
    {  
        cout << "\nLe nombre " << Nombre << " est dans l'intervalle permis";  
    }  
}
```

ÉCRAN DE SORTIE

Entrer votre nombre: 15 Le nombre 15 est dans l'intervalle permis
--

3.7 L'INSTRUCTION IF() - ELSE

L'instruction *if()*-*else* est employée quand on doit choisir entre deux blocs d'instructions **distincts**. Lorsque l'évaluation de la condition est vraie alors on exécute le bloc d'instructions qui suit le *if()*. Lorsque la condition est fausse c'est le bloc suivant le *else* qui est exécuté. Notez qu'un bloc peut n'être composé que d'un seul énoncé.

syntaxe:

```
if (condition)
    bloc-instructions
else
    bloc-instructions
```

exemples:

```
if (n>10)
    cout << "n > 10";
else
    cout << "n <= 10";
```

```
if (n>10)
{
    cout << "n est un nombre entier";
    cout << "\nplus grand que 10";
}
else
{
    cout << "n est un nombre entier";
    cout << "\nplus petit ou égal à 10";
}
```

diagramme d'action correspondant:

```
graph TD
    A[Si n>10] --> B[Écrire "n > 10"]
    A --> C[Si non]
    C --> D[Écrire "n <= 10"]
```

```
graph TD
    A[Si n>10] --> B[Écrire "n est un nombre entier"]
    A --> C[Écrire "plus grand que 10"]
    A --> D[Si non]
    D --> E[Écrire "n est un nombre entier"]
    D --> F[Écrire "plus petit ou égal à 10"]
```

N.B. Quand nous imbriquons plusieurs énoncés *if()*-*else* les uns dans les autres, et qu'un des énoncés *if()* imbriqués ne comprend pas de clause *else*, il faut alors que ce *if()*, et tous ses énoncés, soit placé entre accolades. Ceci permettra au compilateur de bien comprendre que le *else* qui suit cet énoncé appartient au *if()* du niveau hiérarchique supérieur.

exemple:

```
if (Note >= 0 && Note <= 100)
{
    if (Note>=60)
        cout << "Cours réussi";
    }
else
    cout << "Note invalide";
```

EXEMPLE 1

LE PROBLÈME:

Lire, au clavier, deux nombres et afficher le plus grand.

QUE VEUT-ON À LA SORTIE?

Afficher le plus grand nombre.

QUELLES SONT LES DONNÉES CONNUES?

Deux nombres lus au clavier.

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
NbA	valeur du 1 ^{er} nombre	int	lue à l'entrée
NbB	valeur du 2 ^e nombre	int	lue à l'entrée
Grand	le plus grand des deux nombres	int	NbA ou NbB

LE DIAGRAMME D'ACTION

```
┌ Lire NbA, NbB
├   ┌ Si NbA>NbB
├   │ Grand=NbA
├   └ Sinon
├   │ Grand=NbB
└ Écrire Grand
```


PROGRAMME

```

/*****
AUTEUR: Marcel L'Italien
FICHIER 3si4.cpp
DESCRIPTION: Trouver le plus grand de deux nombres
*****/
#include <iostream>

using namespace std;

void main(void)
{
    int NbA,
        NbB,
        Grand;

    cout << "Donner le premier nombre: ";
    cin >> NbA;

    cout << "Donner le deuxième nombre: ";
    cin >> NbB;

    /* Recherche du plus petit des deux nombres */
    if (NbA>NbB)
        Grand=NbA;
    else
        Grand=NbB;

    cout << "\nLe plus grand nombre est: " << Grand;
}

```

ÉCRAN DE SORTIE

```

Donner le premier nombre: 10
Donner le deuxième nombre: 15

Le plus grand nombre est: 15

```

EXEMPLE 2

LE PROBLÈME:

Lire un nombre entier, au clavier, puis trouver s'il est pair ou impair..

QUE VEUT-ON À LA SORTIE?

On veut la parité d'un entier.

QUELLES SONT LES DONNÉES CONNUES?

Un nombre lu au clavier.

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
Nombre	nombre à traiter	int	lue à l'entrée
Reste	reste de la division de Nombre par 2	int	Nombre%2

LE DIAGRAMME D'ACTION:

```
┌ Lire Nombre
├ Reste=Nombre%2
├ ┌ Si Reste==0
│ │ Écrire Nombre, " est un nombre pair"
│ └ Sinon
└ ┌ Écrire Nombre, " est un nombre impair"
```

PROGRAMME

```
/******  
AUTEUR:  Marcel L'Italien  
FICHIER: 3si5.cpp  
DESCRIPTION: Trouver la parité d'un entier  
*****/  
#include <iostream>  
  
using namespace std;  
  
void main(void)  
{  
    int Nombre,  
        Reste;  
  
    cout << "Donner votre nombre: ";  
    cin >> Nombre;  
  
    Reste=Nombre%2;  
  
    if (Reste==0)  
        cout << "\n\n" << Nombre << " est un nombre pair";  
    else  
        cout << "\n\n" << Nombre << " est un nombre impair";  
}
```

ÉCRAN DE SORTIE (test impair):

```
Donner votre nombre: 5  
  
5 est un nombre impair
```

ÉCRAN DE SORTIE (test pair):

```
Donner votre nombre: 4  
  
4 est un nombre pair
```

EXEMPLE 3

LE PROBLÈME:

Lire un nombre entier puis trouver sa valeur absolue, ainsi que s'il est positif ou négatif

QUE VEUT-ON À LA SORTIE?

On veut la valeur absolue d'un entier et un message indiquant si le nombre est positif ou négatif.

QUELLES SONT LES DONNÉES CONNUES?

Un nombre lu au clavier.

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
Abs	valeur absolue du nombre	int	Nombre ou -Nombre
Nombre	nombre à traiter	int	lue à l'entrée

LE DIAGRAMME D'ACTION:

```
┌ Lire Nombre
├   ┌ Si Nombre<0
│   │   Écrire "Le nombre est négatif"
│   │   Abs=-Nombre
│   └─ Sinon
│       Écrire "Le nombre est positif"
│       Abs=Nombre
└   Écrire Abs
```

PROGRAMME

```
/*
AUTEUR: Marcel L'Italien
FICHIER: 3si6.cpp
DESCRIPTION: Calculer la valeur absolue d'un entier lu au clavier
*/
#include <iostream>

using namespace std;

void main(void)
{
    int Nombre,
        Abs;

    cout << "Entrer un nombre: ";
    cin >> Nombre;

    if (Nombre<0)
    {
        cout << "\nLe nombre est négatif";
        Abs = -Nombre;
    }
    else
    {
        cout << "\nLe nombre est positif";
        Abs = Nombre;
    }

    cout << "\nLa valeur absolue du nombre est de: " << Abs;
}
```

ÉCRAN DE SORTIE (test positif):

```
Entrer un nombre: 25
Le nombre est positif
La valeur absolue du nombre est de: 25
```

ÉCRAN DE SORTIE (test négatif):

```
Entrer un nombre: -56
Le nombre est négatif
La valeur absolue du nombre est de: 56
```

EXEMPLE 4

LE PROBLÈME

Trouver la relation qui existe entre deux nombres (plus petit, plus grand ou égal)

QUE VEUT-ON À LA SORTIE?

Les deux nombres et leur relation.

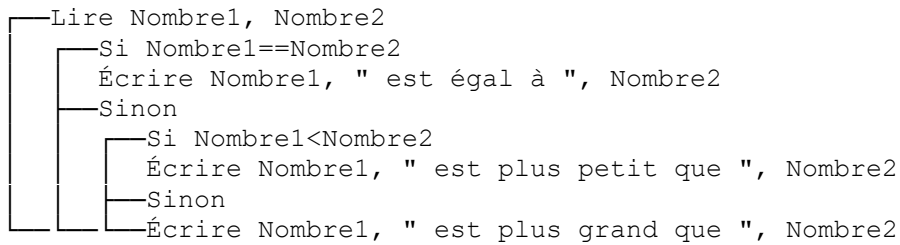
QUELLES SONT LES DONNÉES CONNUES?

Les deux nombres

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
Nombre1	le premier nombre	int	lue à l'entrée
Nombre2	le second nombre	int	lue à l'entrée

LE DIAGRAMME D'ACTION:



PROGRAMME

```

/*****
AUTEUR:  Marcel L'Italien
FICHIER: 3si7.cpp
DESCRIPTION: Comparer deux valeurs (<, > ou ==)
*****/
#include <iostream>

using namespace std;

void main(void)
{
    int Nombre1,
        Nombre2;

    cout << "Entrer les deux nombres: ";
    cin >> Nombre1 >> Nombre2;

    if (Nombre1==Nombre2)
        cout << "\n" << Nombre1 << " est égal à " << Nombre2;
    else
        if (Nombre1<Nombre2)
            cout << "\n" << Nombre1 << " est plus petit que " << Nombre2;
        else
            cout << "\n" << Nombre1 << " est plus grand que " << Nombre2;
}

```

ÉCRAN DE SORTIE

Entrer les deux nombres: 10 5
10 est plus grand que 5

→ Donnez deux jeux d'essais qui permettent de compléter les tests.

3.8 LES FONCTIONS `CIN.FAIL()`, `CIN.CLEAR()` ET `CIN.IGNORE()`

Il est courant que l'utilisateur fasse une erreur lorsqu'il entre une valeur au clavier. Avant de poursuivre un traitement il est préférable de vérifier que la lecture des données s'est bien déroulée. Cette vérification s'appelle *validation* des données.

Une première méthode de validation vous est fournie par l'exemple des pages 77 et 78. Dans cet exemple on vérifie que le nombre lu est bien compris entre 0 et 100. Toutefois que se passerait-il si l'utilisateur entrait la valeur **abc** au lieu d'un nombre entier? Voici les résultats produits par le programme **3si3.cpp** de la page 78:

ÉCRAN DE SORTIE

```
Entrer votre nombre: abc
Le nombre 1 est dans l'intervalle permis
```

Dans ce cas, puisque la valeur tapée au clavier n'est pas un entier, la variable **Nombre** n'est pas lue (donc son contenu reste inchangé). Comme le contenu initial (avant lecture) était un (1), le programme écrit alors «Le nombre 1 est dans l'intervalle permis»...

L'explication provient du fait que toute la lecture à partir du flux *cin* se poursuit tant que les données entrées sont correctes. On s'arrête dès qu'un caractère invalide est rencontré. Ici, le premier caractère entré étant incorrect (lettre **a** au lieu d'un chiffre) la lecture se termine avant toute affectation à la variable.

Une seconde méthode de validation, présentée à l'exemple suivant, consiste à vérifier que la lecture au clavier s'est effectuée correctement. Pour ce faire nous utiliserons la fonction *cin.fail()*. Cette fonction détecte toute erreur lors de la lecture à partir du flux *cin*. Elle retourne **vrai** lorsqu'une erreur fut commise lors de la dernière lecture, sinon elle retourne **faux**.

Suite à la détection d'une erreur avec *cin.fail()*, le flux *cin* devient inactif. Il n'est donc plus possible de faire une autre lecture, à moins de réactiver ce flux avec la fonction *cin.clear()*. De plus, puisque la dernière valeur entrée était incorrecte, et donc qu'elle ne fut pas affectée à la variable à lire, cette valeur est toujours présente dans le tampon de lecture. Il faudra donc vider le tampon de lecture avec *cin.ignore()*. Par exemple pour vider un tampon contenant jusqu'à 80 caractères, on fera:

→ `cin.ignore(80, '\n');`

ce qui effacera tous les caractères jusqu'au premier **'n'**, pour un maximum de 80 caractères.

EXEMPLE

```

/*****
AUTEUR:  Marcel L'Italien
FICHIER: 3si3a.cpp
DESCRIPTION: Valider qu'un nombre est bien dans un intervalle donné.
             Utilisation de cin.fail()
*****/
#include <iostream>

using namespace std;

void main(void)
{
    const int MINIMUM=0,
              MAXIMUM=100;
    int Nombre;

    cout << "Entrer votre nombre: ";
    cin >> Nombre;

    /* Vérification que le nombre est dans l'intervalle */
    if(cin.fail())
    {
        cout << "\nVous n'avez pas entré un nombre entier";
        cin.clear();           // pour réactiver le flux cin
        cin.ignore(80, '\n');  // pour vider le tampon
    }
    else
        if(MINIMUM<=Nombre && Nombre<=MAXIMUM)
            cout << "\nLe nombre " << Nombre << " est dans l'intervalle permis";
        else
            cout << "\nLe nombre " << Nombre << " n'est pas dans l'intervalle permis";
}

```

ÉCRAN DE SORTIE (valeur non numérique):

```

Entrer votre nombre: abc
Vous n'avez pas entré un nombre entier

```

ÉCRAN DE SORTIE (valeur numérique valide):

```

Entrer votre nombre: 10
Le nombre 10 est dans l'intervalle permis

```

ÉCRAN DE SORTIE (valeur numérique invalide):

```

Entrer votre nombre: 110
Le nombre 110 n'est pas dans l'intervalle permis

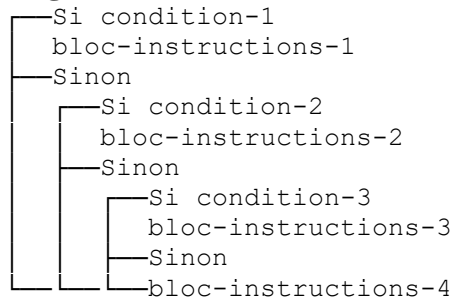
```

3.9 LA SÉLECTION MULTIPLE ET LES IF()-ELSE IMBRIQUÉS

Nous avons vu précédemment l'énoncé de sélection dans les diagrammes d'action ainsi que l'instruction *if()-else* qui permettent de faire un choix parmi deux alternatives. Lorsqu'un programme doit faire un choix parmi plusieurs possibilités, nous devons imbriquer plusieurs énoncés de sélection ou instructions *if()-else* (en fait il en faut **n-1**, où **n** représente le nombre de possibilités).

➔Exemple (faire un choix parmi 4 possibilités):

diagramme d'action:

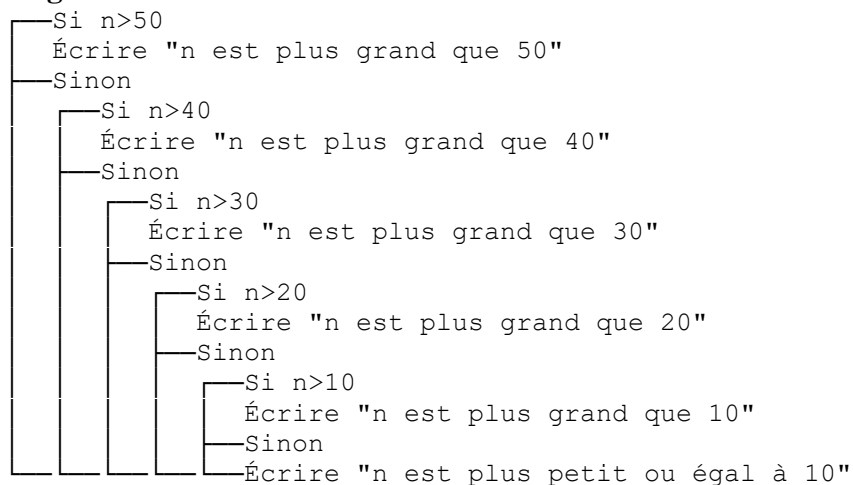


syntaxe en C:

```
if (condition-1)
    bloc-instructions-1
else
    if(condition-2)
        bloc-instructions-2
    else
        if(condition-3)
            bloc-instructions-3
        else
            bloc-instructions-4
```

➔Exemple (faire un choix parmi 6 possibilités):

diagramme d'action:



syntaxe en C++ (avec *sinon*)

```

if      (condition-1)
    bloc-instructions-1
else if (condition-2)
    bloc-instructions-2
else if (condition-3)
    bloc-instructions-3
else
    bloc-instructions-4

```

ou (sans *sinon*):

```

if      (condition-1)
    bloc-instructions-1
else if (condition-2)
    bloc-instructions-2
else if (condition-3)
    bloc-instructions-3
else if (condition-4)
    bloc-instructions-4

```

➔Exemple (faire un choix parmi 6 possibilités):

diagramme d'action (sélection multiple):

```

├─ Si n>50
│   Écrire "n est plus grand que 50"
├─ Si n>40
│   Écrire "n est plus grand que 40"
├─ Si n>30
│   Écrire "n est plus grand que 30"
├─ Si n>20
│   Écrire "n est plus grand que 20"
├─ Si n>10
│   Écrire "n est plus grand que 10"
├─ Sinon
│   Écrire "n est plus petit ou égal à 10"
└─

```

programme (nouvelle présentation):

```

if (n>50)
{
    cout<< "n est plus grand que 50;
}
else if(n>40)
{
    cout<<"n est plus grand que 40";
}
else if(n>30)
{
    cout<<"n est plus grand que 30";
}
else if(n>20)
{
    cout<<"n est plus grand que 20";
}
else if(n>10)
{
    cout<<"n est plus grand que 10";
}
else
{
    cout<<"n est plus petit ou égal à 10";
}

```

➔À chaque fois que possible, nous préférons la *sélection multiple* à des sélections simples imbriquées, ceci afin d'améliorer la lisibilité du programme, et d'en simplifier l'algorithme.

Exemple 1

LE PROBLÈME:

On lit au clavier la valeur de trois variables: CoteA, CoteB et CoteC. Celles-ci représentent la longueur des trois cotés d'un triangle. Il faut déterminer si le triangle est équilatéral (trois côtés égaux), isocèle (deux côtés égaux) ou scalène (trois côtés différents).

QUE VEUT-ON À LA SORTIE?

Le type du triangle déterminé par les trois longueurs.

QUELLES SONT LES DONNÉES CONNUES?

Trois valeurs qui représentent les longueurs des côtés d'un triangle.

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
CoteA	longueur d'un côté d'un triangle	int	lue à l'entrée
CoteB	longueur d'un côté d'un triangle	int	lue à l'entrée
CoteC	longueur d'un côté d'un triangle	int	lue à l'entrée

LE DIAGRAMME D'ACTION

```
┌ Lire CoteA, CoteB, CoteC
├ Écrire CoteA, CoteB, CoteC
├ ┌ Si CoteA==CoteB et CoteA==CoteC
│ │ Écrire "Le triangle est équilatéral"
│ └ Sinon
│   ┌ Si CoteA==CoteB ou CoteA==CoteC ou CoteB==CoteC
│   │ Écrire "Le triangle est isocèle"
│   └ Sinon
│     Écrire "Le triangle est scalène"
```

PROGRAMME

```

/*****
AUTEUR:  Marcel L'Italien
FICHIER: 3si8.cpp
DESCRIPTION: Lire la longueur des trois cotés d'un triangle, puis
             déterminer si ce triangle est équilatéral, isocèle ou scalène.
             Utilisation de if()-else imbriqués.
*****/
#include <iostream>

using namespace std;

void main(void)
{
    int CoteA,
        CoteB,
        CoteC;

    cout << "Donner la longueur du premier côté: ";
    cin >> CoteA;
    cout << "Donner la longueur du deuxième côté: ";
    cin >> CoteB;
    cout << "Donner la longueur du troisième côté: ";
    cin >> CoteC;
    cout << "\nLes longueurs des trois côtés sont:\t"
         << CoteA << "\t" << CoteB << "\t" << CoteC;

    if ((CoteA==CoteB) && (CoteA==CoteC))
        cout << "\n\nLe triangle est équilatéral";
    else
        if ((CoteA==CoteB) || (CoteA==CoteC) || (CoteB==CoteC))
            cout << "\n\nLe triangle est isocèle";
        else
            cout << "\n\nLe triangle est scalène";
}

```

ÉCRAN DE SORTIE

```

Donner la longueur du premier côté: 3
Donner la longueur du deuxième côté: 3
Donner la longueur du troisième côté: 3

Les longueurs des trois côtés sont:      3      3      3

Le triangle est équilatéral

```

Donnez deux jeux d'essais qui permettent de compléter les tests.

EXEMPLE 1A

(comme l'exemple précédent mais en utilisant la *sélection multiple*)

LE PROBLÈME:

On lit au clavier la valeur de trois variables: CoteA, CoteB et CoteC. Celles-ci représentent la longueur des trois cotés d'un triangle. Il faut déterminer si le triangle est équilatéral (trois côtés égaux), isocèle (deux côtés égaux) ou scalène (trois côtés différents).

QUE VEUT-ON À LA SORTIE?

Le type du triangle déterminé par les trois longueurs.

QUELLES SONT LES DONNÉES CONNUES?

Trois valeurs qui représentent les longueurs des côtés d'un triangle.

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
CoteA	longueur d'un côté d'un triangle	int	lue à l'entrée
CoteB	longueur d'un côté d'un triangle	int	lue à l'entrée
CoteC	longueur d'un côté d'un triangle	int	lue à l'entrée

LE DIAGRAMME D'ACTION

