ULCO-L1-Algo1-CM : Initiation à l'algorithmique avec le langage C++

Table des matières

1 Généralités

Avertissement : ce texte ne constitue pas un manuel, ni abrégé, ni encore moins exhaustif, du langage C++. Son objectif se limite à introduire les composants élémentaires du langage, jugés nécessaires à apprendre les bases de l'algorithmique et de la programmation.

1.1 Définitions/Glossaire

Instruction :
Opération élémentaire réalisable par un ordinateur (calcul, affichage...).

Donnée :
Valeur (numérique, alphabétique...) utilisée dans une instruction.

Variable :
Emplacement mémoire, identifié par un nom, destiné à contenir une donnée. Dans beaucoup de langages informatiques, les variables sont **typées** : une variable typée ne peut contenir qu'un seul type (une seule sorte) de données : par exemple que des entiers, ou que des réels, ou que des caractères...

Constante :
Donnée exprimée littéralement ("en toutes lettres"), exemples : 12 ; 3.14159 ; 'A' ; "bonjour". Une constante peut être associée dans un programme à un nom (comme PI).

Structure de contrôle :
Mot-clé du langage qui contrôle l'exécution d'un ensemble d'instructions.

Structure de données :
Structure logique qui permet de définir des variables agrègeant plusieurs données.

Algorithme ou programme :
Suite finie et non ambiguë d'instructions utilisant des données, permettant de résoudre un problème (en temps fini).

Algorithmique :
Ensemble des règles et des techniques qui sont impliquées dans la conception d'algorithmes.

1.2 Exemples d'algorithmes de la vie courante

Recette de cuisine :

Intitulé

Livre de cuisine/Brownies

Le **brownie** est une pâtisserie des États-Unis d'Amérique.

Données

Ingrédients

- 120 g de chocolat noir
- 125 g de beurre mou
- 2 œufs
- 60 g de farine
- 125 g de sucre en poudre
- 50 g de noix de pécan hachées

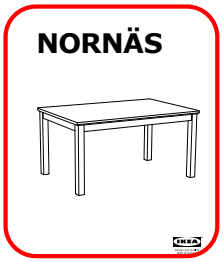
Instructions

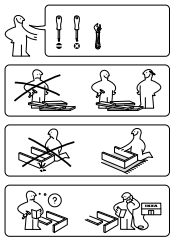
Préparation

- Faire fondre le chocolat au bain-marie
- Fouetter les œufs avec le sucre
- Faire fondre le beurre et le mélanger avec le chocolat fondu.
- Incorporer au mélange œufs-sucre
- Ajouter la farine et les noix de pécan. Mélanger
- Verser la préparation dans un moule recouvert de papier sulfurisé
- Cuire au four dix à quinze minutes à 170°C. A mi-cuisson, couper le brownie en carré

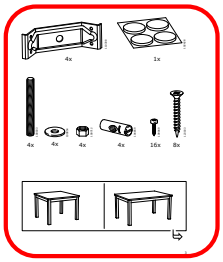
Plan de montage d'un meuble :