```
┌ Lire CoteA, CoteB, CoteC
├ Écrire CoteA, CoteB, CoteC
│   ┌ Si CoteA==CoteB et CoteA==CoteC
│   │   Écrire "Le triangle est équilatéral"
│   └ Si CoteA==CoteB ou CoteA==CoteC ou CoteB==CoteC
│     │   Écrire "Le triangle est isocèle"
│     └ Sinon
│       │   Écrire "Le triangle est scalène"
```

PROGRAMME

```

/*****
AUTEUR:  Marcel L'Italien
FICHIER: 3SI8A.C
DESCRIPTION: Lire la longueur des trois cotés d'un triangle, puis
              déterminer si ce triangle est équilatéral, isocèle ou scalène.
              Utilisation de la sélection multiple.
*****/
#include <iostream>

using namespace std;

void main(void)
{
    int CoteA,
        CoteB,
        CoteC;

    cout << "Donner la longueur du premier côté: ";
    cin >> CoteA;
    cout << "Donner la longueur du deuxième côté: ";
    cin >> CoteB;
    cout << "Donner la longueur du troisième côté: ";
    cin >> CoteC;
    cout << "\nLes longueurs des trois côtés sont:\t"
         << CoteA << "\t" << CoteB << "\t" << CoteC;

    if ((CoteA==CoteB) && (CoteA==CoteC))
        cout << "\n\nLe triangle est équilatéral";
    else if ((CoteA==CoteB) || (CoteA==CoteC) || (CoteB==CoteC))
        cout << "\n\nLe triangle est isocèle";
    else
        cout << "\n\nLe triangle est scalène";
}

```

ÉCRAN DE SORTIE

```

Donner la longueur du premier côté: 3
Donner la longueur du deuxième côté: 3
Donner la longueur du troisième côté: 3

Les longueurs des trois côtés sont:      3      3      3

Le triangle est équilatéral

```

Donner deux jeux d'essais qui permettent de compléter les tests.

3.10 L'INSTRUCTION SWITCH().

L'instruction *switch()* sert à écrire, en C++, une *sélection multiple*. Cette instruction est donc similaire à une structure de plusieurs *if()-else* imbriqués. Toutefois, puisqu'un *switch()* est plus lisible, ce sera l'instruction à **privilégier** lors de l'utilisation de la *sélection multiple* dans nos programmes. Cependant il existe des cas où il ne sera pas possible de l'utiliser, et alors on utilisera la méthode des *if()-else* imbriqués.

L'instruction *switch()* évalue une expression simple qui peut prendre plusieurs valeurs (pas seulement deux comme la condition d'un *if()*), et elle sélectionnera une action différente pour chaque valeur. L'expression simple est inscrite à la suite du mot "switch" entre parenthèses. Le résultat de l'expression doit être un entier (types **char**, **int**, ...).

exemples: <i>switch</i> (Carac)	où <i>Carac</i> est de type char
<i>switch</i> (Nombre)	où <i>Nombre</i> est de type int
<i>switch</i> (Nombre*ValA)	où <i>Nombre</i> et <i>ValA</i> sont de type int

L'expression est suivie d'un bloc d'instructions entre accolades. Ce bloc est constitué de plusieurs instructions qui représentent les différentes tâches à exécuter. Chaque alternative débute par une étiquette constituée du mot **case** suivi d'une seule constante (du type de l'expression) et d'un deux-points (:).

exemples: case 'b':
 case 5:
 case 10:

Une seule des alternatives sera choisie, soit celle dont la constante est le résultat de l'expression. S'il n'y a aucune constante égale au résultat de l'expression, alors l'alternative choisie sera celle de la clause **default:** (si présente).

À la suite de chaque étiquette on retrouve le bloc-instructions à exécuter. Il peut y avoir plus d'une étiquette pour une même alternative (bloc-instructions). Chaque alternative se termine par l'énoncé **break** qui provoque la sortie immédiate de l'instruction *switch()*, et empêche ainsi l'exécution des instructions des autres alternatives. Si l'énoncé **break** était omis, les instructions des **case** suivants seraient exécutées, et ce jusqu'à ce qu'on rencontre un **break** (ou la fin du *switch()*).

syntaxe (sans *default*):

```
switch(expression simple)
{
    case constante1: énoncé(s);
                    break;
    case constante2: énoncé(s);
                    break;
    case constante3: énoncé(s);
}
```

diagramme:

```
—si expression==constante1
  énoncé(s)
—si expression==constante2
  énoncé(s)
—si expression==constante3
  énoncé(s)
```

N.B. Le seul opérateur relationnel possible est `==` pour pouvoir utiliser l'énoncé *switch()*. Dans les autres cas on utilisera des *if()-else* imbriqués (voir page 87).

syntaxe (avec *default*):

```
switch(expression simple)
{
    case constante1: énoncé(s);
                    break;
    case constante2: énoncé(s);
                    break;
    default:        énoncé(s);
}
```

diagramme:

```
—si expression==constante1
  énoncé(s)
—si expression==constante2
  énoncé(s)
—sinon
  énoncé(s)
```

N.B. Il peut y avoir autant de **case** que désiré. L'étiquette **default** est facultative.
Les énoncés d'un **case** ne sont pas écrits entre accolades ({ }).

exemples:

```
switch(Nombre)
{
    case 1:cout << "Vous m'avez donné 1 sous";
            break;

    case 5:cout << "Vous m'avez donné 5 sous";
            break;

    case 10:cout << "Vous m'avez donné 10 sous";
            break;

    case 25:cout << "Vous m'avez donné 25";
            break;

    default:cout << "Pièce inconnue";
}
```

```
switch(Nombre)
{
    case 1:

    case 5:cout << "Ce montant est insuffisant";
            break;

    case 10:cout << "Vous m'avez donné dix sous";
            cout << "\nVoici votre crayon à mine";
            break;

    case 25:cout << "Vous m'avez donné 25 sous";
            cout << "\nVoici votre crayon rouge";
            break;

    default:cout << "Pièce inconnue";
}
```

diagrammes d'action correspondant:

```
graph TD
    A[Si Nombre==1] --> B[Écrire "Vous m'avez donné 1 sou"]
    B --> C[Si Nombre==5]
    C --> D[Écrire "Vous m'avez donné 5 sous"]
    D --> E[Si Nombre==10]
    E --> F[Écrire "Vous m'avez donné 10 sous"]
    F --> G[Si Nombre==25]
    G --> H[Écrire "Vous m'avez donné 25 sous"]
    H --> I[Sinon]
    I --> J[Écrire "Pièce inconnue"]
```

```
graph TD
    A["Si Nombre==1 ou Nombre==5"] --> B[Écrire "Ce montant est insuffisant"]
    B --> C[Si nombre==10]
    C --> D[Écrire "Vous m'avez donné 10 sous"]
    D --> E[Écrire "Voici votre crayon à mine"]
    E --> F[Si Nombre==25]
    F --> G[Écrire "Vous m'avez donné 25 sous"]
    G --> H[Écrire "Voici votre crayon rouge"]
    H --> I[Sinon]
    I --> J[Écrire "Pièce inconnue"]
```

EXEMPLE 1

LE PROBLÈME:

On veut construire un calculateur simple qui permettra d'exécuter les quatre opérations mathématiques de base (+, -, *, /).

QUE VEUT-ON À LA SORTIE?

L'opération arithmétique (opérateur et opérandes) et son résultat.

QUELLES SONT LES DONNÉES CONNUES?

Deux nombres et l'opérateur arithmétique.

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
NbA	le premier nombre	float	lue à l'entrée
NbB	le deuxième nombre	float	lue à l'entrée
Operateur	opérateur arithmétique	char	lue à l'entrée
Reponse	réponse de l'opération	float	NbA Operateur NbB

LE DIAGRAMME D'ACTION:

```
┌ Lire Operateur
├ Lire NbA, NbB
│   ┌ Si Operateur == '+'
│   │ Reponse=NbA+NbB
│   └ Si Operateur == '-'
│   │ Reponse=NbA-NbB
│   └ Si Operateur == '*'
│   │ Reponse=NbA*NbB
│   └ Si Operateur == '/'
│   │ Reponse=NbA/NbB
└ Écrire NbA, Operateur, NbB, '=', Reponse
```

PROGRAMME

```

/*****
AUTEUR:  Marcel L'Italien
FICHIER: 3si9.cpp
DESCRIPTION: Lecture d'un opérateur et de ses opérandes, puis exécution
             de l'opération arithmétique choisie
*****/
#include <iostream>
#include <iomanip>

using namespace std;

void main(void)
{
    float NbA,
          NbB,
          Reponse;
    char  Operateur;

    /* Lecture de l'opérateur arithmétique. */
    cout << "Entrer le symbole de l'opération arithmétique: ";
    cin >> Operateur;

    /* Lecture des deux nombres réels utilisés dans l'opération. */
    cout<<"\nDonner les 2 nombres faisant partie de l'expression à évaluer: ";
    cin >> NbA >> NbB;

    /* Sélection de l'opération et calcul du résultat */
    switch (Operateur)
    {
        case '+': Reponse = NbA + NbB;
                  break;

        case '-': Reponse = NbA - NbB;
                  break;

        case '*': Reponse = NbA * NbB;
                  break;

        case '/': Reponse = NbA / NbB;
    }

    cout << fixed << setprecision(2) << "\n\n" << NbA
         << " " << Operateur << " " << NbB << " = " << Reponse;
}

```

ÉCRAN DE SORTIE (avec l'opérateur *):

```

Entrer le symbole de l'opération arithmétique: *
Donner les 2 nombres faisant partie de l'expression à évaluer: 2.5  3
2.50 * 3.00 = 7.50

```

Exemple 1a

(comme l'exemple précédent mais en utilisant la clause **default**: pour valider l'opérateur)

LE PROBLÈME:

Construire un calculateur simple qui permettra d'exécuter quatre opérations mathématiques(+, -, *, /).

QUE VEUT-ON À LA SORTIE?

L'opération arithmétique (opérateur et opérandes) et son résultat, ou le message «??? Opérateur inconnu».

QUELLES SONT LES DONNÉES CONNUES?

Deux nombres et l'opérateur arithmétique.

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
MessageErreur	message d'erreur qui sera affiché	string	"" ou "??? Opérateur inconnu"
NbA	le premier nombre	float	lue à l'entrée
NbB	le deuxième nombre	float	lue à l'entrée
Operateur	opérateur arithmétique	char	lue à l'entrée
Reponse	réponse de l'opération	float	NbA Operateur NbB

LE DIAGRAMME D'ACTION:

```
MessageErreur=""
Lire Operateur
Lire NbA,NbB
  Si Operateur == '+'
    Reponse=NbA+NbB
  Si Operateur == '-'
    Reponse=NbA-NbB
  Si Operateur == '*'
    Reponse=NbA*NbB
  Si Operateur == '/'
    Reponse=NbA/NbB
  Sinon
    MessageErreur = "??? Opérateur inconnu"
Écrire NbA, Operateur, NbB, '='
  si MessageErreur == ""
    Écrire Reponse
  sinon
    Écrire MessageErreur
```

PROGRAMME

```

/*****
AUTEUR:  Marcel L'Italien
FICHIER: 3si9a.cpp
DESCRIPTION: Lecture d'un opérateur et de ses opérandes,
              validation de l'opérateur et exécution de l'opération choisie
*****/
#include <iostream>
#include <iomanip>
#include <string> // Pour la classe string

using namespace std;

void main(void)
{
    float NbA,
          NbB,
          Reponse;
    char  Operateur;
    string MessageErreur = "";

    /* Lecture de l'opérateur arithmétique. */
    cout << "Entrer le symbole de l'opération arithmétique: ";
    cin >> Operateur;

    /* Lecture des deux nombres réels utilisés dans l'opération. */
    cout<<"\nDonner les 2 nombres faisant partie de l'expression à évaluer: ";
    cin >> NbA >> NbB;

    /* Sélection de l'opération et calcul du résultat */
    switch (Operateur)
    {
        case '+': Reponse = NbA + NbB;
                  break;
        case '-': Reponse = NbA - NbB;
                  break;
        case '*': Reponse = NbA * NbB;
                  break;
        case '/': Reponse = NbA / NbB;
                  break;
        default : MessageErreur = "??? Opérateur inconnu";
    }

    cout << fixed << setprecision(2) << "\n\n" << NbA
         << " " << Operateur << " " << NbB << " = ";

    if (MessageErreur == "")
        cout << Reponse;
    else
        cout << MessageErreur;
}

```

ÉCRAN DE SORTIE (opérateur inconnu):

Entrer le symbole de l'opération arithmétique: >

Donner les deux nombres faisant partie de l'expression à évaluer: 2.5 3.2

2.50 > 3.20 = ??? Opérateur inconnu

ÉCRAN DE SORTIE (avec l'opérateur *):

Entrer le symbole de l'opération arithmétique: *

Donner les deux nombres faisant partie de l'expression à évaluer: 2.5 3

2.50 * 3.00 = 7.50

EXEMPLE 2:

LE PROBLÈME:

On veut connaître le nom d'une couleur selon un code. Les couleurs acceptées sont celles d'un feu de circulation.

QUE VEUT-ON À LA SORTIE?

Le nom d'une couleur.

QUELLES SONT LES DONNÉES CONNUES?

Le code d'une couleur

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
Code	code d'une couleur	char	lue à l'entrée

LE DIAGRAMME D'ACTION:

```
LireCode
┌ SiCode == 'r' || Code == 'R'
│   Écrire "Le feu est rouge"
├ Si Code == 'j' || Code == 'J'
│   Écrire "Le feu est jaune"
├ Si Code == 'v' || Code == 'V'
│   Écrire "Le feu est vert"
├ Sinon
│   Écrire "Code inconnu"
```

PROGRAMME (première version: utilisation de *switch()*)

```

/*****
AUTEUR:  Marcel L'Italien
FICHIER: 3sil0.cpp
DESCRIPTION: Lecture d'un code de couleur pour un feu de circulation,
             puis affichage du nom de cette couleur
*****/
#include <iostream>

using namespace std;

void main(void)
{
    char Code;

    /* Lecture du code de couleur */
    cout << "Entrer le code de la couleur: ";
    cin >> Code;

    switch (Code)
    {
        case 'R':
        case 'r': cout << "\n\nLe feu est rouge";
                  break;

        case 'J':
        case 'j': cout << "\n\nLe feu est jaune";
                  break;

        case 'V':
        case 'v': cout << "\n\nLe feu est vert";
                  break;

        default:  cout << "\n\nCode inconnu";
    }
}

```

ÉCRAN DE SORTIE

Entrer le code de la couleur: r

Le feu est rouge

PROGRAMME (seconde version: pas d'utilisation de *switch()*)

```

/*****
AUTEUR:  Marcel L'Italien
FICHIER: 3sil0a.cpp
DESCRIPTION: Lecture d'un code de couleur pour un feu de circulation,
             puis affichage du nom de cette couleur
*****/
#include <iostream>

using namespace std;

void main(void)
{
    char Code;

    /* Lecture du code de couleur */
    cout << "Entrer le code de la couleur: ";
    cin >> Code;

    if (Code=='R' || Code=='r')
        cout << "\n\nLe feu est rouge";
    else if (Code=='J' || Code=='j')
        cout << "\n\nLe feu est jaune";
    else if (Code=='V' || Code=='v')
        cout << "\n\nLe feu est vert";
    else
        cout << "\n\nCode inconnu";
}

```

ÉCRAN DE SORTIE

Entrer le code de la couleur: z

Code inconnu

3.11 EXERCICES

- 1 Dans un énoncé **if()** sans le mot-clé **else**, qu'arrive-t-il si la condition dans le **if()** est fausse?
 - a) le programme cherche le dernier **else** du code pour exécuter les énoncés qui le suivent
 - b) le programme s'arrête
 - c) le contrôle passe à l'énoncé suivant le corps du **if()**
 - d) les énoncés du corps du **if()** sont exécutés
- 2 Les énoncés suivant le **else** d'un énoncé **if()-else** sont exécutés quand:
 - a) la condition suivant le **if()** est fausse
 - b) la condition suivant le **if()** est vraie
 - c) la condition suivant le **else** est fausse
 - d) la condition suivant le **else** est vraie
- 3 Les énoncés suivant le **if()** d'une forme **...else-if()** sont exécutés lorsque:
 - a) la condition suivant le **else-if()** est vraie et que toutes les conditions précédentes sont vraies
 - b) la condition suivant le **else-if()** est vraie et que toutes les conditions précédentes sont fausses
 - c) la condition suivant le **else-if()** est fausse et que toutes les conditions précédentes sont vraies
 - d) la condition suivant le **else-if()** est fausse et que toutes les conditions précédentes sont fausses
- 4 Un **else** est toujours rattaché au:
 - a) dernier **if()** qui est au même niveau d'indentation
 - b) dernier **if()** qui est au même niveau d'imbrication
 - b) dernier **if()** qui n'est pas rattaché à un **else**
 - c) dernier **if()** qui n'est pas entre accolades
 - d) dernier **if()** qui n'est pas entre accolades et qui n'est pas rattaché à un **else**
- 5 Est-il possible d'associer plusieurs étiquettes **case** à un même groupe d'instructions?
- 6 Que se passe-t-il lorsque l'expression contenue dans une instruction **switch()** prend la même valeur que l'une des étiquettes **case**? Que se passe-t-il si elle ne prend la valeur d'aucune des étiquettes **case**.
- 7 A quoi sert l'étiquette **default** dans une instruction **switch()**, et à quelles règles de syntaxe obéit-elle?
- 8 Quel est le rôle de l'instruction **break**?

- 9 Vrai ou faux: un énoncé **break** doit nécessairement être le dernier énoncé d'un **case**, si on désire que seule cette possibilité d'un **switch()** soit exécutée.

- 10 L'énoncé **switch()** suivant fonctionnera-t-il?

```
switch(temp)
{
    case temp<10:cout << "La température est froide";
                  break;
    case temp<30:cout << "Quelle belle température";
                  break;
    default      :      cout << "Il fait trop chaud";
}
```

- 11 Écrire un programme qui lira la variable **Devinette** (type int), puis qui affichera un des messages suivants (selon la valeur lue):

"CHAUD"	si Devinette vaut 1
"TIEDE"	si Devinette vaut 2
"FROID"	si Devinette vaut 3
"valeur inconnue"	dans les autres cas

Utilisez un *switch()*.

12. Compléter l'analyse et coder le problème qui suit:

LE PROBLÈME:

Lire trois nombres: Nb1, Nb2 et Nb3. Ces nombres doivent être placés en ordre décroissant. Le plus grand doit être rangé dans la variable Nb1 et le plus petit dans la variable Nb3.

QUE VEUT-ON À LA SORTIE?

Afficher trois nombres en ordre décroissant.

QUELLES SONT LES DONNÉES CONNUES?

Trois nombres lus au clavier.

LES ÉCRANS:

```
Donner le premier nombre (Nb1): xxxx
Donner le deuxième nombre (Nb2): xxxx
Donner le troisième nombre (Nb3): xxxx

Les nombres en ordre décroissant sont:
          xxxx      xxxx      xxxx
```

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
Echange	variable de travail pour permettre l'échange de valeurs entre deux variables	int	Nb1, Nb2 ou Nb3
Nb1	le premier nombre	int	lue à l'entrée
Nb2	le deuxième nombre	int	lue à l'entrée
Nb3	le troisième nombre	int	lue à l'entrée

LE DIAGRAMME D'ACTION:

à compléter

LE PROGRAMME:

à compléter

13. Compléter l'analyse et coder le problème qui suit:

LE PROBLÈME:

Lire, au clavier, un nombre entier et déterminer s'il est positif, négatif ou nul.

QUE VEUT-ON À LA SORTIE?

On veut la caractéristique d'un entier (positif, négatif ou nul).

QUELLES SONT LES DONNÉES CONNUES?

Un nombre lu au clavier.

LES ÉCRANS:

Donner votre nombre: xxxx

xxxx est un nombre aaaaa

(où aaaaa sera remplacé par «positif», «négatif» ou «nul»)

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
Nombre	nombre à traiter	int	lue à l'entrée

LE DIAGRAMME D'ACTION:

à compléter

LE PROGRAMME:

à compléter

14. Coder le problème qui suit

LE PROBLÈME

On veut donner des messages d'avertissement à propos de la température d'une bouilloire. Si la température est plus petite que 750°, la température est normale. Entre 750° et 900° la bouilloire surchauffe. Au-dessus de 900° il faut entamer une procédure d'urgence.

QUE VEUT-ON À LA SORTIE?

Un message d'avertissement.

QUELLES SONT LES DONNÉES CONNUES?

La température de la bouilloire, lue au clavier.

LES ÉCRANS:

Donner la température de la bouilloire: xxxx

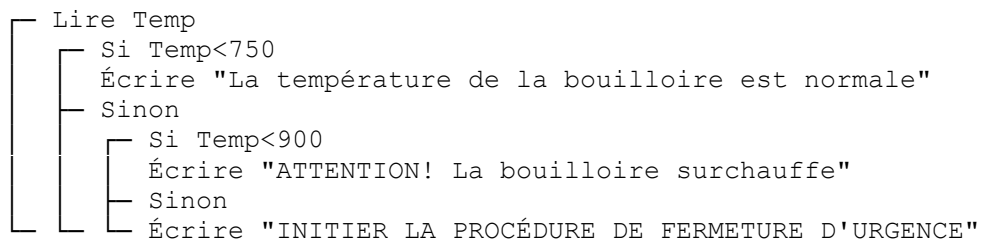
mmmmmmmmmmmm

(où mmmmmmmmmmm représente le message approprié)

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
Temp	température de la bouilloire	int	lue à l'entrée

LE DIAGRAMME D'ACTION:



LE PROGRAMME:

à compléter

15. Modifiez le diagramme d'action du problème précédent (en utilisant la *sélection multiple*), puis codez-le.

16. Coder le problème qui suit:

LE PROBLÈME

On doit afficher une cote correspondante à la note d'un étudiant

Note	Cote
90 à 100	A
80 à 89	B
70 à 79	C
60 à 69	D
0 à 59	E

QUE VEUT-ON À LA SORTIE?

La note de l'étudiant suivie de la cote correspondante

QUELLES SONT LES DONNÉES CONNUES?

La note, lue au clavier.

LES ÉCRANS:

Tapez une note (0 à 100): **xxx**

La note **xxx** correspond à la cote **c**

(où **xxx** représente la note et **c** la cote appropriée)

LE DICTIONNAIRE DE DONNÉES:

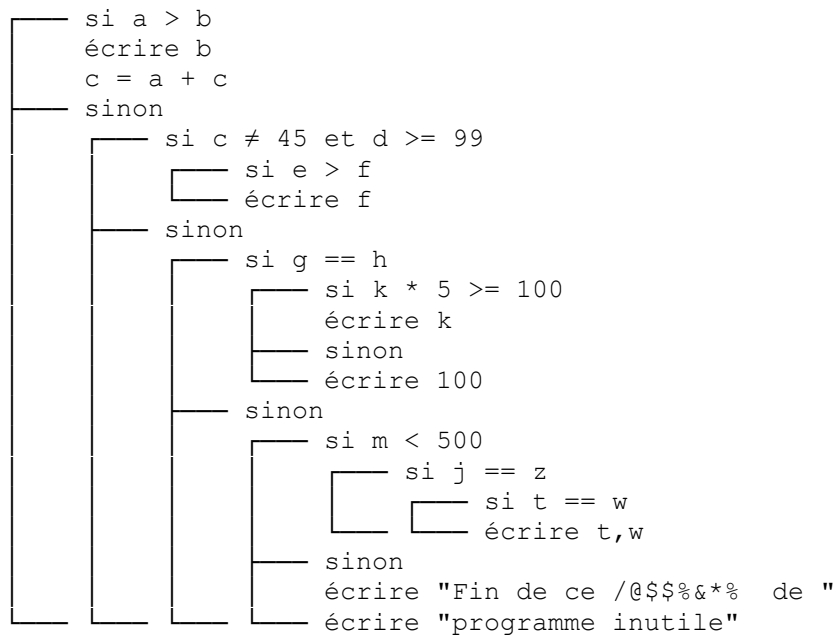
Identificateur	Signification	Type	Valeur
Cote	cote correspondante à une note	char	A, B, C, D ou E
Note	note d'un étudiant	int	lue à l'entrée

LE DIAGRAMME D'ACTION:

```
┌ Lire Note
├   ┌ Si Note >= 90
│   │ Cote='A'
│   └ Si Note >= 80
│     │ Cote='B'
│     └ Si Note >= 70
│       │ Cote='C'
│       └ Si Note >= 60
│         │ Cote='D'
│         └ Sinon
│           │ Cote='E'
└   Écrire Note, Cote
```

17. Indiquer le résultat affiché par le diagramme suivant, et ce pour chaque série de valeurs (faire la trace):

série	variables													
	a	b	c	d	e	f	g	h	j	k	m	t	w	z
1	4	3	48	99	1	2	10	10	5	5	400	9	9	5
2	3	4	48	99	1	2	10	10	5	5	400	9	9	5
3	3	4	45	99	1	2	10	10	5	5	400	9	9	5
4	3	4	48	99	2	1	10	10	5	5	400	9	9	5
5	3	4	45	99	2	1	10	10	5	5	400	9	19	5
6	3	4	45	99	2	1	10	10	5	20	400	9	19	5
7	3	4	45	99	2	1	10	7	5	20	400	9	19	5
8	3	4	45	99	2	1	10	7	5	5	600	9	19	5



18. Coder le diagramme de l'exercice précédent:
- mettre des accolades ({ }) **partout** (pour chaque *if()* et chaque *else*)
 - mettre le **minimum** d' { }

19. · Comparer ce diagramme d'action à celui de l'exercice #17. Est-il équivalent?

· Coder ce diagramme d'action (est-ce possible avec **switch()**?)

```
┌ si a > b
│ écrire b
│ c = a + c
├ si c != 45 et d >= 99
│   ┌ si e > f
│   │ écrire f
│   └ si g == h
│       ┌ si k * 5 >= 100
│       │ écrire k
│       └ sinon
│           écrire 100
├ si m < 500
│   ┌ si j == z
│   │   ┌ si t == w
│   │   │ écrire t,w
│   └ sinon
│       écrire "Fin de ce /@$$%&*% de "
└ écrire "programme inutile"
```

CHAPITRE 4

LES INSTRUCTIONS DE RÉPÉTITIONS

4.1 LA RÉPÉTITION

Jusqu'à maintenant nous avons résolu des problèmes à l'aide d'instructions en séquence (affectation, écriture, lecture, ...) et d'instructions de sélection (sélection simple et sélection multiple).

Une des forces des ordinateurs réside dans leur très grande vitesse. Ils peuvent, entre autres, répéter les mêmes opérations plusieurs fois, et ce très rapidement. Voilà une catégorie d'opérations que nous n'avons pas abordée, les instructions de répétitions. Une fois que vous connaîtrez ces instructions, vous aurez entre les mains les éléments fondamentaux permettant la résolution de la majorité des problèmes informatiques.

Pour bien répondre aux besoins du programmeur, nous utiliserons trois types d'instructions de répétition: le **Pour**, le **Tant Que** et le **Faire...Tant Que**.

4.2 LES RÉPÉTITIONS DANS LES DIAGRAMMES D'ACTION

Les répétitions, dans les diagrammes d'action, s'écrivent selon les règles suivantes:

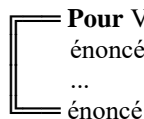
règle 1: On doit choisir l'énoncé de répétition selon les critères suivants:

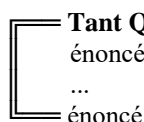
- a) pour répéter un bloc d'instructions un nombre connu et exact de fois, on utilisera un **Pour**.
- b) pour répéter un bloc d'instructions un nombre inconnu de fois, on utilisera un **Tant Que**.
- c) pour exécuter un bloc d'instructions un nombre inconnu de fois, mais au moins une fois, on utilisera un **Faire...Tant Que**.


règle 2: Les énoncés de répétition peuvent être imbriqués, mais non se chevaucher.

ATTENTION: Lorsqu'on imbrique des **Pour**, il faut que chacun ait sa propre variable de contrôle.

règle 3: Le double crochet signale le début et la fin du bloc d'instructions à répéter.

 **Pour** Variable = valeur initiale, condition pour poursuivre, incrément
énoncé
...
énoncé

 **Tant Que** condition pour poursuivre
énoncé
...
énoncé

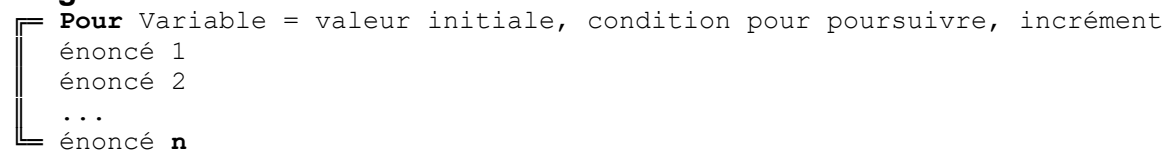
 **Faire**
énoncé
...
Tant Que condition pour poursuivre

4.3 L'ÉNONCÉ POUR

Souvent on voudra répéter un bloc d'instructions un nombre prédéterminé de fois. L'outil à utiliser pour le faire est l'énoncé de répétition **Pour**.

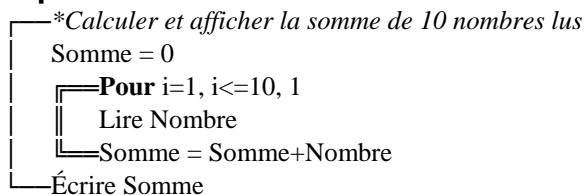
L'énoncé **Pour** doit être utilisé seulement lorsqu'on connaît, au début de son exécution, le nombre exact de fois où la répétition sera exécutée.

diagramme:



Il y aura autant d'énoncés que désiré (de 0 à n)

exemple:



La lecture du nombre (Lire Nombre) et la somme (Somme = Somme+Nombre) sont répétées 10 fois. Afin de le faire, on utilise une variable de contrôle, ici c'est **i**, qui permettra de compter de 1 jusqu'à 10, et ce par la valeur de l'incrément, soit **1**. La variable de contrôle est une variable entière qui devra être définie comme toute autre variable.

On doit définir la variable contrôle de la boucle au même titre que toutes les autres variables utilisées dans le programme.

Un **Pour** est répété tant que sa condition est vraie, c'est pourquoi il s'agit d'une condition pour poursuivre et non pas d'une condition d'arrêt. Notez que pour arrêter cette répétition, ici la valeur de **i** devra dépasser 10 (ainsi la condition $i \leq 10$ sera fausse). Après le **Pour**, la valeur de **i** est donc de **11**.

4.4 L'INSTRUCTION FOR()

L'énoncé du langage C++ correspondant à la répétition **Pour** est l'instruction *for()*.

syntaxe: for (Variable = valeur initiale; condition pour poursuivre; incrémentation)
 bloc-instructions

exemple: le programme correspondant au diagramme de la section précédente:

```
//Calculer et afficher la somme de 10 nombres lus
Somme = 0;
for(i=1; i<=10; i++)
{
    cout << "\nEntrez un nombre entier: ";
    cin >> Nombre;
    Somme += Nombre;
}
cout << "La somme des nombres est: " << Somme;
```

Notez l'absence du point-virgule après la parenthèse fermante du *for()*. Un point-virgule ici signifierait qu'il n'y a aucun énoncé à répéter (ou que l'énoncé «*vide*» sera répété), car un bloc-instructions peut n'être composé que d'une seule instruction se terminant par un point-virgule, et qu'alors les accolades (`{ }`) sont facultatives. Une telle erreur ne sera pas détectée par le compilateur, car c'est correct d'un point de vue syntaxique. Toutefois, d'un point de vue logique, votre programme ne répétera pas les énoncés prévus.

Voici comment s'exécute un *for()*:

1. Affectation de la valeur initiale à la variable de contrôle.
2. Évaluation de la condition pour poursuivre :
 - si la condition est fausse, on quitte le *for()* (la répétition se **termine** ici)
 - si la condition est vraie, on exécute les instructions du *for()*
(son bloc-instructions)
3. Incrémentation (ou décrémentation) de la variable de contrôle.
4. Retour à l'étape 2..

L'affectation initiale n'a lieu qu'une seule fois. Les étapes suivantes se poursuivent jusqu'à ce que la condition soit fausse. Si elle ne devenait jamais fausse, on aurait alors une *boucle infinie*, soit une répétition sans fin. Le programmeur devra s'assurer que ça n'arrive pas.

Il est aussi possible que le bloc-instructions ne soit pas exécuté une seule fois. Cela se produit quand la condition pour poursuivre est fausse dès le début.

EXEMPLE 1

LE PROBLÈME

Lire une suite de 5 nombres quelconques et calculer leur moyenne.

QUE VEUT-ON À LA SORTIE?

La suite de nombres et la moyenne.

QUELLES SONT LES DONNÉES CONNUES?

Les 5 nombres.

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
CompteurDeNbs	compteur des nombres lus	int	1 à 5
Moyenne	moyenne des nombres	float	Somme/5
Nombre	le dernier nombre lu	float	lue au clavier
Somme	somme des nombres	float	0 au début puis Somme+Nombre

LE DIAGRAMME D'ACTION:

```
— Somme = 0
┌
├   Pour CompteurDeNbs=1, CompteurDeNbs<=5, 1
├   ┌ Lire Nombre
├   └ Somme = Somme+Nombre
└ Moyenne = Somme/5
  Écrire Moyenne
```

REMARQUES

La variable de contrôle de la répétition (CompteurDeNbs) peut être utilisée de différentes façons pour permettre la répétition du bloc-instructions 5 fois. On pourrait, par exemple, compter:

de 1 à 5 par incrément de 1

de 0 à 4 par incrément de 1

de 5 à 1 par incrément de -1

de 4 à 0 par incrément de -1

LE PROGRAMME

```

/*****
AUTEUR: Marcel L'Italien
NOM DU PROGRAMME: 4PO1.CPP
DESCRIPTION: Lecture et impression d'une suite de 5 nombres, calcul
             de la moyenne de la suite des nombres.
*****/
#include <iostream>
#include <iomanip>

using namespace std;

void main(void)
{
    int CompteurDeNbs;
    float Moyenne, Nombre, Somme;

    Somme=0;

    // Spécifier la notation avec décimale pour les valeurs réelles
    cout << fixed;

    // Boucle faisant la lecture, la somme et l'impression des nombres.
    for (CompteurDeNbs=1; CompteurDeNbs<=5; CompteurDeNbs++)
    {
        cout << "Le nombre no. " << CompteurDeNbs << " : ";
        cin >> Nombre;
        Somme+=Nombre;
    }

    // Calcul de la moyenne et impression du résultat.
    Moyenne=Somme/5;

    cout << "\n\nLa moyenne est de "
         << setw(10) << setprecision(2) << Moyenne << "\n\n" ;
}

```

LE RÉSULTAT

```

Le nombre no. 1 : 9.5
Le nombre no. 2 : 7.3
Le nombre no. 3 : 5.6
Le nombre no. 4 : 7.8
Le nombre no. 5 : 8.4

La moyenne est de      7.72

```


EXEMPLE 2

LE PROBLÈME

Afficher la moyenne de «N» nombres lus au clavier.

QUE VEUT-ON À LA SORTIE?

La moyenne.

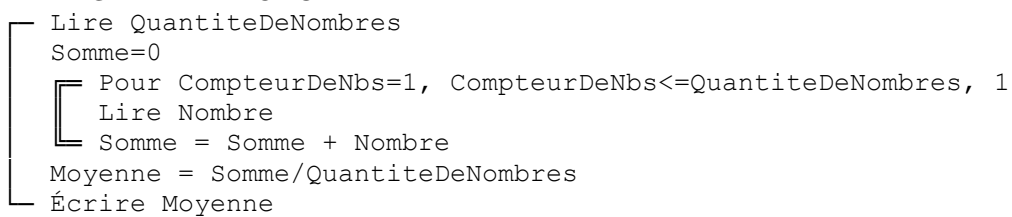
QUELLES SONT LES DONNÉES CONNUES?

Le nombre de valeurs lues et les valeurs.

LE DICTIONNAIRE DE DONNÉES

Identificateur	Signification	Type	Valeur
CompteurDeNbs	compte le nombre de valeurs lues	int	1 à QuantiteDeNombres
Moyenne	moyenne des valeurs lues	float	Somme/NbDeLecture
QuantiteDeNombres	nombre de valeurs à lire	int	Lue au clavier
Somme	somme des valeurs lues	float	0 et Somme+Nombre
Nombre	chaque nombre à lire	float	Lue au clavier

LE DIAGRAMME D'ACTION



LE PROGRAMME

```

/*****
AUTEUR: Marcel L'Italien
NOM DU PROGRAMME: 4PO1.CPP
DESCRIPTION: Lecture et impression d'une suite de N nombres, calcul
             de la moyenne de la suite des nombres.
*****/
#include <iostream>
#include <iomanip>

using namespace std;

void main(void)
{
    int CompteurDeNbs, QuantiteDeNombres;
    float Moyenne, Nombre, Somme;

    cout << "Combien de nombre a saisir ?";
    cin >> QuantiteDeNombre;

    Somme=0;

    // Boucle faisant la lecture, la somme et l'impression des nombres.
    for (CompteurDeNbs=1; CompteurDeNbs<=QuantiteDeNombres; CompteurDeNbs++)
    {
        cout << "Le nombre no. " << CompteurDeNbs << " : ";
        cin >> Nombre;
        Somme += Nombre;
    }

    // Calcul de la moyenne et impression du resultat.
    Moyenne=Somme/QuantiteDeNombre;

    cout << "\n\nLa moyenne est de " << setprecision(8) << Moyenne;
}

```

LE RÉSULTAT

```

Combien de nombre a saisir ? 3
Le nombre no. 1 : 3.5
Le nombre no. 2 : 2.8
Le nombre no. 3 : 4.9

La moyenne est de 3.7333336

```

EXEMPLE 3

LE PROBLÈME

Afficher la valeur décimale, caractère, octale et hexadécimale des codes ASCII de 33 à 101.

QUE VEUT-ON À LA SORTIE?

La valeur décimale, caractère, octale et hexadécimal des codes ASCII de 33 à 101, et ce sur 3 colonnes de résultats (Colonnes: 1, 31 et 61).

QUELLES SONT LES DONNÉES CONNUES?

Aucune.

LE DICTIONNAIRE DE DONNÉES

Identificateur	Signification	Type	Valeur
Colonne	Colonne à laquelle on affiche un code ascii	int	1, 31 et 61
Ligne	Ligne à laquelle on affiche un code ascii	int	3 à 23
CodeAsciiDebut	Variable de contrôle représentant le premier code ascii au début de chaque ligne	int	33 à 99 par 3
CodeAscii	chacun des 3 codes ASCII d'une ligne	int	CodeAsciiDebut à CodeAsciiDebut + 3

LE DIAGRAMME D'ACTION

```
Ligne=3
┌ Pour CodeAsciiDebut=33, CodeAsciiDebut<=99, 3
│   Colonne=1
│   ┌ Pour CodeAscii = CodeAsciiDebut, CodeAscii < CodeAsciiDebut+3, 1
│   │   gotoxy(Colonne,Ligne)
│   │   Écrire CodeAscii, CodeAscii, CodeAscii, CodeAscii
│   │   Colonne=Colonne+30
│   └ Ligne=Ligne+1
└
```

LE PROGRAMME

```

/*****
NOM DU PROGRAMME: 4PO3.CPP
DESCRIPTION: Ce programme permet d'imprimer les codes ASCII de 33 à 101
*****/
#include <iostream>
#include <iomanip>
#include "c:\\cvm.h"
using namespace std;
void main(void)
{   int   CodeAsciiDebut, CodeAscii;
    int   Colonne, Ligne;
    cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n";
    gotoxy(0,0);
    cout << "  DEC      CAR OCT HEX          DEC      CAR OCT HEX          DEC      CAR OCT HEX";
    Ligne=2;
    for (CodeAsciiDebut=33; CodeAsciiDebut<=99; CodeAsciiDebut+=3)
    {   Colonne=0;
        for (CodeAscii=CodeAsciiDebut; CodeAscii<CodeAsciiDebut+3; CodeAscii++)
        {
            gotoxy(Colonne, Ligne);
            cout << dec << setw(3) << CodeAscii << ": "
                << setw(4) << (char)CodeAscii
                << oct << setw(4) << CodeAscii
                << hex << setw(4) << CodeAscii;
            Colonne += 30;
        }
        Ligne++;
    }
}

```

LE RÉSULTAT

DEC	CAR	OCT	HEX	DEC	CAR	OCT	HEX	DEC	CAR	OCT	HEX
33:	!	41	21	34:	"	42	22	35:	#	43	23
36:	\$	44	24	37:	%	45	25	38:	&	46	26
39:	'	47	27	40:	(50	28	41:)	51	29
42:	*	52	2a	43:	+	53	2b	44:	,	54	2c
45:	-	55	2d	46:	.	56	2e	47:	/	57	2f
48:	0	60	30	49:	1	61	31	50:	2	62	32
51:	3	63	33	52:	4	64	34	53:	5	65	35
54:	6	66	36	55:	7	67	37	56:	8	70	38
57:	9	71	39	58:	:	72	3a	59:	;	73	3b
60:	<	74	3c	61:	=	75	3d	62:	>	76	3e
63:	?	77	3f	64:	@	100	40	65:	A	101	41
66:	B	102	42	67:	C	103	43	68:	D	104	44
69:	E	105	45	70:	F	106	46	71:	G	107	47
72:	H	110	48	73:	I	111	49	74:	J	112	4a
75:	K	113	4b	76:	L	114	4c	77:	M	115	4d
78:	N	116	4e	79:	O	117	4f	80:	P	120	50
81:	Q	121	51	82:	R	122	52	83:	S	123	53
84:	T	124	54	85:	U	125	55	86:	V	126	56
87:	W	127	57	88:	X	130	58	89:	Y	131	59
90:	Z	132	5a	91:	[133	5b	92:	\	134	5c
93:]	135	5d	94:	^	136	5e	95:		137	5f
96:		140	60	97:	a	141	61	98:	␣	142	62
99:	c	143	63	100:	d	144	64	101:	e	145	65

Version 2 - avec si