Intitulé

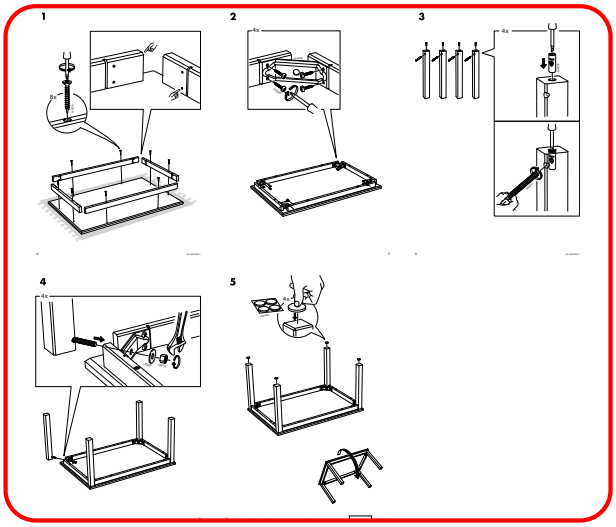




Données



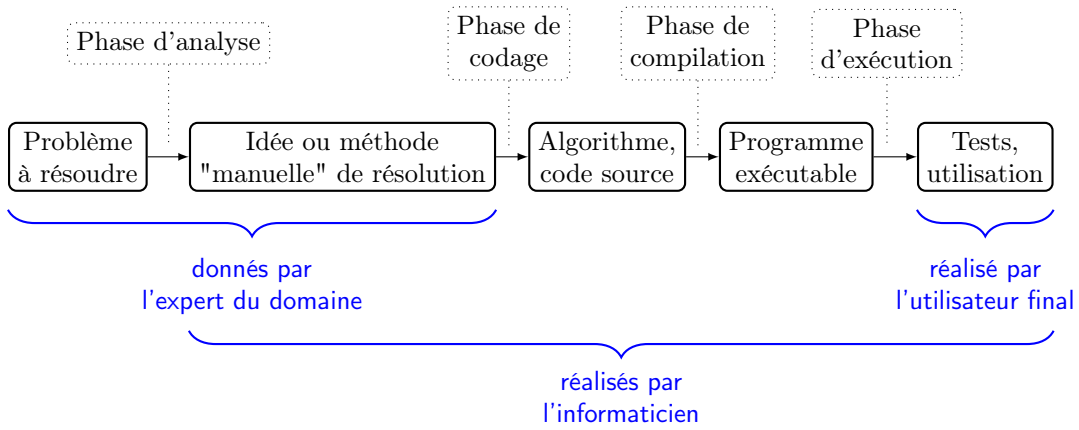
Instructions



1.3 Résolution d'un problème en informatique

1.3.1 Principe

Étapes de la résolution d'un problème informatique



1.3.2 Exemple

Problème :
Obtenir la moyenne de 2 nombres.

Méthode :
- prendre deux nombres a et b
- calculer la moyenne : $m = \frac{a+b}{2}$
- donner comme réponse la valeur de m

Schéma d'algorithme : On s'attachera systématiquement à écrire une première version de l'algorithme en français. Comme le français n'est pas un langage informatique, ce schéma d'algorithme restera approximatif, **mais** on cherchera à le rendre le moins ambigu possible. Ce schéma en français sera placé en commentaire dans votre code informatique : il sert de référence pour rappeler en cours d'écriture quelles sont les étapes de résolution du problème. Il permet quand on reprend votre programme plus tard de comprendre comment il fonctionne et à quoi il sert.

Ici le schéma d'algorithme est placé derrière des // qui signalent au compilateur C++ qu'il doit ignorer ces lignes (elles ne lui sont pas destinées).

```
// saisir 2 entiers (dans 2 variables entières)
// calculer la moyenne (et la ranger dans une variable réelle)
// afficher la moyenne
```

“Squelette” d’un programme en C++ :

```
// description du problème
// auteur, date

// inclure la bibliothèque d'entrées/sorties
#include <iostream>
using namespace std;

// déclarer le début de programme
int main(void)
{

  // INSERER VOTRE ALGORITHME ICI

  // fin du programme
  return 0;
} // main
```

Codage de l’algorithme sous forme de code source en C++ :
On saisit (ou édite) le code source avec un éditeur de texte spécifique (par exemple **geany** ou **emacs**) et on le sauve (on l’enregistre) dans un fichier, par exemple de nom : moyenne2entiers.cc