```

┌ Ligne=1
├ ┌ Pour CodeAscii=33, CodeAscii<=101, 1
│ │ ┌ Si CodeAscii%3 == 0
│ │ │ Colonne=0
│ │ └ Ligne=Ligne+1
│ │ gotoxy(Colonne, Ligne)
│ │ Écrire ValeurAscii,ValeurAscii,ValeurAscii,ValeurAscii
│ └ Colonne=Colonne+30
└

```

LE PROGRAMME

```

/*****
AUTEUR: Marcel L'Italien
NOM DU PROGRAMME: 4PO3-SI.CPP
DESCRIPTION: Même programme que le précédent, mais ici il n'y a pas
             de boucles imbriquées (pas de for() dans un autre).
*****/
#include <iostream>
#include <iomanip>
#include "c:\\cvm.h"
using namespace std;

void main(void)
{
    int CodeAscii;
    int Colonne, Ligne;

    cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n";
    gotoxy(0,0);
    cout << " DEC      CAR OCT HEX                "
         << " DEC      CAR OCT HEX                DEC      CAR OCT HEX";
    Ligne=1;
    for (CodeAscii=33; CodeAscii<=101; CodeAscii++)
    {
        if(CodeAscii%3 == 0)
        {
            Ligne++;
            Colonne = 0;
        }

        gotoxy(Colonne, Ligne);
        cout << dec << setw(3) << CodeAscii << ": "
             << setw(4) << (char)CodeAscii
             << oct << setw(4) << CodeAscii
             << hex << setw(4) << CodeAscii;
        Colonne += 30;
    }
}

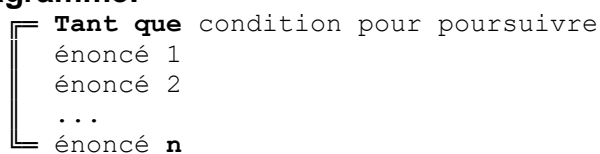
```

4.5 L'ÉNONCÉ TANT QUE

Parfois nous aurons à répéter un bloc d'instructions, mais nous ne connaissons pas le nombre exact de fois. Nous ne pourrions donc pas utiliser un **Pour**. Dans ce cas, nous utiliserons l'énoncé de répétition **Tant que**.

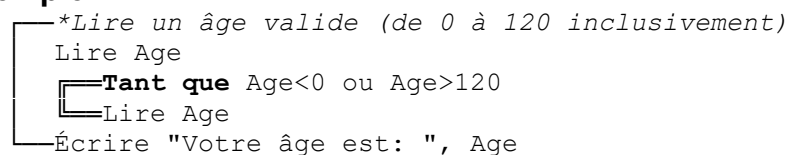
L'énoncé **Tant que** peut être utilisé lorsqu'on ne connaît pas le nombre exact de fois où la répétition sera exécutée.

diagramme:



Il y aura autant d'énoncés que désiré (de 0 à n)

exemple:



Ici la lecture de l'âge est répétée jusqu'à ce que l'âge soit valide (compris entre 0 et 120 inclusivement). Puisqu'on ne sait pas combien de tentatives il faudra à l'utilisateur pour entrer une valeur correcte, on ne sait pas combien de fois la répétition se fera. Notez que si la première lecture (avant le **Tant que**) est correcte, les énoncés du **Tant que** ne sont pas exécutés.

Un **Tant que** est répété tant que sa condition est vraie, c'est pourquoi il s'agit d'une condition pour poursuivre, et non pas d'une condition d'arrêt. Notez que pour arrêter cette répétition, ici la variable **Age** devra avoir une valeur comprise entre 0 et 120 inclusivement (ainsi la condition *Age<0 ou Age>120* sera fausse). Après l'exécution de ce **Tant que**, la valeur de **Age** est donc valide.

4.6 L'INSTRUCTION WHILE()

L'énoncé du langage C++ correspondant à la répétition **Tant que** est l'instruction *while()*.

syntaxe: while (condition pour poursuivre)
 bloc-instructions

exemple: Voici le programme correspondant au diagramme de la page précédente:

```
//Lire un âge valide (de 0 à 120 inclusivement)
cout << "Entrez votre âge: ";
cin >> Age;
while(Age<0 || Age>120)
{
    cout << "\nÂge invalide, recommencez: ";
    cin >> Age;
}
cout << "\n\nVotre âge est: " << Age;
```

Notez l'absence du point-virgule après la parenthèse fermante du *while()*. Un point-virgule ici signifierait qu'il n'y a aucun énoncé à répéter (ou que l'énoncé «*vide*» sera répété). La raison en est qu'un bloc-instructions peut n'être composé que d'une seule instruction se terminant par un point-virgule, et qu'alors les accolades (`{ }`) sont facultatives. Une telle erreur ne sera pas détectée par le compilateur, car c'est correct d'un point de vue syntaxique. Toutefois, d'un point de vue logique, votre programme ne répétera pas les énoncés prévus.

Voici comment s'exécute un *while()*:

1. Évaluation de la condition pour poursuivre :
 - si la condition est fausse, on quitte le *while()* (la répétition se **termine** ici)
 - si la condition est vraie, on exécute les instructions du *while()*
2. Retour à l'étape 1..

La répétition se poursuit donc jusqu'à ce que la condition soit fausse. Si elle ne devenait jamais fausse, on aurait alors une *boucle infinie*, soit une répétition sans fin. Le programmeur devra s'assurer que ça n'arrive pas. Il est aussi possible que le bloc-instructions ne soit pas exécuté une seule fois. Cela se produit quand la condition pour poursuivre est fausse dès le début.

EXEMPLE 1

LE PROBLÈME

On veut calculer la moyenne de plusieurs nombres réels. On ne sait pas d'avance combien il y aura de nombres à lire.

QUE VEUT-ON À LA SORTIE?

La moyenne de tous les nombres

QUELLES SONT LES DONNÉES CONNUES?

Les nombres dont on veut calculer la moyenne.

LE DICTIONNAIRE DE DONNÉES

Identificateur	Signification	Type	Valeur
Nombre	Chacun des nombres à lire	float	lue au clavier
Somme	Somme de tous les nombres	float	Somme + Nombre
Moyenne	Moyenne de tous les nombres	float	Somme/QteNombres
QteNombres	Quantité de nombres lus, i.e. combien il y a de nombres pour lesquels on veut la moyenne	int	QteNombres + 1 (1, 2, 3 ...)
Continuer	Indique s'il y a d'autres nombres (si on veut continuer)	char	

LE DIAGRAMME D'ACTION

```
Somme = 0
QteNombres=0
Continuer='O'
  Tant que Continuer == 'O'
    Lire Nombre
    Somme = Somme+Nombre
    QteNombres = QteNombres+1
    Écrire "Y-a-t-il d'autres nombres (O/N)?: "
    Lire Continuer
  Moyenne = Somme/QteNombres
  Écrire "La moyenne est: ", Moyenne
```


LE PROGRAMME

```

/*****
AUTEUR: Marcel L'Italien
NOM DU PROGRAMME: 4TQ1.CPP
DESCRIPTION: Programme qui calcule la moyenne de n nombres.
*****/
#include <iostream>
#include <iomanip>
using namespace std;

void main(void)
{
    int QteNombres;
    float Nombre,
          Somme,
          Moyenne;
    char Continuer;

    Somme = 0;
    QteNombres = 0;

    Continuer = 'O';
    while(Continuer == 'O')
    {
        cout << "\nEntrez un nombre: ";
        cin >> Nombre;

        Somme += Nombre;
        QteNombres++;

        cout << "\nY-a-t-il d'autres nombres (O/N)?: ";
        cin >> Continuer;
        Continuer=toupper(Continuer); // transformer en majuscule
    }

    Moyenne = Somme/QteNombres;

    cout << "\n\n\tLa moyenne est: " << fixed << setprecision(2) << Moyenne;
}

```

LE RÉSULTAT

```

Entrez un nombre: 10
Y-a-t-il d'autres nombres (O/N)?: O
Entrez un nombre: 5
Y-a-t-il d'autres nombres (O/N)?: O
Entrez un nombre: 0
Y-a-t-il d'autres nombres (O/N)?: N

\t\t\t\t\tLa moyenne est: 5.00

```

EXEMPLE 2

LE PROBLÈME

Afficher le nombre de jours qu'il y a dans un mois quelconque. Le mois sera lu sous la forme d'un nombre entier. On devra valider le mois (1 à 12 seulement). Ne pas tenir compte des années bissextiles.

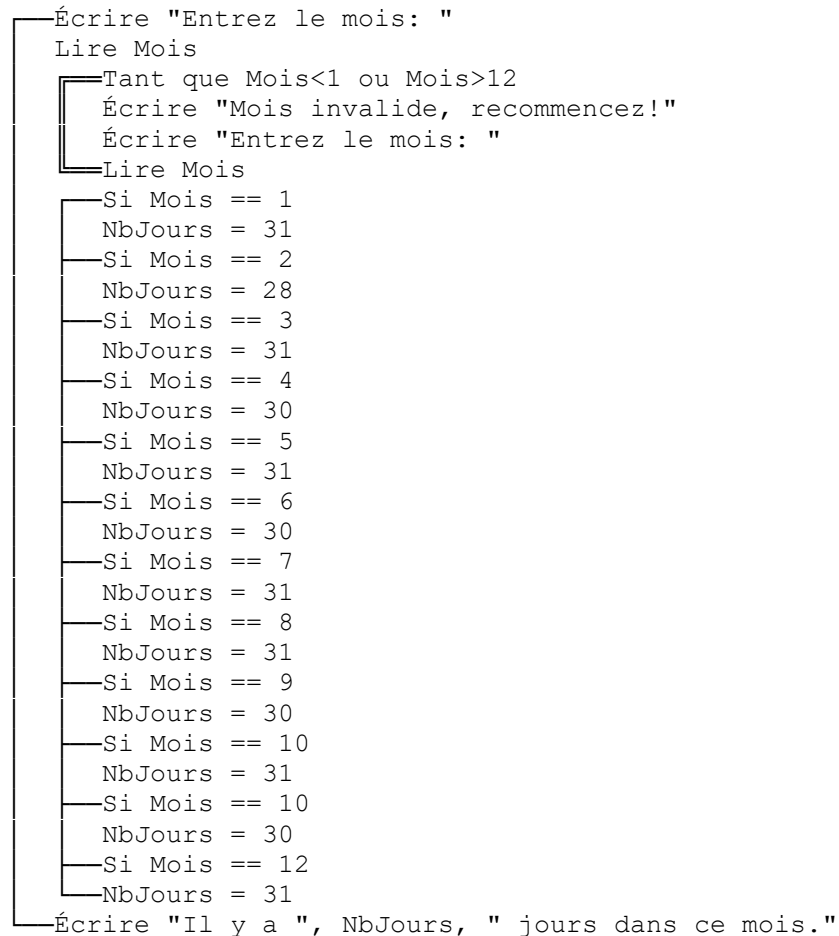
QUE VEUT-ON À LA SORTIE? Le nombre de jours

QUELLES SONT LES DONNÉES CONNUES? Le numéro du mois

LE DICTIONNAIRE DE DONNÉES

Identificateur	Signification	Type	Valeur
Mois	Mois en chiffres	int	lue au clavier
NbJours	Nombre de jours dans Mois	int	31, 30 ou 28 selon le mois

LE DIAGRAMME D'ACTION



LE PROGRAMME

```

/*****
AUTEUR: Marcel L'Italien
NOM DU PROGRAMME: 4TQ2.CPP
DESCRIPTION: Programme affichant le nombre de jours qu'il y a dans un
             mois. On ne tient pas compte de l'année bissextile.
             Le mois sera lu au clavier sous forme d'un entier. Il y
             aura validation du mois (1 à 12 seulement).
*****/
#include <iostream>
using namespace std;
void main(void)
{   int   Mois,
        NbJours;

    cout << "Entrez le mois: ";
    cin >> Mois;

    while(Mois<1 || Mois>12)
    {
        cout << "\nMois invalide, recommencez!\n\n";
        cout << "Entrez le mois: ";
        cin >> Mois;
    }
    switch(Mois)
    {
        case 1: NbJours = 31;
                 break;
        case 2: NbJours = 28;
                 break;
        case 3: NbJours = 31;
                 break;
        case 4: NbJours = 30;
                 break;
        case 5: NbJours = 31;
                 break;
        case 6: NbJours = 30;
                 break;
        case 7: NbJours = 31;
                 break;
        case 8: NbJours = 31;
                 break;
        case 9: NbJours = 30;
                 break;
        case 10: NbJours = 31;
                 break;
        case 11: NbJours = 30;
                 break;
        case 12: NbJours = 31;
                 break;
    }
    cout << "\n\n\tIl y a " << NbJours << " jours dans ce mois.";
}

```

LE RÉSULTAT

```
Entrez le mois: 13  
  
Mois invalide, recommencez!  
  
Entrez le mois: 1  
  
Il y a 31 jours dans ce mois.
```

EXEMPLE 2A

LE PROBLÈME

Problème identique au précédent, mais avec un algorithme un peu différent. Il s'agit d'afficher le nombre de jours qu'il y a dans un mois quelconque. Le mois sera lu sous la forme d'un nombre entier. On devra valider le mois (1 à 12 seulement). Ne pas tenir compte des années bissextiles.

QUE VEUT-ON À LA SORTIE?

Le nombre de jours

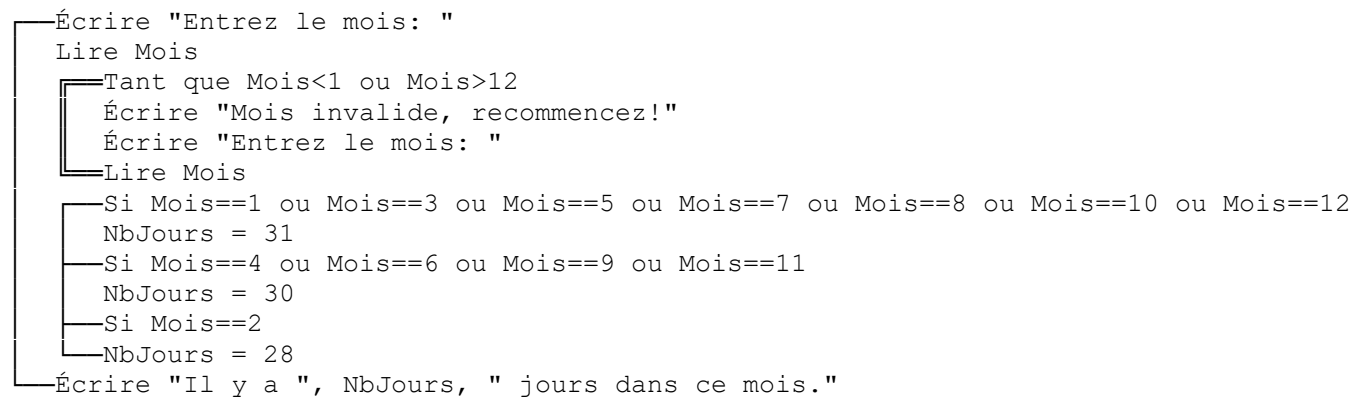
QUELLES SONT LES DONNÉES CONNUES?

Le numéro du mois

LE DICTIONNAIRE DE DONNÉES

Identificateur	Signification	Type	Valeur
Mois	Mois en chiffres	int	lue au clavier
NbJours	Nombre de jours dans Mois	int	31, 30 ou 28 selon le mois

LE DIAGRAMME D'ACTION



LE PROGRAMME

```

/*****
AUTEUR: Marcel L'Italien
NOM DU PROGRAMME: 4TQ2A.CPP
DESCRIPTION: Programme similaire au précédent, seule la SÉLECTION MULTIPLE
             est différente.
             On affiche le nombre de jours qu'il y a dans un
             mois. On ne tient pas compte de l'année bissextile.
             Le mois sera lu au clavier sous forme d'un entier. Il y
             aura validation du mois (1 à 12 seulement).
*****/
#include <iostream>

using namespace std;

void main(void)
{
    int  Mois,
        NbJours;

    cout << "Entrez le mois: ";
    cin >> Mois;

    while(Mois<1 || Mois>12)
    {
        cout << "\nMois invalide, recommencez!\n\n";
        cout << "Entrez le mois: ";
        cin >> Mois;
    }

    switch(Mois)
    {
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12: NbJours = 31;
                break;

        case 4:
        case 6:
        case 9:
        case 11: NbJours = 30;
                break;

        case 2: NbJours = 28;
    }

    cout << "\n\n\tIl y a " << NbJours << " jours dans ce mois.";
}

```

EXEMPLE 3

LE PROBLÈME

Problème identique au précédent, mais avec un algorithme un peu différent.

Il s'agit d'afficher le nombre de jours qu'il y a dans un mois quelconque. Le mois sera lu sous la forme d'un nombre entier. On devra valider le mois (nombre de 1 à 12 seulement). Ne pas tenir compte des années bissextiles.

QUE VEUT-ON À LA SORTIE?

Le nombre de jours

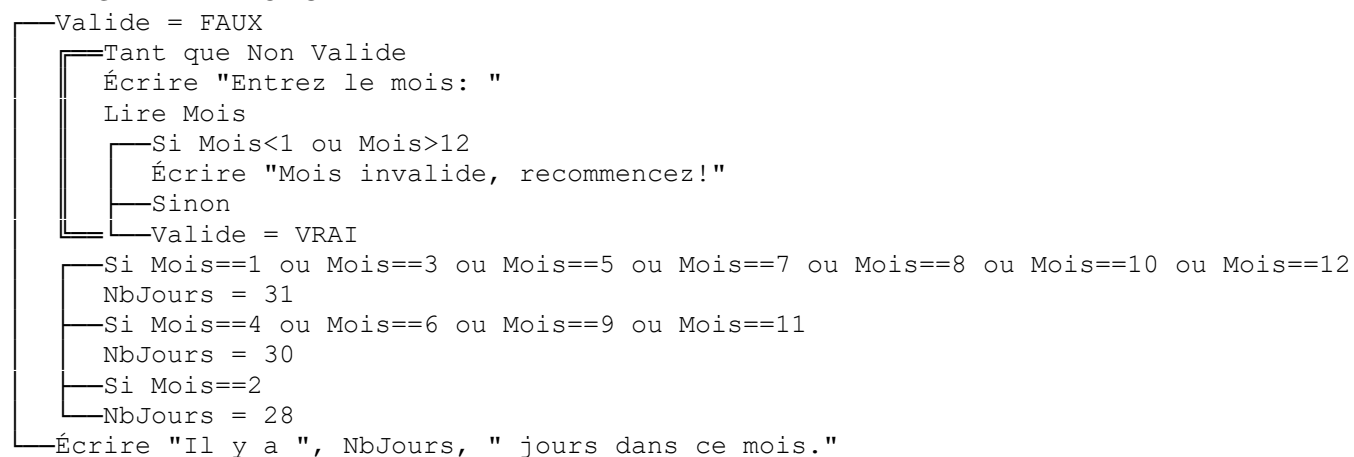
QUELLES SONT LES DONNÉES CONNUES?

Le numéro du mois

LE DICTIONNAIRE DE DONNÉES

Identificateur	Signification	Type	Valeur
Mois	Mois en chiffres	int	lue au clavier
NbJours	Nombre de jours dans Mois	int	31, 30 ou 28 selon le mois
Valide	Indique si le Mois est valide ou non	int	VRAI ou FAUX

LE DIAGRAMME D'ACTION



LE PROGRAMME

```

/*****
AUTEUR: et Marcel L'Italien
NOM DU PROGRAMME: 4TQ3.CPP
DESCRIPTION: Programme similaire au précédent, mais ici on utilise une
              variable BOOLÉENNE comme condition du while(). Aussi on codera
              la sélection multiple différemment ( sans switch() )

*****/
#include <iostream>

using namespace std;

void main(void)
{
    const int VRAI=1,
              FAUX=0;
    int  Mois,
        NbJours,
        Valide;

    Valide=FAUX;

    while(!Valide)
    {
        cout << "Entrez le mois: ";
        cin >> Mois;

        if(Mois<1 || Mois>12)
            cout << "\nMois invalide, recommencez!\n\n";
        else
            Valide=VRAI;
    }

    if (Mois==1 || Mois==3 || Mois==5 || Mois==7 || Mois==8 || Mois==10 || Mois==12)
        NbJours = 31;
    else if (Mois==4 || Mois==6 || Mois==9 || Mois==11)
        NbJours = 30;
    else if (Mois==2)
        NbJours = 28;

    cout << "\n\n\tIl y a " << NbJours << " jours dans ce mois.";
}

```

EXEMPLE 4

LE PROBLÈME

Problème identique au précédent, mais en plus on vérifiera que le mois entré au clavier est bien un **entier** valide.

Il s'agit d'afficher le nombre de jours qu'il y a dans un mois quelconque. Le mois sera lu sous la forme d'un nombre entier. On devra valider le mois (nombre de 1 à 12 seulement). Ne pas tenir compte des années bissextiles.

QUE VEUT-ON À LA SORTIE?

Le nombre de jours

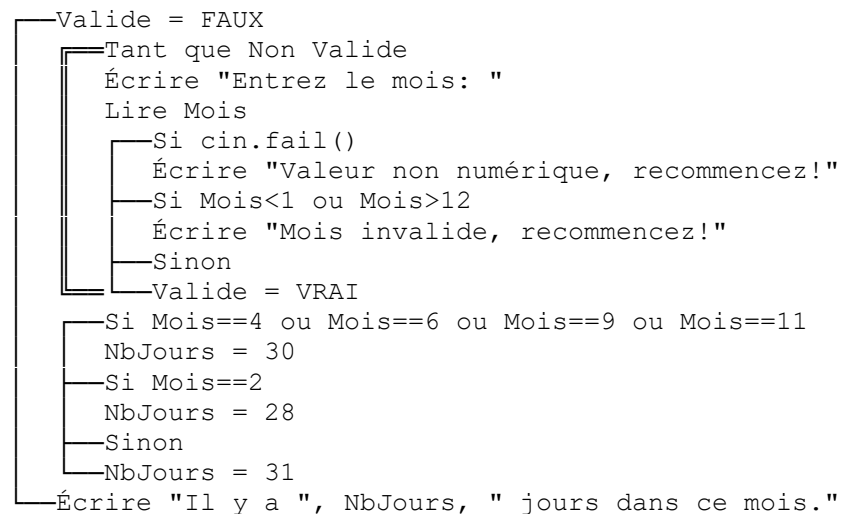
QUELLES SONT LES DONNÉES CONNUES?

Le numéro du mois

LE DICTIONNAIRE DE DONNÉES

Identificateur	Signification	Type	Valeur
Mois	Mois en chiffres	int	lue au clavier
NbJours	Nombre de jours dans Mois	int	31, 30 ou 28 selon le mois
Valide	Indique si le Mois est valide ou non	int	VRAI ou FAUX

LE DIAGRAMME D'ACTION



LE PROGRAMME

```

/*****
AUTEUR: Marcel L'Italien
NOM DU PROGRAMME: 4TQ4.CPP
DESCRIPTION: Similaire au précédent, mais en plus on vérifiera que le mois
             lu est bien un entier valide (et pas du texte ou autres
             caractères invalides). Nous utiliserons donc CIN.FAIL().
*****/
#include <iostream>
using namespace std;
void main(void)
{
    const int VRAI=1, FAUX=0;
    int Mois,
        NbJours,
        Valide;

    Valide=FAUX;
    while(!Valide)
    {
        cout << "Entrez le mois: ";
        cin >> Mois;
        if (cin.fail())
        {
            cin.clear();
            cin.ignore(80, '\n');
            cout << "\nValeur non numérique, recommencez!\n\n";
        }
        else if(Mois<1 || Mois>12)
            cout << "\nMois invalide, recommencez!\n\n";
        else
            Valide=VRAI;
    }
    if (Mois==4 || Mois==6 || Mois==9 || Mois==11)
        NbJours = 30;
    else if (Mois==2)
        NbJours = 28;
    else
        NbJours = 31;
    cout << "\n\tIl y a " << NbJours << " jours dans ce mois.";
}

```

LE RÉSULTAT

```

Entrez le mois: 13

Mois invalide, recommencez!

Entrez le mois: abc

Valeur non numérique, recommencez!

Entrez le mois: 12

    Il y a 31 jours dans ce mois.

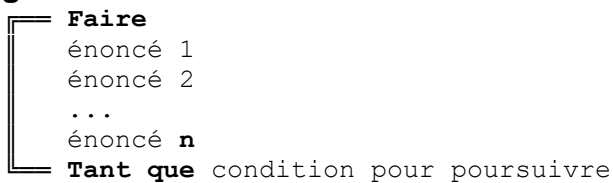
```

4.7 L'ÉNONCÉ FAIRE...TANT QUE

Cet énoncé de répétition est très semblable à un **Tant que**. Il sert donc à répéter un bloc d'instructions lorsque nous ne connaissons pas le nombre exact de fois. Il faut toutefois noter que cette répétition se fera au moins une fois, c'est d'ailleurs ce qui différencie le **Faire...tant que** du **Tant que**.

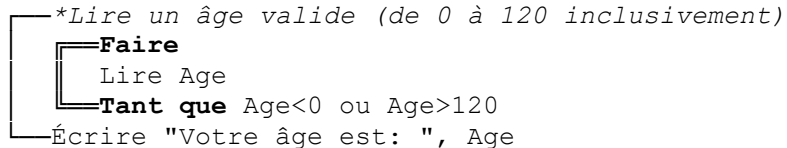
L'énoncé **Faire...tant que** peut être utilisé lorsqu'on ne connaît pas le nombre exact de fois où la répétition sera exécutée, mais qu'elle doit l'être au moins une fois.

diagramme:



Il y aura autant d'énoncés que désiré (de 0 à n)

exemple:



Ici la lecture de l'âge est répétée jusqu'à ce que l'âge soit valide (compris entre 0 et 120 inclusivement). Puisqu'on ne sait pas combien de tentatives il faudra à l'utilisateur pour entrer une valeur correcte, on ne sait pas combien de fois la répétition se fera.

Cet exemple est similaire à celui présenté à la page 127 (pour l'énoncé Tant que), mais on notera que l'exemple présenté ici est préférable, car on évite une lecture avant la boucle, obtenant donc une solution plus simple. Notez que la lecture de l'âge se fait au moins une fois, puisque la condition est évaluée après, ce qui justifie, ici, l'utilisation du **Faire...tant que** plutôt que du **Tant que**. Cependant, il s'agit d'un des rares exemples où c'est le cas.

Un **Faire...tant que** est répété tant que sa condition est vraie, c'est pourquoi il s'agit d'une condition pour poursuivre, et non pas d'une condition d'arrêt. Pour arrêter cette répétition, ici, la variable **Age** devra avoir une valeur comprise entre 0 et 120 inclusivement (ainsi la condition *Age < 0 ou Age > 120* sera fausse). Après l'exécution de ce **Faire...tant que**, la valeur de **Age** est donc valide.

4.8 L'INSTRUCTION DO...WHILE()

L'énoncé du langage C++ correspondant à la répétition **Faire...tant que** est l'instruction *do...while()*.

syntaxe:

```
do
    bloc-instructions
while (condition pour poursuivre);
```

exemple: Voici le programme correspondant au diagramme de la page précédente:

```
//Lire un âge valide (de 0 à 120 inclusivement)
do
{
    cout << "\nEntrez votre âge: ";
    cin >> Age;
}
while (Age<0 || Age>120);
cout << "Votre âge est: " << Age;
```

Notez la présence du point-virgule après la parenthèse fermante du *while()*. Le point-virgule ici permet de différencier un *do...while()* d'un *while()*.

Voici comment s'exécute un *do...while()*:

1. Exécution du bloc-instructions du *do...while()*
2. Évaluation de la condition pour poursuivre :
 - si la condition est fausse, on quitte le *do...while()* (la répétition se **termine**)
 - si la condition est vraie, on retourne à l'étape 1..

La répétition se poursuit donc jusqu'à ce que la condition soit fausse. Si elle ne devenait jamais fausse, on aurait alors une *boucle infinie*, soit une répétition sans fin. Le programmeur devra s'assurer que ça n'arrive pas.

EXEMPLE 1

LE PROBLÈME

On veut calculer la moyenne de plusieurs nombres réels. On ne sait pas d'avance combien il y aura de nombres à lire.

QUE VEUT-ON À LA SORTIE?

La moyenne de tous les nombres

QUELLES SONT LES DONNÉES CONNUES?

Les nombres dont on veut calculer la moyenne.

LE DICTIONNAIRE DE DONNÉES

Identificateur	Signification	Type	Valeur
Nombre	Chacun des nombres à lire	float	lue au clavier
Somme	Somme de tous les nombres	float	Somme + Nombre
Moyenne	Moyenne de tous les nombres	float	Somme/QteNombres
QteNombres	Quantité de nombres lus, i.e. combien il y a de nombres pour lesquels on veut la moyenne	int	QteNombres + 1 (1, 2, 3 ...)
Continuer	Indique s'il y a d'autres nombres (si on veut continuer)	char	

LE DIAGRAMME D'ACTION

```
Somme = 0
QteNombres=0
  Faire
    Lire Nombre
    Somme = Somme+Nombre
    QteNombres = QteNombres+1
    Écrire "Y-a-t-il d'autres nombres (O/N)?: "
    Lire Continuer
  Tant que Continuer == 'O'
Moyenne = Somme/QteNombres
Écrire "La moyenne est: ", Moyenne
```

LE PROGRAMME

```

/*****
AUTEUR: Marcel L'Italien
NOM DU PROGRAMME: 4FTQ1.CPP
DESCRIPTION: Programme qui calcule la moyenne de n nombres.
             Ce programme est similaire à l'exemple 4TQ1.CPP de la page 129.
             Puisque pour calculer une moyenne il faut lire au moins
             un nombre (au moins une répétition) le do-while() est utilisé
             ici. Avec cette solution on n'a pas besoin d'initialiser la
             variable «Continuer» avant la boucle.
*****/
#include <iostream>
#include <iomanip>
using namespace std;
void main(void)
{
    int QteNombres;
    float Nombre,
          Somme,
          Moyenne;
    char Continuer;
    Somme = 0;
    QteNombres = 0;
    do
    {
        cout << "\nEntrez un nombre: ";
        cin >> Nombre;

        Somme += Nombre;
        QteNombres++;

        cout << "\nY-a-t-il d'autres nombres (O/N)?: ";
        cin >> Continuer;
        Continuer=toupper(Continuer); // transformer en majuscule
    }
    while(Continuer == 'O');
    Moyenne = Somme/QteNombres;
    cout << "\n\n\tLa moyenne est: "
         << fixed << setprecision(2) << Moyenne;
}

```

LE RÉSULTAT

EXEMPLE 2

LE PROBLÈME

Simuler un chronomètre. Pour démarrer le chronomètre il faudra presser la touche *<Retour>*. On pourra arrêter le chronomètre avec n'importe quelle touche.

QUE VEUT-ON À LA SORTIE?

Le nombre de secondes écoulées

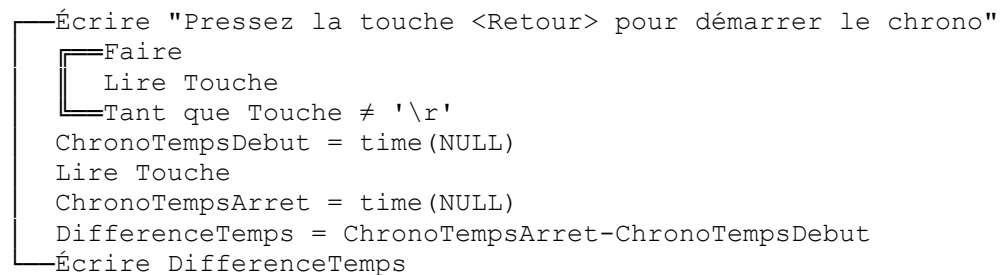
QUELLES SONT LES DONNÉES CONNUES?

Le signal de départ et le signal d'arrêt.

LE DICTIONNAIRE DE DONNÉES

Identificateur	Signification	Type	Valeur
ChronoTempsArret	Temps à l'arrêt du chronomètre	unsigned long	time(NULL)
ChronoTempsDebut	Temps au départ du chronomètre	unsigned long	time(NULL)
DifferenceTemps	Différence entre les 2 temps	unsigned long	ChronoTempsArret – ChronoTempsDebut
Touche	La touche pressée par l'usager. Elle servira à indiquer le départ et l'arrêt du chronomètre.	char	lue au clavier

LE DIAGRAMME D'ACTION



4.9 FONCTION TIME() (EXEMPLE)

LE PROGRAMME

```

/*****
AUTEUR: Marcel L'Italien
NOM DU PROGRAMME: 4FTQ2.CPP
DESCRIPTION: Ce programme simule un chronomètre.
*****/
#include <iostream>
#include <time.h> // Pour la fonction time()
#include <conio.h> // Pour _getch()
using namespace std;

void main(void)
{
    time_t    ChronoTempsArret,    // time_t est le type du résultat de la
                                ChronoTempsDebut,    // fonction time()
                                DifferenceTemps;    // ce type est un dérivé de unsigned long

    char Touche;

    cout << "\nPressez la touche «Retour» pour démarrer le chrono\n";
    do
    {
        Touche = _getch();
    }
    while( Touche != '\r');

    // Le temps de l'horloge au début de la boucle
    ChronoTempsDebut = time(NULL); /* Cette fonction retourne le nombre de secondes
                                    écoulées depuis le 1er janvier 1970 */

    cout << "\n\nPressez une touche pour arrêter le chrono";
    Touche = _getch();

    // Le temps de l'horloge à la fin de la boucle
    ChronoTempsArret = time(NULL);

    DifferenceTemps = ChronoTempsArret - ChronoTempsDebut;
    cout << "\n\nLe temps écoulé: " << DifferenceTemps << " secondes\n";
}

```

LE RÉSULTAT

```

Pressez la touche «Retour» pour démarrer le chrono

Pressez une touche pour arrêter le chrono

Le temps écoulé: 2 secondes

```

EXEMPLE 3

LE PROBLÈME

On désire un programme qui calcule les intérêts sur un dépôt après un certain nombre d'années.

QUE VEUT-ON À LA SORTIE?

La valeur finale du placement (dépôt).

QUELLES SONT LES DONNÉES CONNUES?

La valeur initiale du dépôt, le nombre d'années et le taux d'intérêt

LE DICTIONNAIRE DE DONNÉES

Identificateur	Signification	Type	Valeur
CoefficientInteret	Taux d'intérêt	float	TauxInteret/100
NombreAnnees	Nombre d'année du dépôt	float	Lue au clavier
ValeurFinale	Valeur finale du dépôt	float	$ValeurPresente * (1 + CoefficientInteret)^{NombreAnnees}$
Valeur Presente	Valeur initiale du dépôt	float	Lue au clavier
TauxInteret	Taux de l'intérêt	float	Lue au clavier
Poursuivre	Indicateur de fin de traitement	char	Lue au clavier

LE DIAGRAMME D'ACTION

```
Faire
┌ Faire
│   Lire ValeurPresente
│   Tant Que ValeurPresente < 0
│   Lire NombreAnnees
│   Tant Que NombreAnnees < 0
│   Lire TauxInteret
│   Tant Que TauxInteret < 0
│   Coef_Interet = TauxInteret/100
│   ValeurFinale = ValeurPresente * (1+CoefficientInteret)NombreAnnees
│   Écrire ValeurFinale
│   Lire Poursuivre
│   Tant Que Poursuivre == '0'
```


LE PROGRAMME

```

/*****
AUTEUR: Marcel L'Italien
NOM DU PROGRAMME: 4FTQ3.CPP
DESCRIPTION: Calculer la valeur d'un dépôt, après n années, à un
             taux d'intérêt x.
*****/

#include <iostream>
#include <iomanip>
#include <math.h>      // Pour la fonction pow()
#include "c:\cvm.h"
using namespace std;

void main(void)
{
    float ValeurPresente, TauxInteret, CoefficientInteret, ValeurFinale,
          NombreAnnee;

    char Poursuivre;

    cout << fixed;

    do
    {
        clrscr();
        do
        {
            gotoxy(1,1);
            clreol();
            cout << "Donner la valeur initiale du montant (>0): ";
            cin >>ValeurPresente;
        }
        while (ValeurPresente < 0);

        do
        {
            gotoxy(1,2);
            clreol();
            cout << "Donner le nombre d'année (>0): ";
            cin >> NombreAnnee;
        }
        while (NombreAnnee < 0);

        do
        {
            gotoxy(1,3);
            clreol();
            cout << "Donner le taux d'intérêt (>0): ";
            cin >> TauxInteret;
        }
        while (TauxInteret < 0);
    }
}
```

```

CoefficientInteret = TauxInteret/100;

ValeurFinale = ValeurPresente * pow((1+CoefficientInteret),NombreAnnee);

cout << "\nLa valeur finale est de: "
      << setw(10) << setprecision(2) << ValeurFinale << '$';

gotoxy(1, 10);
cout << "Voulez-vous continuer(O/N)? ";
cin >> Poursuivre;
}
while(toupper(Poursuivre) == 'O');
}

```

LE RÉSULTAT

```

Donner la valeur initiale du montant (>0): 2000
Donner le nombre d'année (>0): 5
Donner le taux d'intérêt (>0): 7.5

La valeur finale est de:      2871.26$

Voulez-vous continuer(O/N)? n

```

4.10 EXERCICES

1. Nommez les trois parties de l'énoncé *for()*?
Quand doit-on employer l'énoncé *for()*?
2. Quel est le rôle de l'instruction *while()*?
À quel moment sa condition est-elle évaluée?
Combien de fois, au minimum, une boucle *while()* peut-elle être répétée?
À quel moment l'exécution d'une boucle *while()* se termine-t-elle?
3. Quel est le rôle de l'instruction *for()*?
À quel moment sa condition est-elle évaluée?
Combien de fois, au minimum, une boucle *for()* peut-elle être répétée?
À quel moment l'exécution d'une boucle *for()* se termine-t-elle?
4. Quel est le rôle de l'instruction *do...while()*?
À quel moment sa condition est-elle évaluée?
Combien de fois, au minimum, une boucle *do...while()* peut-elle être parcourue?
À quel moment l'exécution d'une boucle *do...while()* se termine-t-elle?
5. Le corps d'une boucle (son bloc-instructions) débute par:
 - a) un point virgule;
 - b) une accolade fermante;
 - c) une accolade ouverte;
 - d) l'indentation du bloc d'instructions.
6. On utilise l'instruction *while()* plutôt que l'instruction *for()* lorsque:
 - a) la fin de la boucle est prévisible;
 - b) le corps de la boucle sera exécuté au moins une fois;
 - c) le programme est exécuté au moins une fois;
 - d) le nombre de fois que la boucle sera exécutée est prévisible.
7. Écrire le diagramme d'action, puis l'instruction (en C) permettant de répéter un bloc-instructions selon les critères qui suivent:
 - a) répéter un bloc-instructions trois fois
 - b) répéter un bloc-instructions trois fois, mais nous aurons besoin, pour le traitement, des nombres 10, 20 et 30 (un pour chaque répétition)
 - c) répéter un bloc-instructions *Nb* fois, en comptant de façon décroissante.
 - d) répéter un bloc-instructions 10 fois, maximum, à condition que la variable *Nombre* soit positive
 - e) répéter un bloc-instructions tant que la variable *Car* est différente de 'N'
 - f) répéter un bloc-instructions tant que *Fin* est différent de VRAI
 - g) répéter un bloc-instructions jusqu'à ce que la variable *Car* soit égale à 'N'

8. Écrire un programme qui permet d'afficher les entiers de 0 à 9 en ordre décroissant.

LE DIAGRAMME D'ACTION

```

┌─ Pour Nombre=9, Nombre>=0, -1
└─ Écrire Nombre

```

9. Écrire un programme qui permet d'afficher 4 lignes de 10 étoiles (afficher une étoile à la fois).

```

→ * * * * *
  * * * * *
  * * * * *
  * * * * *

```

10. Écrire un programme qui permet d'afficher un carré formé d'étoiles. Nous demanderons à l'utilisateur combien d'étoiles il veut, par exemple 5, puis nous afficherons un carré constitué de 5 lignes et 5 colonnes, et rempli d'étoiles (afficher une étoile à la fois).