Vous constatez que le schéma d’algorithme en français a été inclus dans le code source.

```
// moyenne de 2 entiers
// et al., le 09/06/2015

// inclure la bibliothèque d'entrées/sorties
#include <iostream>
using namespace std;

// déclarer le début de programme
int main(void)
{

  // déclaration des variables
  int a, b; // a et b sont des variables de type entier (int)
  float m;  // m est une variable de type réel "à virgule flottante" (float)

  // saisir 2 entiers (dans 2 variables entières)
  cout << "Saisir 2 entiers (séparés par un espace ou par <Entrée>) s.v.p. : ";
  cin >> a >> b;

  // calculer la moyenne (et la ranger dans la variable m)
  m = (a + b) / 2.0;

  // afficher la moyenne (et passer à la ligne)
  cout << "La moyenne est " << m << endl;

  // fin du programme
  return 0;
} // main
```

Compilation du code source en programme exécutable :

On suppose que le code source est dans le fichier moyenne2entiers.cc, on compile dans un **terminal de commande**, c'est à dire qu'on exécute le programme du compilateur C++ (ici il s'appelle g++), qui va transformer le code source en code exécutable, c'est à dire que la machine est capable d'exécuter. Le fichier exécutable résultant sera nommé ici moyenne2entiers :

```
$ g++ -W -Wall -std=c++0x moyenne2entiers.cc -o moyenne2entiers
$
```

Exemple d'exécution :
On teste le programme en l'exécutant, dans le terminal de commande, sur des données exemples :

```
$./moyenne2entiers
Saisir 2 entiers (séparés par un espace ou par <Entrée>) s.v.p. : 12 18
La moyenne est 15
$
```

1.4 Exercices

Exercice 1 : Carré de 2 nombres

Écrivez un programme qui permet de calculer le carré de la somme de deux entiers. Les entiers seront demandés à l'utilisateur et le résultat sera affiché à l'écran.

Exemple d'exécution :

```
Valeur de a et b ? 2 3
Le carré de la somme vaut = 25
```

Exercice 2 : Polynôme

Écrivez un programme qui permet de calculer le polynôme $P(x) = 2x^2 - 3x + 7$. La valeur à calculer doit être saisie et le résultat affiché.

Exemple d'exécution :

```
Valeur de x ? 5
P(5) = 42
```

Exercice 3 : Affichages multiples

Qu'affiche le programme en page suivante ?

```cpp
// Affichages multiples
// et al., le 11/06/2015

#include <iostream>
using namespace std;

int main(void)
{

  int a, b, c; // 3 entiers

  // Partie 1 : initialiser, calculer et afficher
  a = 2;
  b = 3;
  c = 4;
  a = a * 2;
  b = b + c;
  c = a + c;
  cout << a << b << c << endl;

  // Partie 2 : initialiser, calculer et afficher
  a = 2;
  b = 3;
  c = 4;
  b = b + c;
  a = a * 2;
  c = a + c;
  cout << a << b << c << endl;

  // Partie 3 : initialiser, calculer et afficher
  a = 2;
  b = 3;
  c = 4;
  b = b + c;
  c = a + c;
  a = a * 2;
  cout << a << b << c << endl;

  // Partie 4 : initialiser, calculer et afficher
  a = 2;
  b = 3;
  c = 4;
  c = a + c;
  b = b + c;
  a = a * 2;
  cout << a << b << c << endl;

  return 0;
} // main
```

1.5 Fondamentaux de l'algorithmique

1.5.1 Conception :

Comme on peut le voir sur l'exemple précédent l'ordre des instructions est important. Concevoir un algorithme c'est s'interroger simultanément sur :

— quelles opérations doit-on effectuer ?

— dans quel ordre doit-on effectuer ces opérations ?

On verra souvent par la suite que dans un algorithme certaines opérations ne doivent être exécutées que dans certains cas, ou doivent être répétées un certain nombre de fois. C'est le rôle des **structures de contrôle** de gérer cet ordre d'exécution des instructions.

La séquence : une structure de contrôle implicite

De manière implicite, les instructions sont exécutées par l'ordinateur **séquentiellement** dans l'ordre de leur lecture (lecture occidentale : de gauche à droite et de haut en bas). **Toutefois**, on recommande de n'écrire qu'une instruction par ligne pour la lisibilité : le non respect de cette recommandation sera sanctionné.