```

→ exemple: * * * * *
            * * * * *
            * * * * *
            * * * * *
            * * * * *

```

11. Écrire un programme qui affiche des triangles formés d'étoiles. On désire les triangles suivants :

```

*
* *
* * *
* * * *
* * * * *
et
*
* * *
* * * *
* * * * *

```

12. Écrire un programme qui affiche la racine carrée, le carré et le cube des entiers de 1 à 9.

L'ÉCRAN DE RÉSULTATS

Nombre	Racine	Carré	Cube
1	1.00	1.00	1.00
2	1.41	4.00	8.00
...			

13. Écrire un programme qui affiche le texte **MENU PRINCIPAL** entouré d'un cadre formé d'étoiles.

L'ÉCRAN DE RÉSULTATS

```

*****
*  MENU PRINCIPAL  *
*****

```

LE DIAGRAMME D'ACTION

```

┌─ Pour Colonne=1, Colonne<=18, 1
└─ Écrire '*'
  Écrire '*', "MENU PRINCIPAL", '*'
┌─ Pour Colonne=1, <=18, 1
└─ Écrire '*'

```

14. Écrire un programme qui calcule la valeur d'un nombre entier élevé à une certaine puissance.
15. Écrire un programme qui lit au clavier la longueur des trois côtés d'un triangle. Il faudra ensuite déterminer si ces trois longueurs sont vraiment celles d'un triangle, et si oui, si ce triangle est équilatéral, isocèle ou scalène. Dans le cas où les longueurs des côtés sont invalides (on ne peut créer un triangle avec ces côtés) on redemande de nouvelles données, jusqu'à l'obtention de valeurs correctes.
16. Lire, au clavier, des nombres entre 0 et 100. Compter et afficher les statistiques suivantes:
 - combien de nombres sont inférieurs à 50
 - combien de nombres sont supérieurs ou égaux à 50
 - combien de nombres sont pairs
 - combien de nombres sont impairs
17. Écrire un programme qui permet à l'ordinateur de deviner le nombre choisi par l'utilisateur (entier entre 1 et 99 inclusivement). Il faudra 7 tentatives au maximum. L'ordinateur affiche diverses valeurs (des essais) et pour chacune l'utilisateur indique si le nombre est plus grand, plus petit ou égal au nombre qu'il a choisi. La méthode de utilisée est une recherche *binnaire*.

L' ÉCRAN

Choisissez un nombre entre 1 et 99, et je le devine:

tapez 'e' si j'ai trouvé la bonne réponse,

tapez 'g' si votre nombre est plus grand que ma réponse

tapez 'p' si votre nombre est plus petit que ma réponse

Est-ce que votre nombre est égal, plus petit ou plus grand que: 50? g

Est-ce que votre nombre est égal, plus petit ou plus grand que: 25? g

Est-ce que votre nombre est égal, plus petit ou plus grand que: 12? g

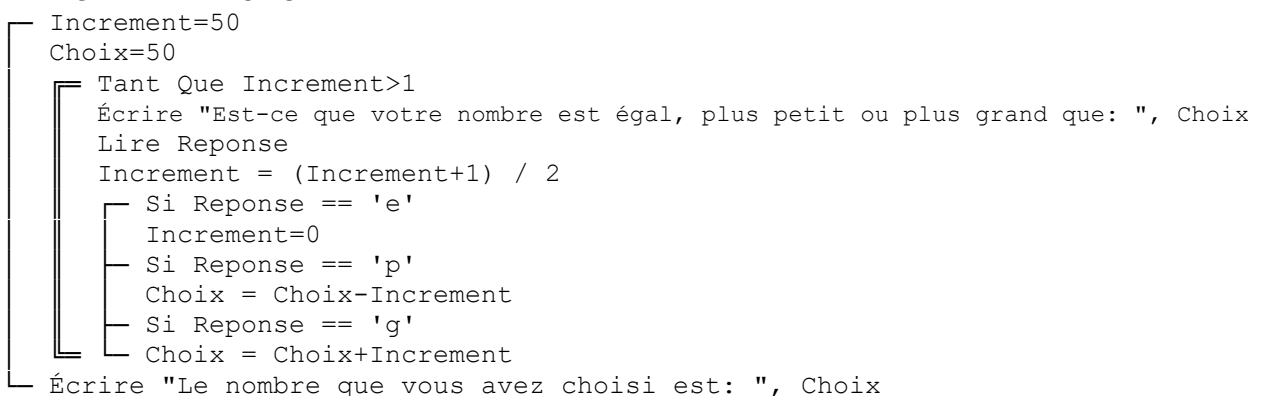
Est-ce que votre nombre est égal, plus petit ou plus grand que: 19? g

Est-ce que votre nombre est égal, plus petit ou plus grand que: 23? g

Est-ce que votre nombre est égal, plus petit ou plus grand que: 21? g

Le nombre que vous avez choisi est: 22

LE DIAGRAMME D'ACTION



CHAPITRE 5

LES VARIABLES DIMENSIONNÉES

5.1 LES VARIABLES DIMENSIONNÉES

Considérez le programme suivant:

```
/******  
AUTEUR: Marcel L'Italien  
NOM DU PROGRAMME: 5DIM0.CPP  
DESCRIPTION: Lire 3 nombres puis les afficher en ordre inverse.  
*****/  
#include <iostream>  
using namespace std;  
  
void main(void)  
{  
    int Nb1,  
        Nb2,  
        Nb3;  
  
    cout << "entrez 3 nombres entiers: ";  
    cin >> Nb1 >> Nb2 >> Nb3;  
  
    cout << "\nVoici les trois nombres en ordre inverse:\n";  
    cout << Nb3 << "\n" << Nb2 << "\n" << Nb1;  
}
```

LE RÉSULTAT

```
entrez 3 nombres entiers: 10 5 20  
  
Voici les trois nombres en ordre inverse:  
20  
5  
10
```

Le problème consistait donc à lire 3 nombres, puis à les afficher en ordre inverse. Supposons maintenant que nous devons faire la même chose, mais avec 50 nombres. En utilisant la même méthode que précédemment, il faudra définir 50 variables, puis il faudra les lire et les afficher une à une. Ce sera long et fastidieux.

Le principal problème que nous rencontrons ici est la définition de plusieurs variables (en fait 50). La solution sera la création d'une *variable dimensionnée*. Une variable dimensionnée est un ensemble d'éléments de données qui sont tous du même type. Une telle variable peut contenir autant d'éléments que nécessaire. Par exemple pour stocker 50 nombres entiers nous définirons la variable dimensionnée suivante:

int Nombre[50];

Notez que la définition de cette variable est similaire à toute autre définition, sauf pour les crochets qui y apparaissent ([50]).

Une variable dimensionnée devra donc, au même titre que toute autre variable, être définie. En voici la syntaxe:

Une variable dimensionnée se définit en précisant son type, son identificateur et sa grandeur, comme suit:

```
type identificateur[grandeur];
```

Le *type* peut être tout type prédéfini du langage (**int**, **float**, **char**, etc) ou défini par le programmeur (voir *typedef* et *struct*). La grandeur représente le nombre d'éléments qui composent la variable dimensionnée, et est indiquée par le nombre qui apparaît entre crochets [] à la suite de l'identificateur. Cette grandeur devra être une constante seulement (pas une variable).

Exemples:

<code>int Note[5];</code>	définition de la variable <i>Note</i> composée de 5 éléments de type entier.
---------------------------	--

<code>string Nom[30];</code>	définition de la variable <i>Nom</i> composée de 30 éléments de type string.
------------------------------	--

<code>const int NB_VOYELLES=6;</code>	définition de la variable <i>Voyelle</i> composée de 6 éléments
<code>char Voyelle[NB_VOYELLES];</code>	de type caractère.

5.2 LES ÉLÉMENTS D'UNE VARIABLE DIMENSIONNÉE.

Considérez la variable suivante:

```
float Ventes[12];
```

La variable **Ventes** est une variable dimensionnée servant à stocker les ventes totales de chaque mois de l'année. Pour afficher les ventes du 1^{er} mois, soit janvier, on fera:

```
cout << Ventes[0];
```

De même, pour afficher les ventes du 12^e mois (décembre) on fera:

```
cout << Ventes[11];
```

Notez la présence, entre crochets et à la suite de l'identificateur, d'un *indice*. L'indice est un nombre, une variable ou une expression entière qui identifie un seul élément parmi l'ensemble des éléments composant la variable dimensionnée. L'indice est en fait un **déplacement** relatif au début de la variable dimensionnée, c'est pourquoi le premier élément est référé par l'indice **[0]** (le premier élément se trouve au début de la variable dimensionnée, donc à un déplacement de **0**). Puisque tous les éléments d'une variable dimensionnée sont placés dans des positions contiguës, le second

élément se trouve à la suite du premier, donc en avançant d'un élément par rapport au début, d'où l'indice **[1]**, le troisième élément se trouve à la suite des deux premiers, donc en avançant de deux éléments par rapport au début, d'où l'indice **[2]**, etc.

Pour une variable dimensionnée définie comme suit:

type var[grandeur];

puisque l'indice est un déplacement, ses valeurs permises vont de **0** à **grandeur-1**. Les éléments sont donc référés comme suit:

var[0]	→ premier élément
var[1]	→ second élément
...	
var[grandeur-1]	→ dernier élément

5.3 LA VÉRIFICATION DES FRONTIÈRES.

En langage C++, il n'y a pas de vérification automatique des frontières d'une variable dimensionnée *statique* (le type de variable dimensionnée vu ici). La valeur de l'indice peut donc être invalide sans que le compilateur nous en informe lors de la compilation. Par exemple avec la définition de variable **int Nombre[50]**; il est possible d'utiliser l'élément **Nombre[75]**, même si cet élément n'existe pas. On fera tout simplement comme s'il existait. Dans ce cas, il y aura *peut-être* un message d'erreur apparaissant lors de l'exécution du programme. Notez, enfin, que l'élément **Nombre[50]** n'existe pas lui non plus, puisque ce serait le 51^e élément.

Lors de l'utilisation d'une variable dimensionnée, le programmeur a la responsabilité de s'assurer, en tout temps, que les indices sont dans l'intervalle permis.

EXEMPLE 1

LE PROBLÈME:

On veut lire 3 nombres entiers, puis les afficher en ordre inverse.

QUE VEUT-ON À LA SORTIE?

Les nombres dans l'ordre inverse.

QUELLES SONT LES DONNÉES CONNUES?

Les nombres lus.

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
NB_ELEM	Nombre d'éléments dans la variable Nombre	const int	3
Nombre[NB_ELEM]	Les nombres à lire et à afficher en ordre inverse	int	lus
i	variable de contrôle et indice de la variable Nombre	int	0 à NB_ELEM-1

LE DIAGRAMME D'ACTION:

```
Écrire << "Entrez les ", NB_ELEM, " nombres entiers:"  
┌  
├ Pour i=0, i<NB_ELEM, 1  
│   Écrire "Nombre ", i+1  
└ Lire Nombre[i]  
Écrire "Voici les nombres en ordre inverse:"  
┌  
├ Pour i=NB_ELEM-1, i>=0, -1  
└ Écrire Nombre[i]
```

LE PROGRAMME

```

/*****
AUTEUR: Marcel L'Italien
NOM DU PROGRAMME: 5DIM1.CPP
DESCRIPTION: Lire 3 nombres puis les afficher en ordre inverse.
             Ce programme peut être modifié facilement pour traiter
             autant de nombre que voulu, en changeant la valeur de NB_ELEM.
*****/
#include <iostream>

using namespace std;

void main(void)
{
    const int NB_ELEM = 3;
    int Nombre[NB_ELEM],
        i;

    cout << "Entrez les " << NB_ELEM << " nombres entiers:\n";
    for(i=0; i<NB_ELEM; i++)
    {
        cout << "    Nombre " << (i+1) << ": ";
        cin >> Nombre[i];
    }

    cout << "\nVoici les nombres en ordre inverse:\n\t";
    for(i=NB_ELEM-1; i>=0; i--)
        cout << "\n" << Nombre[i];
}

```

LE RÉSULTAT

```

Entrez les 3 nombres entiers:
    Nombre 1: 50
    Nombre 2: 10
    Nombre 3: 25

Voici les nombres en ordre inverse:

25
10
50

```

EXEMPLE 2

LE PROBLÈME

Calculer la moyenne des 10 nombres dans un vecteur.

QUE VEUT-ON À LA SORTIE?

La moyenne des 10 nombres.

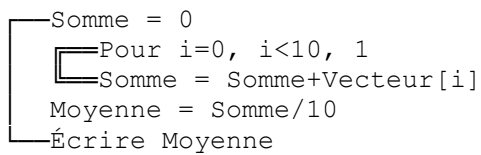
QUELLES SONT LES DONNÉES CONNUES?

Les 10 nombres.

LE DICTIONNAIRE DE DONNÉES

Identificateur	Signification	Type	Valeur
Vecteur[10]	Les 10 nombres pour lesquels on veut la moyenne	int	{5, 6, 8, 10, 2, 78, 35, 56, 99, 88}
Somme	Somme des 10 nombres	int	Somme + Vecteur[i]
i	variable de contrôle et indice de Vecteur	int	0 à 9
Moyenne	Moyenne des 10 nombres	float	Somme/10

LE DIAGRAMME D'ACTION:



LE PROGRAMME

```

/*****
AUTEUR: Marcel L'Italien
NOM DU PROGRAMME: 5DIM2.CPP
DESCRIPTION: Calcul de la moyenne des 10 nombres d'un vecteur.

*****/
#include <iostream>
#include <iomanip>

using namespace std;

void main(void)
{
    int Vecteur[10] = {5, 6, 8, 10, 2, 78, 35, 56, 99, 88},
        Somme,
        i;

    float Moyenne;

    // Calcul de la somme des 10 éléments du vecteur
    Somme = 0;
    for(i=0; i<10; i++)
        Somme += Vecteur[i];

    Moyenne = (float)Somme/10;

    cout << "\nLa moyenne est: " << fixed << setprecision(2) << Moyenne;
}

```

LE RÉSULTAT

EXEMPLE 3

LE PROBLÈME

Trier une liste de 10 nombres en ordre croissant

QUE VEUT-ON À LA SORTIE?

Les 10 nombres en ordre croissant.

QUELLES SONT LES DONNÉES CONNUES?

Les 10 nombres.

LE DICTIONNAIRE DE DONNÉES

Identificateur	Signification	Type	Valeur
MAX	Le nombre d'éléments dans Vecteur	const int	
Vecteur[MAX]	Les nombres à trier	int	{99, 88, 5, 9, 1, 45, 56, 77, 44, 0}
i	variable de contrôle et indice de Vecteur	int	0 à MAX-2
j	variable de contrôle et indice de Vecteur	int	i+1 à MAX-1
k	variable de contrôle et indice de Vecteur	int	0 à MAX-1
Echange	Permet d'échanger les contenus de Vecteur[i] et Vecteur[j]	int	Vecteur[i]

LE DIAGRAMME D'ACTION:

```
Écrire "Voici les nombres initiaux:"
Pour k=0, k<MAX, 1
  Écrire Vecteur[k]
  Pour i=0, i<MAX-1, 1
    Pour j=i+1, j<MAX, 1
      Si Vecteur[i] > Vecteur[j]
        Echange = Vecteur[i]
        Vecteur[i] = Vecteur[j]
        Vecteur[j] = Echange
  Écrire "Voici les nombres après le tri:"
  Pour k=0, k<MAX, 1
    Écrire Vecteur[k]
```

LE PROGRAMME

```

/*****
AUTEUR: Marcel L'Italien
NOM DU PROGRAMME: 5DIM3.CPP
DESCRIPTION: Ce programme place une liste de nombres en ordre croissant.
             La méthode utilisée est le tri à bulles (bubble sort).
*****/
#include <iostream>

using namespace std;

void main(void)
{
    const int MAX=10;
    int Vecteur[MAX] = {99, 88, 5, 9, 1, 45, 56, 77, 44, 0},
        i, j,
        Echange;

    // Affichage des nombres initiaux
    cout << "\nVoici les nombres initiaux:\n\t";
    for(k=0; k<MAX; k++)
        cout << Vecteur[k] << " ";

    /* Tri à bulles(ordre croissant). Cette méthode consiste à comparer un élément
       du vecteur avec les suivants. Lorsque l'élément à comparer est supérieur
       à un nombre qui le suit, c'est que ces nombres sont mal ordonnés l'un par
       rapport à l'autre. Dans ce cas on les inverse (le premier sera le second,
       et le second le premier).
       On répète ce processus pour tous les nombres du vecteur (sauf le dernier).
    */
    for(i=0; i<MAX-1; i++)
        for(j=i+1; j<MAX; j++)
            if(Vecteur[i] > Vecteur[j])
            {
                Echange = Vecteur[i];
                Vecteur[i] = Vecteur[j];
                Vecteur[j] = Echange;
            }

    // Affichage des nombres après le tri
    cout << "\n\nVoici les nombres après le tri:\n\t";
    for(k=0; k<MAX; k++)
        cout << Vecteur[k] << " ";
}

```

LE RÉSULTAT

Voici les nombres initiaux: 99 88 5 9 1 45 56 77 44 0
Voici les nombres après le tri: 0 1 5 9 44 45 56 77 88 99

EXEMPLE 4

LE PROBLÈME

Afficher le nombre de jours qu'il y a dans un mois. Le mois sera lu au clavier.

QUE VEUT-ON À LA SORTIE?

Le nombre de jours dans le mois.

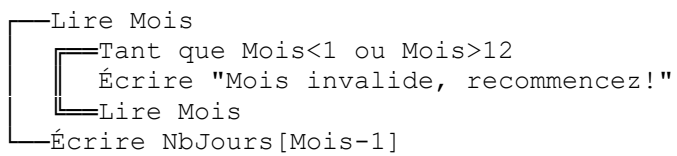
QUELLES SONT LES DONNÉES CONNUES?

Le mois (1 à 12).

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
Mois	Le mois pour lequel on veut connaître le nombre de jours	int	lue
NbJours[12]	Le nombre de jours de chaque mois	int	{31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}

LE DIAGRAMME D'ACTION:



LE PROGRAMME

```

/*****
AUTEUR: Marcel L'Italien
NOM DU PROGRAMME: 5DIM4.CPP
DESCRIPTION: Programme affichant le nombre de jours qu'il y a dans un
             mois (voir aussi l'exemple 4TQ2.CPP).
*****/
#include <iostream>

using namespace std;

void main(void)
{
    int  Mois,
        NbJours[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    cout << "Entrez le mois: ";
    cin >> Mois;

    while(Mois<1 || Mois>12)
    {
        cout << "\nMois invalide, recommencez!\n\n";
        cout << "Entrez le mois: ";
        cin >> Mois;
    }

    cout << "\n\n\tIl y a " << NbJours[Mois-1] << " jours dans ce mois.";
}

```

LES RÉSULTATS

```

Entrez le mois: 0

Mois invalide, recommencez!

Entrez le mois: 2

      Il y a 28 jours dans ce mois.

```


5.4 LES CHÂÎNES DE CARACTÈRES (STRING) ET LES VARIABLES DIMENSIONNÉES.

Une chaîne de caractères définie avec la classe *string* n'est **pas** une variable dimensionnée. Il est quand même possible de référer à un de ses éléments en utilisant un indice.

Comme pour toute variable dimensionnée, l'indice de la variable de type **string** devra être valide, i.e. identifier un élément existant seulement. On pourra s'aider de la fonction *length()* fournie avec cette classe.

Considérez l'exemple suivant:

```

/*****
AUTEUR: Marcel L'Italien
NOM DU PROGRAMME: STRING.CPP
DESCRIPTION: Exemple d'utilisation d'une STRING comme variable dimensionnée
*****/
#include <iostream>
#include <string>

using namespace std;

void main(void)
{
    string Chaine;

    Chaine = "Bonzour";

    cout << "Voici la chaine initiale: " << Chaine;

    Chaine[3] = 'j';

    cout << "\n\nVoici la chaine après correction: " << Chaine;
}

```

RÉSULTAT

Voici la chaine initiale: Bonzour
Voici la chaine après correction: Bonjour

On a fait la correction du 4^e caractère de la chaîne avec l'énoncé **Chaine[3] = 'j';**, qui utilise un indice pour référer à un des éléments de la variable **Chaine**.

Attention: → Il est correct d'accéder à un élément d'une variable de type **string** pour consultation ou modification, mais on ne doit pas ajouter d'élément à cette variable. En fait, on ne doit utiliser que des éléments existants seulement.

Il serait donc incorrect, ici, de faire:

Chaine[7] = '!';

pour y ajouter un caractère. Il faudrait plutôt utiliser la concaténation, réalisée avec l'opérateur +, tel que dans l'exemple suivant: **Chaine = Chaine + '!';**

EXEMPLE 5

LE PROBLÈME

Afficher une chaîne de caractères, et ce caractère par caractère.

QUE VEUT-ON À LA SORTIE?

La chaîne.

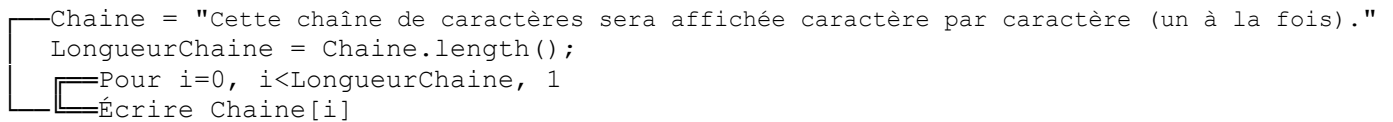
QUELLES SONT LES DONNÉES CONNUES?

La chaîne.

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
Chaine	Chaîne à afficher	string	"Cette chaîne de caractères sera affichée caractère par caractère (un à la fois)."
LongueurChaine	Longueur de Chaine	int	Chaine.length()
i	Variable de contrôle et indice de Chaine	int	0 à LongueurChaine-1

LE DIAGRAMME D'ACTION:



LE PROGRAMME

```
/*
*****
AUTEUR: Marcel L'Italien
NOM DU PROGRAMME: 5DIM5.CPP
DESCRIPTION: Ce programme montre qu'une chaîne de caractères (string) peut
être considérée comme une variable dimensionnée de char.
Ici on affiche une chaîne, caractère par caractère.
*****
*/
#include <iostream>
#include <string>
using namespace std;
void main(void)
{
    string Chaine =
        "Cette chaîne de caractères sera affichée caractère par caractère (un à la fois).";
    int i, LongueurChaine;