Exemple à ne pas suivre (illisible car plusieurs instructions par ligne et pas de commentaires) :

```
#include <iostream>
using namespace std;
int main(void)
{
  int a;
  cout << "Saisir 1 entier : "; cin >> a; a = a *2;
  cout << a << endl;
  return 0;
}
```

Exécution possible :

```
$ Saisir 1 entier : 4
8
$
```

1.5.2 Les tests

Un programme ne peut pas être considéré comme correct si l'on n'a pas testé que son déroulement donne bien le résultat attendu. Un seul test ne suffit généralement pas, il faut chercher à mettre en défaut l'algorithme en choisissant des valeurs en entrée qui puissent poser problème. Par exemple on utilisera 0 dans un programme qui utilise la division pour tester si on déclenche une erreur de division par 0.

En cas d'erreur, il faudra généralement tester l'ordre de déroulement des instructions en faisant une **trace d'exécution** du programme.

1.5.3 Traces d'algorithmes

Principe : dérouler/exécuter à la main les instructions de l'algorithme, avec des données exemples.

Formalisme :

On peut représenter la trace d'un algorithme sous la forme d'un tableau à double entrée. En tête de colonne, on met les noms de variables et sur chaque ligne, les valeurs que contiennent les variables (donc l'état de la mémoire de la machine) au fur et à mesure des étapes successives du déroulement de l'algorithme. Ces étapes sont notées en commentaire E_i dans l'exemple ci-dessous et les cellules d'une

ligne de la trace indiquent la valeur des variables une fois l'instruction (où les instructions) de l'étape exécutée.

Exemple :

```cpp
// somme de 2 entiers
// et al., le 09/06/2015

#include <iostream>
using namespace std;

int main(void)
{

  int a, b, somme; // E0

  // saisir 2 entiers
  cout << "Saisir 2 entiers s.v.p. : "; // E1
  cin >> a >> b; // E2

  // calculer la somme
  somme = a + b; // E3

  // afficher la somme
  cout << "La somme est " << somme << endl; // E4

  return 0;
} // main
```

Trace :
Trace pour les données exemples (37, 5) :

Étape	Valeur de **a**	Valeur de **b**	Valeur de **somme**	**Remarque**
E0	?	?	?	initialement les variables sont indéterminées
E1	?	?	?	affichage écran de "Saisir 2 entiers s.v.p. : "
E2	**37**	**5**	?	saisie successive de 37 dans a, puis 5 dans b
E3	37	5	**42**	le calcul est effectué, puis rangé dans somme
E4	37	5	42	affichage écran de "La somme est 42"

2 Les variables

Pour pouvoir traiter des informations, des données, il faut qu'elles soient "entrées" dans la mémoire de la machine. C'est le rôle des variables de contenir les données.

2.1 Principe

Une variable est un emplacement dans la mémoire de l'ordinateur, pour y stocker une donnée (une valeur). Une variable est caractérisée par un nom, un type et une valeur. En C++, une variable doit être déclarée : c'est à dire décrite avec ses caractéristiques **avant** toute utilisation.

2.2 Type et valeur

Le type d'une variable permet de spécifier l'ensemble des valeurs possibles parmi lesquelles on peut choisir son contenu. Le type détermine aussi les instructions que l'on peut exécuter avec le contenu de la variable. Exemple : une variable de type entier peut contenir n'importe quel entier (plus exactement dans la limite de la capacité de la machine), et un entier peut être utilisé avec les opérations comme +, -, * (multiplier) et / (diviser). Particularité du type entier, quand on divise des entiers entre eux, il s'agit de la division entière (ou euclidienne), sans partie décimale : 3 / 2 vaut 1 et pas 1.5 .

Quelques types simples :

Type	nom C++	Valeurs possibles	Instructions possibles
entier	int	42, 0, -3,...	calculs, affichage, ...
réel	float	42.0, 0.5, -3.2,...	calculs, affichage, ...
texte	string	"bonjour", "2",...	affichage, ...
booléen	bool*	true, false	calculs logiques, tests

bool* : on préfèrera souvent remplacer ce type par un entier qui prendra les valeurs 0 (pour false) et 1 (pour true).

Types composés : il existe d'autres types, comme les tableaux et les enregistrements, qui sont composés en agrégeant les types simples et qui seront vus par la suite.

2.3 Conversions

Une valeur de type entier (par exemple, 3) peut être transformée en valeur de type réel (3.0) en préfixant la valeur par le nom du type souhaité entre parenthèses : (float) 3 vaut 3.0 . De même, un réel peut être converti en un entier, mais la partie décimale est alors perdue : (int) 3.5 vaut 3 .

2.4 Opérations élémentaires sur les variables

Déclaration :

Une déclaration de variable est une instruction qui indique au compilateur que le programme va utiliser de la mémoire pour une variable, en indiquant son nom et son type. Par convention les noms de variables seront écrits en minuscules, on les choisira de façon à ce qu'ils aident à comprendre l'algorithme. Attention, après cette étape de l'algorithme, la variable existe mais son contenu (sa valeur) est indéterminé. On déclare une variable en indiquant son type, suivi du nom de la variable (ou des variables). Une variable peut être initialisée (recevoir une valeur initiale) lors de sa déclaration, afin de ne pas rester indéterminée. Exemple :

```
int a; // a est une variable entière, de valeur indéterminée
int b, c; // idem pour b et c
int d = 2; // d est déclarée et prend pour valeur 2
int e = 2, f, g = 3; // e et g ont des valeurs déterminées mais pas f (peu lisible)
```

Affectation :
L'affectation est une instruction qui donne (associe) une valeur à une variable (la valeur est écrite (rangée) dans la mémoire associée à la variable) à l'aide du symbole = qui ne doit pas être confondu avec le test d'égalité (==). Attention, le type de la valeur à affecter doit correspondre au type de la variable.

Utilisation :
Lit (ou récupère) en mémoire la valeur actuellement associée à la variable.

2.5 Exemples

Exemple 1 :

```
// opposé d'un entier
// et al., le 16/06/2015

#include <iostream>
using namespace std;

int main(void)
{

  int nb, nbopp; //déclaration

  // saisie
  cout << "Saisir 1 entier s.v.p. : ";
  cin >> nb; // affectation (masquée) de nb

  // calcul de l'opposé
  nbopp = -nb; // affectation de nbopp, utilisation de nb

  // sortir le résultat (utilisation de nb et nbopp)
  cout << "L'opposé de " << nb << " est " << nbopp << endl;

  return 0;
} // main
```

Exemple 2 (qqs erreurs possibles) :

```
// qqs erreurs possibles sur les variables
// !!! ne compile pas !!! il s'agit d'un contre exemple !!!
// et al., le 16/06/2015
#include <iostream>
using namespace std;

int main(void)
{
  a = 2; // ERRONÉ : a n'est pas déclaré

  int a;

  a = "21"; // ERRONÉ : erreur de type, a est entière, "21" un texte

  a = 21; // ok

  a * 2; //  ERRONÉ : le résultat du calcul est ignoré/abandonné

  cout << a*2 << endl; // ok valeur utilisée (mais peu lisible)

  return 0;
} // main
```

2.6 Exercices

Exercice 4 : Permutation de 2 variables

Écrivez un programme qui saisit 2 entiers à ranger dans 2 variables, puis échange le contenu de ces 2 variables et les affiche (idée : utilisez une 3ème variable pour vous aider à réaliser l'échange). Vérifiez en faisant la trace de votre programme pour les entrées (3, 5).