    LongueurChaine = Chaine.length();

    for(i=0; i<LongueurChaine; i++)
        cout << Chaine[i];
}
```

LE RÉSULTAT

Cette chaîne de caractères sera affichée caractère par caractère (un à la fois).

EXEMPLE 6

LE PROBLÈME

Compter le nombre de voyelles dans une chaîne lue.

QUE VEUT-ON À LA SORTIE?

Le nombre de voyelles.

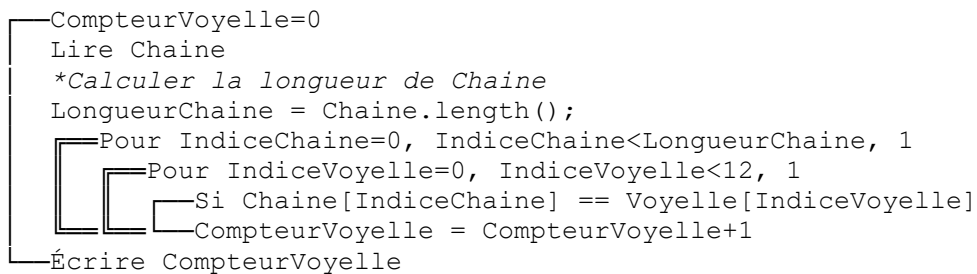
QUELLES SONT LES DONNÉES CONNUES?

La chaîne.

LE DICTIONNAIRE DE DONNÉES

Identificateur	Signification	Type	Valeur
Chaine	Chaîne à lire pour en compter les voyelles	string	lue
Voyelle[12]	les 12 voyelles	const char	{'a', 'e', 'i', 'o', 'u', 'y' 'A', 'E', 'I', 'O', 'U', 'Y'}
CompteurVoyelle	nombre de voyelles dans Chaine	int	CompteurVoyelle + 1
IndiceChaine	indice de Chaine	int	0 à LongueurChaine-1
IndiceVoyelle	indice de Voyelle	int	0 à 11
LongueurChaine	Longueur de Chaine	int	Chaine.length()

LE DIAGRAMME D'ACTION:



LE PROGRAMME

```

/*****
AUTEUR: Marcel L'Italien
NOM DU PROGRAMME: 5dim6.cpp
DESCRIPTION: Compter les voyelles dans une chaîne.
*****/
#include <iostream>
#include <string>

using namespace std;

void main(void)
{
    int IndiceChaine,
        IndiceVoyelle,
        CompteurVoyelle,
        LongueurChaine;

    string Chaine;

    const char Voyelle[12] = { 'a', 'e', 'i', 'o', 'u', 'y',
                               'A', 'E', 'I', 'O', 'U', 'Y' };
                               //Tableau des 12 voyelles

    CompteurVoyelle=0;

    cout << "Taper une chaîne de caractères: ";
    getline(cin, Chaine);

    LongueurChaine = Chaine.length();

    for (IndiceChaine=0; IndiceChaine<LongueurChaine; IndiceChaine++)
        for (IndiceVoyelle=0; IndiceVoyelle<12; IndiceVoyelle++)
            if (Chaine[IndiceChaine] == Voyelle[IndiceVoyelle])
                CompteurVoyelle++;

    cout << "\n\nLe nombre de voyelles est: " << CompteurVoyelle;
}

```

LE RÉSULTAT

Taper une chaîne de caractères: Programmation en C++
Le nombre de voyelles est: 7

EXEMPLE 6A

LE PROBLÈME

Compter le nombre de voyelles dans une chaîne lue.

Même exemple que précédemment, mais ici la variable voyelle est de type *string*.

QUE VEUT-ON À LA SORTIE?

Le nombre de voyelles.

QUELLES SONT LES DONNÉES CONNUES?

La chaîne.

LE DICTIONNAIRE DE DONNÉES

Identificateur	Signification	Type	Valeur
Chaine	Chaîne à lire pour en compter les voyelles	string	lue
Voyelle	chaîne contenant les 12 voyelles	const string	"aeiouyAEIOUY"
CompteurVoyelle	nombre de voyelles dans Chaine	int	CompteurVoyelle + 1
IndiceChaine	indice de Chaine	int	0 à LongueurChaine-1
IndiceVoyelle	indice de Voyelle	int	0 à 11
LongueurChaine	Longueur de Chaine	int	Chaine.length()

LE DIAGRAMME D'ACTION:

```
CompteurVoyelle=0
Lire Chaine
  *Calculer la longueur de Chaine
  LongueurChaine = Chaine.length();
  Pour IndiceChaine=0, IndiceChaine<LongueurChaine, 1
    Pour IndiceVoyelle=0, IndiceVoyelle<12, 1
      Si Chaine[IndiceChaine] == Voyelle[IndiceVoyelle]
        CompteurVoyelle = CompteurVoyelle+1
    Écrire CompteurVoyelle
```

LE PROGRAMME

```

/*****
AUTEUR: Marcel L'Italien
NOM DU PROGRAMME: 5dim6a.cpp
DESCRIPTION: Programme identique au précédent, mais ici on remplace la
             variable dimensionnée Voyelle par une string.
*****/
#include <iostream>
#include <string>

using namespace std;

void main(void)
{
    int IndiceChaine,
        IndiceVoyelle,
        CompteurVoyelle,
        LongueurChaine;

    string Chaine;

    string Voyelle = "aeiouyAEIOUY"; //Chaîne composée des 12 voyelles

    CompteurVoyelle=0;

    cout << "Taper une chaîne de caractères: ";
    getline(cin, Chaine);

    LongueurChaine = Chaine.length();

    for (IndiceChaine=0; IndiceChaine<LongueurChaine; IndiceChaine++)
        for (IndiceVoyelle=0; IndiceVoyelle<12; IndiceVoyelle++)
            if (Chaine[IndiceChaine] == Voyelle[IndiceVoyelle])
                CompteurVoyelle++;

    cout << "\n\nLe nombre de voyelles est: " << CompteurVoyelle;
}

```

LE RÉSULTAT

Taper une chaîne de caractères: Programmation en C++

Le nombre de voyelles est: 7

EXEMPLE 7

LE PROBLÈME

Trouver et afficher le prix unitaire d'un produit à partir du nom du produit.

QUE VEUT-ON À LA SORTIE? Le prix du produit.

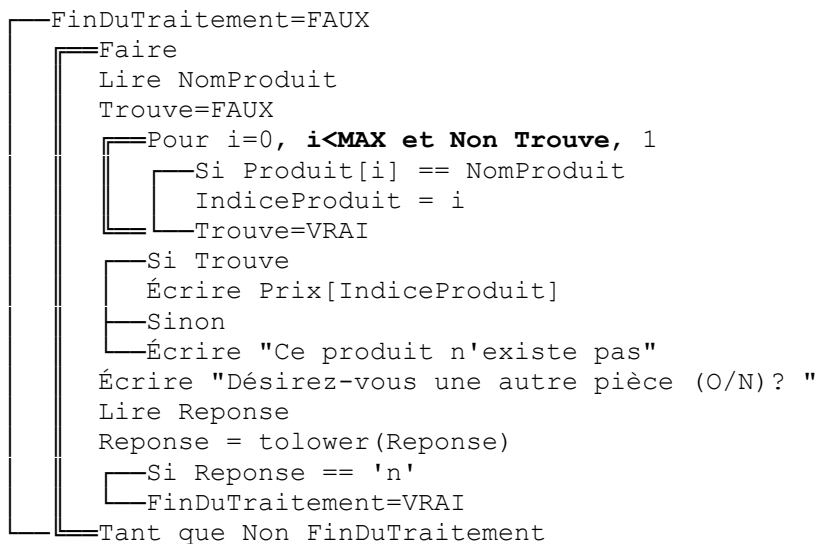
QUELLES SONT LES DONNÉES CONNUES?

Le nom des produits, leur prix, et le nom du produit pour lequel on cherche le prix

LE DICTIONNAIRE DE DONNÉES

Identificateur	Signification	Type	Valeur
FinDuTraitement	indique la fin du programme	int	VRAI ou FAUX
i	variable de contrôle et indice de Produit[]	int	0 à MAX-1
IndiceProduit	indice du produit recherché	int	i
NomProduit	nom du produit dont on veut le prix	string	lue
Prix[MAX]	les prix de tous les produits	float	{ 1.85, 2.25, 8.99, 0.99, 1.99, 750, 6.66, 6.66, 6.66, 60 }
Produit[MAX]	les noms de tous les produits	string	{ "Pain", "Lait", "Bière rousse", "Macaroni au fromage", "Croustilles", "Freins ABS", "Lotion à barbe", "Lotion contre l'acné", "Lotion solaire", "Lotion scolaire" }
Reponse	Reponse de l'utilisateur	char	lue
Trouve	Indique si NomProduit existe dans Produit[]	int	VRAI ou FAUX

LE DIAGRAMME D'ACTION:



LE PROGRAMME

```

/*****
AUTEUR: Marcel L'Italien
NOM DU PROGRAMME: 5dim7.cpp
DESCRIPTION: Recherche du prix unitaire d'un produit en inventaire.
             La recherche se fera à partir du nom du produit.
*****/
#include <iostream>
#include <string>
#include <conio.h>

#include "c:\cvm.h"

using namespace std;

void main(void)
{
    const int MAX = 10,
              VRAI = 1, FAUX = 0;
    char Reponse;

    int i, IndiceProduit,
        FinDuTraitement,
        Trouve;

    string NomProduit,
        Produit[MAX]= { "Pain", "Lait", "Bière rousse",
                        "Macaroni au fromage", "Croustilles",
                        "Freins ABS", "Lotion à barbe",
                        "Lotion contre l'acné",
                        "Lotion solaire", "Lotion scolaire"
                      };

    float Prix[MAX]={1.85, 2.25, 8.99, 0.99, 1.99, 750, 6.66, 6.66, 6.66, 60};

    FinDuTraitement=FAUX;
    do
    {
        clrscr();
        cout << "Quel est le produit désiré: ";
        getline(cin, NomProduit);

        Trouve = FAUX;
        for (i=0; i<MAX && !Trouve; i++)
            if (Produit[i] == NomProduit)
            {
                IndiceProduit = i;
                Trouve = VRAI;
            }
        if (Trouve)
            cout << "\nLe prix unitaire est de: " << Prix[IndiceProduit] << "$";
        else
            cout << "\nCe produit n'existe pas";
    }
}
```



```

gotoxy(1,10);

cout << "Désirez-vous une autre pièce (O/N)? ";
Reponse = tolower(_getche());

if (Reponse == 'n')
    FinDuTraitement=VRAI;
}
while (!FinDuTraitement);
}

```

LE RÉSULTAT

Quel est le produit désiré: Macaroni au fromage

Le prix unitaire est de: 0.99\$

Désirez-vous une autre pièce (O/N)? N

5.5 LES VARIABLES MULTIDIMENSIONNELLES

Jusqu'ici, nous avons utilisé des variables dimensionnées à une seule dimension. Pour référer à un élément, nous utilisons donc un seul indice. Il existe des variables dimensionnées ayant plus d'une dimension. En fait, il peut y en avoir autant que désiré, et donc il y aura plusieurs indices pour un seul élément.

Prenons le cas de l'étudiant d'un cours. Celui-ci pourrait, par exemple, avoir 5 notes pour ce cours. Nous pourrions définir une variable pour stocker ces notes:

```
int NoteEtud[5];
```

Si nous voulions stocker les notes de tous les étudiants du cours (25 étudiants), nous pourrions définir les variables suivantes:

```
int NoteEtud1[5],  
    NoteEtud2[5],  
    ...  
    NoteEtud25[5];
```

Cette façon de faire est lourde et pose les mêmes problèmes qui nous ont amené à utiliser des variables dimensionnées. Pour ne pas avoir à manipuler une variable différente pour chaque étudiant, nous allons les regrouper dans une seule variable à deux dimensions. Il y aura une dimension pour chaque étudiant de la classe et une autre pour chaque note. Voici la définition de cette variable:

```
int Note[25][5]; // création de 25 étudiants ayant chacun 5 notes
```

Pour référer à une note en particulier il faudra spécifier 2 indices. Le premier identifiera l'étudiant, et le second la note.

Une variable à deux dimensions est composée de rangées et de colonnes (telle une matrice). Le premier indice représentera toujours la rangée (ligne), et le second la colonne. L'élément **Note[4][2]** est donc l'élément de la cinquième rangée et de la troisième colonne, i.e. la troisième note du cinquième étudiant. Rappelons-nous qu'un indice est un déplacement par rapport au début, et donc que le premier élément de la variable **Note** est référé par **Note[0][0]**.

Représentation de la variable **Note**:

		No Examen→	1	2	3	4	5
↓No Etudiant	INDICES	0	1	2	3	4	
1	0	34	67	88	67	78	
2	1	56	99	45	77	65	
3	2	74	67	78	89	69	
...	...						
24	23	66	67	78	78	80	
25	24	83	63	58	59	78	

Exemples:

Note[3][4] → nous donne la 5^e note du 4^e étudiant
 Note[24][0] → nous donne la 1^{re} note du 25^e étudiant

EXEMPLE 8

LE PROBLÈME

On veut faire la somme de deux matrices.

QUE VEUT-ON À LA SORTIE?

Les deux matrices originales et leur somme.

QUELLES SONT LES DONNÉES CONNUES?

Les deux matrices.

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
NB_LIG	le nombre de lignes	const int	3
NB_COL	le nombre de colonnes	const int	4
MatA[NB_LIG][NB_COL]	la première matrice	int	<i>voir le programme</i>
MatB[NB_LIG][NB_COL]	la seconde matrice	int	<i>voir le programme</i>
MatC[NB_LIG][NB_COL]	La somme des deux matrices	int	$\text{MatC}[i][j] = \text{MatA}[i][j] + \text{MatB}[i][j]$
Ligne	Indice représentant la ligne	int	0 à NB_LIG-1
Col	Indice représentant la colonne	int	0 à NB_COL-1

LE DIAGRAMME D'ACTION

```
*Calcul de la somme de MatA et MatB. Le résultat est mis dans MatC.
┌ Pour Ligne=0, Ligne<NB_LIG, 1
├   ┌ Pour Col=0; Col<NB_COL, 1
├   │   MatC[Ligne][Col] = MatA[Ligne][Col] + MatB[Ligne][Col]
├   └
└ *Affichage des trois matrices
    ┌ Pour Ligne=0, Ligne<NB_LIG, 1
    │   Écrire "\n"
    │   ┌ Pour Col=0, Col<NB_COL, 1
    │   │   Écrire MatA[Ligne][Col]
    │   └
    │   ┌ Pour Ligne=0, Ligne<NB_LIG, 1
    │   │   Écrire "\n"
    │   │   ┌ Pour Col=0, Col<NB_COL, 1
    │   │   │   Écrire MatB[Ligne][Col]
    │   │   └
    │   └ Pour Ligne=0, Ligne<NB_LIG, 1
    │       Écrire "\n"
    │       ┌ Pour Col=0, Col<NB_COL, 1
    │       │   Écrire MatC[Ligne][Col]
```

LE PROGRAMME

```

/*****
AUTEUR: Marcel L'Italien
NOM DU PROGRAMME: 5dim8.cpp
DESCRIPTION: Addition de deux matrices.
*****/
#include <iostream>
#include <iomanip>
using namespace std;
void main(void)
{
    const int NB_LIG=3,
              NB_COL=4;
    int MatA[NB_LIG][NB_COL] =
        {
            { 1, 2, 3, 4},
            { 5, 6, 7, 8},
            { 9, 10, 11, 12}
        };
    int MatB[NB_LIG][NB_COL] =
        {
            { 1, 2, 3, 4},
            { 5, 6, 7, 8},
            { 9, 10, 11, 12}
        };
    int MatC[NB_LIG][NB_COL],
        Ligne,
        Col;

    // Calcul de la somme de MatA et MatB. Le résultat est mis dans MatC.
    for(Ligne=0; Ligne<NB_LIG; Ligne++)
        for(Col=0; Col<NB_COL; Col++)
            MatC[Ligne][Col] = MatA[Ligne][Col] + MatB[Ligne][Col];
    // Affichage des trois matrices
    cout << "Matrice A:\n";
    for(Ligne=0; Ligne<NB_LIG; Ligne++)
    {
        cout << "\n";
        for(Col=0; Col<NB_COL; Col++)
            cout << setw(4) << MatA[Ligne][Col];
    }
    cout << "\n\nMatrice B:\n";
    for(Ligne=0; Ligne<NB_LIG; Ligne++)
    {
        cout << "\n";
        for(Col=0; Col<NB_COL; Col++)
            cout << setw(4) << MatB[Ligne][Col];
    }
    cout << "\n\n\nMatrice C (somme des deux matrices):\n";
    for(Ligne=0; Ligne<NB_LIG; Ligne++)
    {
        cout << "\n";
        for(Col=0; Col<NB_COL; Col++)
            cout << setw(4) << MatC[Ligne][Col];
    }
}

```

LES RÉSULTATS

Matrice A:

1	2	3	4
5	6	7	8
9	10	11	12

Matrice B:

1	2	3	4
5	6	7	8
9	10	11	12

Matrice C (somme des deux matrices):

2	4	6	8
10	12	14	16
18	20	22	24

EXEMPLE 9

LE PROBLÈME

On veut faire la multiplication de deux matrices.

QUE VEUT-ON À LA SORTIE?

Les deux matrices originales et leur produit.

QUELLES SONT LES DONNÉES CONNUES?

Les deux matrices.

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
MatA[3][4]	la première matrice	int	<i>voir le programme</i>
MatB[4][5]	la seconde matrice	int	<i>voir le programme</i>
MatC[3][5]	Le produit des deux matrices	int	$\text{MatC}[\text{Ligne}][\text{Col}] = \sum_{i=0}^3 \text{MatA}[\text{Ligne}][i] * \text{MatB}[i][\text{Col}]$
i	indice	int	0 à 3
Ligne	Indice représentant la ligne	int	0 à 2 ou 0 à 3
Col	Indice représentant la colonne	int	0 à 4 ou 0 à 3

LE DIAGRAMME D'ACTION

```
*Calcul de la somme de MatA et MatB. Le résultat est mis dans MatC.
┌─ Pour Ligne=0, Ligne<3, 1
│   ┌─ Pour Col=0; Col<5, 1
│   │   MatC[Ligne][Col]=0
│   │   ┌─ Pour i=0, i<4, 1
│   │   │   MatC[Ligne][Col] = MatC[Ligne][Col] + (MatA[Ligne][i] * MatB[i][Col])
│   └─
└─

*Affichage des trois matrices
┌─ Pour Ligne=0, Ligne<3, 1
│   Écrire "\n"
│   ┌─ Pour Col=0, Col<4, 1
│   │   Écrire MatA[Ligne][Col]
│   └─
┌─ Pour Ligne=0, Ligne<4, 1
│   Écrire "\n"
│   ┌─ Pour Col=0, Col<5, 1
│   │   Écrire MatB[Ligne][Col]
│   └─
┌─ Pour Ligne=0, Ligne<3, 1
│   Écrire "\n"
│   ┌─ Pour Col=0, Col<5, 1
│   │   Écrire MatC[Ligne][Col]
│   └─
```

LE PROGRAMME

```

/*****
AUTEUR: Marcel L'Italien
NOM DU PROGRAMME: 5dim9.cpp
DESCRIPTION: Multiplication de deux matrices.
*****/

#include <iostream>
#include <iomanip>

using namespace std;

void main(void)
{
    int MatA[3][4] =
        {
            { 1, 2, 3, 4},
            { 5, 6, 7, 8},
            { 9, 10, 11, 12}
        };

    int MatB[4][5] =
        {
            { 1, 2, 3, 4, 5},
            { 6, 7, 8, 9, 10},
            {11, 12, 13, 14, 15},
            {16, 17, 18, 19, 20}
        };

    int MatC[3][5],
        Ligne,
        Col,
        i;

    // Multiplication de MatA par MatB. Le résultat est mis dans MatC.
    for(Ligne=0; Ligne<3; Ligne++)
        for(Col=0; Col<5; Col++)
        {
            MatC[Ligne][Col] = 0;
            for(i=0; i<4; i++)
                MatC[Ligne][Col] += MatA[Ligne][i]*MatB[i][Col];
        }

    // Affichage des trois matrices
    cout << "Matrice A:\n";
    for(Ligne=0; Ligne<3; Ligne++)
    {
        cout << "\n";
        for(Col=0; Col<4; Col++)
            cout << setw(4) << MatA[Ligne][Col];
    }
}
```



```

cout << "\n\nMatrice B:\n";
for(Ligne=0; Ligne<4; Ligne++)
{
    cout << "\n";
    for(Col=0; Col<5; Col++)
        cout << setw(4) << MatB[Ligne][Col];
}

cout << "\n\n\nMatrice C (multiplication des deux matrices):\n";

for(Ligne=0; Ligne<3; Ligne++)
{
    cout << "\n";
    for(Col=0; Col<5; Col++)
        cout << setw(5) << MatC[Ligne][Col];
}
}

```

LES RÉSULTATS

Matrice A:

1	2	3	4
5	6	7	8
9	10	11	12

Matrice B:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

Matrice C (multiplication des deux matrices):

110	120	130	140	150
246	272	298	324	350
382	424	466	508	550

5.6 EXERCICES

1. Écrire les définitions des variables dimensionnées permettant de stocker les valeurs suivantes:

SÉRIE 1

- a) la température maximum de chaque jour d'une semaine
- b) la note finale de tous vos cours
- c) le nom de chaque étudiant de la classe
- d) la température maximum de chaque jour d'un mois

SÉRIE 2

- a) le nom de chaque mois de l'année et le nombre de jour de chacun
- b) les températures minimum et maximum de chaque jour d'une semaine
- c) les températures minimum et maximum de chaque jour d'un mois
- d) les températures minimum et maximum de chaque jour de l'année
- e) les températures minimum et maximum des 25 dernières années

2. Soient les déclarations suivantes:

int A[5], B[2][3];

string C, D[2];

float E[5], F[2][3];

Si, lors de l'exécution du programme, les adresses suivantes ont été attribuées à chacune:

A 8000

B 8010

C 8030

D 8050

E 8100

F 8200

Évaluer l'adresse de chacun des éléments suivants:

SÉRIE 1

a) A[2]

b) C[4]

c) E[1]

SÉRIE 2

a) B[4][1]

b) D[0][1]

c) F[1][3]

3. On veut additionner la constante 10 à chaque élément d'un tableau.

QUE VEUT-ON À LA SORTIE?

Le contenu du tableau avant et après l'addition.

QUELLES SONT LES DONNÉES CONNUES?

Les éléments du tableau.

LES ÉCRANS:

Le contenu du tableau avant l'addition:

3 5 7 9 11

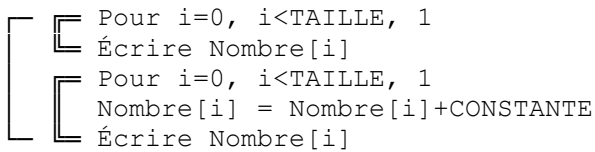
Le contenu du tableau après l'addition:

13 15 17 19 21

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
CONSTANTE	valeur à ajouter à Nombre[i]	const int	10
TAILLE	taille du tableau	const int	5
Nombre[TAILLE]	le tableau à traiter	int	3,5,7,9,11
i	indice	int	0 à TAILLE-1

LE DIAGRAMME D'ACTION:



LE PROGRAMME:

→ à compléter

4. On doit lire, à partir du clavier, les six notes d'un étudiant, en les plaçant dans un vecteur. On doit en faire la moyenne et imprimer les notes et la moyenne.

QUE VEUT-ON À LA SORTIE?

Les six notes et la moyenne.

QUELLES SONT LES DONNÉES CONNUES?

Les six notes.

LES ÉCRANS:

Donner la note numéro 1: xx

Donner la note numéro 2: xx

...

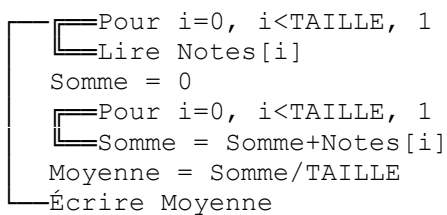
Donner la note numéro 6: xx

La moyenne est: xx.x

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
i	indice de Notes[]	int	0 à TAILLE-1
Moyenne	la moyenne des notes	float	Somme/TAILLE
Notes[TAILLE]	les notes de l'étudiant	float	lue à l'entrée
Somme	la somme des notes	float	TAILLE-1 $\sum_{i=0} \text{Notes}[i]$
TAILLE	nombre de notes	const int	6

LE DIAGRAMME D'ACTION:



LE PROGRAMME:

→ à compléter

5. On lit au clavier deux séries de 10 nombres. Chaque série est placée dans un vecteur. On désire faire le produit des deux vecteurs. Le produit consiste à multiplier le premier élément du premier vecteur avec le premier élément du deuxième vecteur, et placer le résultat dans le premier élément d'un troisième vecteur, et ainsi de suite.

QUE VEUT-ON À LA SORTIE?

Les valeurs du premier vecteur, les valeurs du deuxième vecteur et les valeurs du vecteur résultat.

QUELLES SONT LES DONNÉES CONNUES?

Les valeurs du premier vecteur et du deuxième vecteur.

LES ÉCRANS:

Entrez 10 nombres séparés par un espace (premier vecteur):

xx xx xx xx xx xx xx xx xx xx

Entrez 10 nombres séparés par un espace (deuxième vecteur):

xx xx xx xx xx xx xx xx xx xx

Les résultats sont:

Le premier vecteur

xx xx xx xx xx xx xx xx xx xx

Le deuxième vecteur:

xx xx xx xx xx xx xx xx xx xx

Le vecteur du produit:

xx xx xx xx xx xx xx xx xx xx

LE DICTIONNAIRE DE DONNÉES:

Identificateur	Signification	Type	Valeur
i	indice	int	0 à MAX-1
MAX	nombre de valeurs dans les vecteurs	const int	3
Vect1[MAX]	premier vecteur	int	lue à l'entrée
Vect2[MAX]	deuxième vecteur	int	lue à l'entrée
Produit[MAX]	produit des deux vecteurs (Vect1 et Vect2)	int	Produit[i] = Vect1[i] * Vect2[i]

LE DIAGRAMME D'ACTION:

→ à compléter

LE PROGRAMME:

→ à compléter

6. On lit la température maximale et minimale journalière d'une semaine donnée. On doit calculer la température moyenne journalière ainsi que la température moyenne de la semaine.

QUE VEUT-ON À LA SORTIE?

La température minimale de chaque jour.
La température maximale de chaque jour.
La température moyenne de chaque jour.
La température moyenne de la semaine.

QUELLES SONT LES DONNÉES CONNUES?

La température minimale de chaque jour.
La température maximale de chaque jour.

LES ÉCRANS:

Entrez les températures min. et max. séparées par un espace

Jour 1: xx xx

...

Jour 7: xx xx

JOURS	MIN	MAX	MOYENNE
1	xx.x	xx.x	xx.x

...

7	xx.x	xx.x	xx.x
---	------	------	------

La température moyenne de la semaine est de: xx.x

LE DICTIONNAIRE DE DONNÉES: → à compléter

Identificateur	Signification	Type	Valeur
Ligne	indice	int	
Col	indice	int	
Jours[7][3]	table contenant les diverses températures pour chaque jour de la semaine colonne 0: température minimum colonne 1: température maximum colonne 2: température moyenne	float	
Somme	somme des moyennes de chaque jour	float	
MoyenneSemaine	la moyenne de la semaine	float	

LE DIAGRAMME D'ACTION:

→ à compléter

LE PROGRAMME:

→ à compléter

5.7 Exercices sur les vecteurs

Donnez le contenu des deux vecteurs après chaque question. Utilisez toujours les valeurs initiales.

VectA :

34	3	21	8	16
----	---	----	---	----

VectB :

12	15	9	2	26
----	----	---	---	----

```
const int MAX = 5;
int VectA[MAX], VectB[MAX], i, Echange;
```

- 1) for(i=0; i<MAX; i++)
 VectA[i] = VectB[i];

- 2) for(i=0; i<MAX; i++)
 VectA[i] = VectB[4-i];

- 3) for(i=10; i<10+MAX; i++)
 VectA[(i-(2*5))] += VectB[-20-(i-10)+i*2];

- 4) for(i=0; i<MAX; i++)
 {
 Echange = VectA[4-i];
 VectA[4-i] = VectB[i];
 VectB[i] = Echange;
 }

- 5) for(i=0; i<MAX; i++)
 {
 if(VectA[i]>VectB[i])
 {
 Echange = VectA[4-i];
 VectA[4-i] = VectB[i];
 VectB[i] = Echange;
 }
 }

- 6) for(i=MAX-1; i>=0; i--)
 cin >> VectA[i];

Voici les valeurs entrées au clavier : 10 5 15 35 20

5.8 Exercices sur les matrices

Donnez le contenu des deux matrices après chaque question :

```
const int NB_LIGNES=4, NB_COL=4;
```

```
int MatA[NB_LIGNES][NB_COL],  
    MatB[NB_LIGNES][NB_COL],  
    Ligne, Col;
```

```
1) for(Ligne=0; Ligne<NB_LIGNES; Ligne++)  
    for(Col=0; Col<NB_COL; Col++)  
        MatA[Ligne][Col]=Ligne+Col;
```

MatA


```
2) for(Ligne=0; Ligne<NB_LIGNES; Ligne++)  
    for(Col=0; Col<NB_COL; Col++)  
    {  
        MatA[Ligne][Col] = (2*Ligne)+Col;  
        MatB[Col][Ligne] = MatA[Ligne][Col];  
    }
```

MatA

MatB


```
3) for(Ligne=0; Ligne<NB_LIGNES; Ligne++)  
    for(Col=0; Col<NB_COL; Col++)  
    {  
        MatA[Ligne][Col] = Ligne+Col;  
        MatB[3-Col][Ligne] = MatA[Ligne][Col];  
    }
```

MatA

MatB

Opérateurs du C++

Opérateur	Priorité	Sens	Nombre. Opérandes	Nom	Description
()	15	→	varie	Appel de fonction ou Parenthèses	Appelle la fonction dont le nom se trouve devant les parenthèses, avec les paramètres contenus dans celles-ci. ou Parenthèses afin de modifier l'ordre d'exécution.
[]	15	→	2	Indice de tableau	Renvoie un élément du tableau dont le nom est indiqué devant les [], selon le déplacement indiqué entre crochets.
->	15	→	2	Sélection de membre via pointeur	Identifie un membre de l'objet pointé par le pointeur dont le nom précède le -> .
::	15	→	1 ou 2	Résolution de portée	En opérateur unaire, devant un identificateur, indique une variable globale. En opérateur binaire, limite la portée de recherche de l'identificateur situé à droite de l'opérateur à la classe située à gauche.
.	15	→	2	Sélection de membre	Identifie un membre d'un objet dont le nom précède le point (.)
!	14	←	1	Négation logique	Change un booléen en son opposé.
~	14	←	1	Négation par bits	Change tous les bits d'un entier en leur opposé.
+	14	←	1	+ unaire	Aucun effet.
-	14	←	1	- unaire	Change un nombre en son opposé.
++	14	←	1	Incrémentation	Incrémente un entier ou un pointeur. Peut être placé devant ou derrière (action avant ou après le reste des calculs).
--	14	←	1	Décrémentation	Décrémente un entier ou un pointeur. Peut être placé devant ou derrière (action avant ou après le reste des calculs).
&	14	←	1	Adresse	Renvoie l'adresse de l'identificateur dont le nom suit.
*	14	←	1	Indirection (Déréférenciation)	Renvoie la valeur pointée par le pointeur dont le nom suit.
(type)	14	←	1	Changement de type (cast)	Change le type d'une expression en celui indiqué entre parenthèses.
sizeof	14	←	1	Taille	Taille du type donné en argument, ou du type de la variable donnée en argument.

new	14	←	1	Allocation de mémoire dynamique	Crée un bloc mémoire de taille adéquat pour stocker un objet, et renvoie un pointeur sur ce bloc.
delete	14	←	1	Désaffectation de mémoire dynamique	Détruit un bloc créé par new .
.*	13	→	2	Pointeur vers un membre	Pointe sur un membre d'un objet dont le nom précède le point (.)
->*	13	→	2	Pointeur vers un membre via pointeur	Pointe sur un membre de l'objet pointé par le pointeur dont le nom précède le -> .
*	12	→	2	Multiplication	Multiplie les deux nombres.
/	12	→	2	Division	Divise le nombre de gauche par celui de droite.
%	12	→	2	Reste (modulo)	Donne le reste de la division de deux entiers (division entière)
+	11	→	2	Addition	Additionne les deux nombres.
-	11	→	2	Soustraction	Soustrait les deux nombres.
<<	10	→	2	Décalage à droite ou Écrire dans <i>cout</i>	Décale à droite les bits de l'entier qui précède du nombre de bits indiqué par l'entier suivant l'opérateur, divisant ainsi par une puissance de 2. ou Opérateur surchargé dans les flux (ex: écrire dans le flux <i>cout</i>).
>>	10	→	2	Décalage à gauche ou Lire du flux <i>cin</i>	Décale à gauche les bits de l'entier précédent du nombre de bits indiqué par l'entier suivant l'opérateur, multipliant ainsi par une puissance de 2. ou Opérateur surchargé dans les flux (ex: lire dans le flux <i>cin</i>).
<	9	→	2	Inférieur strict	Renvoie un booléen indiquant si le nombre de gauche est strictement inférieur à celui de droite.
<=	9	→	2	Inférieur large	Renvoie un booléen indiquant si le nombre de gauche est inférieur ou égal à celui de droite.
>	9	→	2	Supérieur strict	Renvoie un booléen indiquant si le nombre de gauche est strictement supérieur à celui de droite.
>=	9	→	2	Supérieur large	Renvoie un booléen indiquant si le nombre de gauche est supérieur ou égal à celui de droite.

==	8	→	2	Égal	Renvoie un booléen indiquant si les deux termes sont égaux.
!=	8	→	2	Différent	Renvoie un booléen indiquant si les deux termes sont différents.
&	7	→	2	ET par bits	Applique l'opération logique ET sur les bits des opérandes entiers. Ne pas utiliser avec les booléens (voir &&).
^	6	→	2	OU EXCLUSIF par bits	Applique l'opération logique OU EXCLUSIF sur les bits des opérandes entiers.
 	5	→	2	OU par bits	Applique l'opération logique OU sur les bits des opérandes entiers. Ne pas utiliser avec les booléens (voir).
&&	4	→	2	ET logique	ET logique entre deux booléens. Si le premier opérande est faux, le second n'est pas évalué.
 	3	→	2	OU logique	OU logique entre deux booléens. Si le premier opérande est vrai, le second n'est pas évalué.
? :	2	←	3	Selon que	Évalue le booléen situé devant ? . Si le résultat est vrai renvoie le terme précédant : , sinon renvoie celui qui suit.
=	1	←	2	Affectation	Calcule le terme de droite et place la valeur dans la variable désigné à gauche. Renvoie la valeur obtenue.
*= /= %= += -= &= ^= = <<= >>=	1	←	2	Affectation + opération	Effectue l'opération dont le symbole précède le signe = , entre la variable identifiée à gauche et le terme situé à droite, puis place le résultat dans cette variable. Renvoie la valeur obtenue.
,	0	→	2	Succession	Évalue le terme de gauche, puis celui de droite et renvoie ce dernier.

Tables des caractères ASCII

Num	Hexa	Valeur									
0	0x0	NUL (rien)	32	0x20	Espace	64	0x40	@	96	0x60	`
1	0x1	SOH	33	0x21	!	65	0x41	A	97	0x61	a
2	0x2	STX	34	0x22	"	66	0x42	B	98	0x62	b
3	0x3	ETX	35	0x23	#	67	0x43	C	99	0x63	c
4	0x4	EOT	36	0x24	\$	68	0x44	D	100	0x64	d
5	0x5	ENQ	37	0x25	%	69	0x45	E	101	0x65	e
6	0x6	ACK	38	0x26	&	70	0x46	F	102	0x66	f
7	0x7	BEL (bip sonore)	39	0x27	'	71	0x47	G	103	0x67	g
8	0x8	BS	40	0x28	(72	0x48	H	104	0x68	h
9	0x9	TAB (tabulation)	41	0x29)	73	0x49	I	105	0x69	i
10	0xa	LF	42	0x2a	*	74	0x4a	J	106	0x6a	j
11	0xb	VT	43	0x2b	+	75	0x4b	K	107	0x6b	k
12	0xc	FF	44	0x2c	,	76	0x4c	L	108	0x6c	l
13	0xd	CR (n. ligne)	45	0x2d	-	77	0x4d	M	109	0x6d	m
14	0xe	SO	46	0x2e	.	78	0x4e	N	110	0x6e	n
15	0xf	SI (entrée)	47	0x2f	/	79	0x4f	O	111	0x6f	o
16	0x10	DLE	48	0x30	0	80	0x50	P	112	0x70	p
17	0x11	DC1	49	0x31	1	81	0x51	Q	113	0x71	q
18	0x12	DC2	50	0x32	2	82	0x52	R	114	0x72	r
19	0x13	DC3	51	0x33	3	83	0x53	S	115	0x73	s
20	0x14	DC4	52	0x34	4	84	0x54	T	116	0x74	t
21	0x15	NAK	53	0x35	5	85	0x55	U	117	0x75	u
22	0x16	SYN	54	0x36	6	86	0x56	V	118	0x76	v
23	0x17	ETB	55	0x37	7	87	0x57	W	119	0x77	w
24	0x18	CAN	56	0x38	8	88	0x58	X	120	0x78	x
25	0x19	EM	57	0x39	9	89	0x59	Y	121	0x79	y
26	0x1a	SUB	58	0x3a	:	90	0x5a	Z	122	0x7a	z
27	0x1b	ESC (escape)	59	0x3b	;	91	0x5b	[123	0x7b	{
28	0x1c	FS	60	0x3c	<	92	0x5c	\	124	0x7c	
29	0x1d	GS	61	0x3d	=	93	0x5d]	125	0x7d	}
30	0x1e	RS	62	0x3e	>	94	0x5e	^	126	0x7e	~
31	0x1f	US	63	0x3f	?	95	0x5f	_	127	0x7f	DEL (efface)

Tables des caractères ASCII étendus

DEC	CAR	OCT	HEX	DEC	CAR	OCT	HEX	DEC	CAR	OCT	HEX
127:	␣	177	7f	128:	Ç	200	80	129:	ü	201	81
130:	é	202	82	131:	à	203	83	132:	ä	204	84
133:	à	205	85	134:	ä	206	86	135:	ç	207	87
136:	ê	210	88	137:	ë	211	89	138:	è	212	8a
139:	ï	213	8b	140:	î	214	8c	141:	ì	215	8d
142:	Ä	216	8e	143:	Å	217	8f	144:	É	220	90
145:	æ	221	91	146:	Æ	222	92	147:	ô	223	93
148:	ö	224	94	149:	ò	225	95	150:	û	226	96
151:	ù	227	97	152:	ÿ	230	98	153:	Ö	231	99
154:	Ü	232	9a	155:	ø	233	9b	156:	ƒ	234	9c
157:	Ø	235	9d	158:	×	236	9e	159:	ƒ	237	9f
160:	á	240	a0	161:	í	241	a1	162:	ó	242	a2
163:	ú	243	a3	164:	ñ	244	a4	165:	Ñ	245	a5
166:	ª	246	a6	167:	º	247	a7	168:	¿	250	a8
169:	®	251	a9	170:	¬	252	aa	171:	½	253	ab
172:	¼	254	ac	173:	¡	255	ad	174:	<<	256	ae
175:	>>	257	af	176:	⋮	260	b0	177:	⌘	261	b1
178:	⌘	262	b2	179:		263	b3	180:	¡	264	b4
181:	À	265	b5	182:	Á	266	b6	183:	Â	267	b7
184:	©	270	b8	185:	⌋	271	b9	186:		272	ba
187:	™	273	bb	188:	⌋	274	bc	189:	¢	275	bd
190:	¥	276	be	191:	¬	277	bf	192:	£	300	c0
193:	±	301	c1	194:	†	302	c2	195:	‡	303	c3
196:	–	304	c4	197:	‡	305	c5	198:	ä	306	c6
199:	Ã	307	c7	200:	⌋	310	c8	201:	ƒ	311	c9
202:	⌋	312	ca	203:	⌋	313	cb	204:	ƒ	314	cc
205:	=	315	cd	206:	⌋	316	ce	207:	α	317	cf
208:	Ð	320	d0	209:	Ð	321	d1	210:	Ê	322	d2
211:	Ë	323	d3	212:	Ê	324	d4	213:	¬	325	d5
214:	Í	326	d6	215:	Î	327	d7	216:	Ï	330	d8
217:		331	d9	218:		332	da	219:	■	333	db
220:	■	334	dc	221:		335	dd	222:		336	de
223:		337	df	224:		340	e0	225:	ß	341	e1
226:		342	e2	227:		343	e3	228:		344	e4
229:		345	e5	230:	μ	346	e6	231:	þ	347	e7
232:		350	e8	233:		351	e9	234:		352	ea
235:		353	eb	236:	ý	354	ec	237:		355	ed
238:		356	ee	239:		357	ef	240:	–	360	f0
241:	±	361	f1	242:		362	f2	243:	¾	363	f3
244:		364	f4	245:		365	f5	246:	÷	366	f6
247:		367	f7	248:		370	f8	249:	÷	371	f9
250:		372	fa	251:		373	fb	252:	³	374	fc
253:		375	fd	254:		376	fe	255:		377	ff