Exemple d'exécution avec les entrées (37, 42) :

```
Valeur de A ? 37
Valeur de B ? 42
Apres permutation : A = 42, B = 37
```

Exercice 5 : Somme de 4 nombres

1. On veut un programme permettant de saisir 4 entiers nommés a, b, c et d, et d'afficher leur somme. Écrivez deux versions possibles de ce programme :
   — une version utilisant 5 variables (4 pour les saisies, 1 pour le résultat)
   — une version utilisant 2 variables (1 pour les saisies, 1 pour le résultat)
   — Comparez et commentez les avantages et inconvénients de ces deux versions du programme.

   Exemple d'exécution avec les entrées (17, -3, 29, -1) :

   ```
   a ? 17
   b ? -3
   c ? 29
   d ? -1
   a+b+c+d = 42
   ```

2. Faites la trace pour les entrées (1, 2, 3, 4).

Exercice 6 : Calcul des premiers termes d'une suite mathématique

1. On veut un programme permettant de saisir les 2 premiers termes d'une suite mathématique, de calculer les 4 termes suivants de la suite et d'afficher les 6 premiers termes. La suite dont vous calculerez les termes est définie comme suit : ($\forall n \in N$, $u_n = u_{n-1} + u_{n-2}$)
   Écrivez trois versions possibles de ce programme :
   — une version utilisant 6 variables
   — une version utilisant 3 variables
   — une version utilisant 2 variables

   Exemple d'exécution avec les entrées (2, 3) :

```
u0 ? 2
u1 ? 3
Les 6 premiers termes de la suite sont :
2 3 5 8 13 21
```

2. Faites la trace pour les entrées (1, 2).

3 Quelques éléments syntaxiques de base du langage C++

3.1 Le code source

Votre programme doit être tapé dans un fichier, généralement d'extension **.cc** (ou encore .cpp ou .cxx) comme :
monprogramme.cc
La longueur des lignes n'est pas a priori limitée, mais on conseille **fortement** de ne pas faire de lignes de plus de 80 caractères de long (idéalement une ligne doit apparaître entière dans la fenètre de votre éditeur).
Les compilateurs actuels acceptent généralement les caractères accentués du français **pour vos commentaires** (voir ci-dessous). Le langage C++ ne se compose lui que de caractères **non accentués**, comme en anglais. Enfin C++ fait la différence entre majuscules et minuscules : toto n'est pas TOTO, ni même Toto.
Tous les mots-clés du C++ sont en minuscules, ainsi que la plupart des noms d'"objets" (variables, bibliothèques, ...) qui seront utilisés.

Tout code source de programme doit comporter une unique fonction (morceau de code) de nom **main**, dont les instructions sont entre accolades **{ ... }** : voir exemples précédents. L'exécution du code commence par cette fonction. Vous utiliserez, puis créerez d'autres fonctions par la suite, mais pour l'essentiel du semestre 1, votre programme sera composé de cette seule fonction, dont vous changerez le contenu selon l'exercice demandé.

Il est recommandé de créer un fichier par exercice, afin de garder une trace de vos travaux précédents, et de vous habituer à coder ce "squelette" d'un programme C++ (inclusion de bibliothèque, fonction main).

3.2 Le point-virgule

Le point-virgule **;** est **terminateur d'instruction** en C++. Il se trouve donc à la fin de chaque instruction élémentaire.

3.3 Le bloc d'instructions

Partout où on peut placer une instruction, le C++ autorise à placer un **bloc d'instructions** encadré d'accolades **{ }** qui contient une ou plusieurs instructions. Notamment le corps (contenu) de la fonction **main** est toujours constitué d'un bloc d'instruction(s).

3.4 Le commentaire

Partout où on peut mettre un espace dans le code source (entre 2 instructions, avant ou après une accolade, ...), on peut placer un commentaire pour expliquer à quoi sert le code et comment il fonctionne. Le commentaire peut s'écrire en C++ de 2 façons :

- avec // : tout ce qui suit sur la même ligne est un commentaire, ignoré par le compilateur ;
- entre /* et */ : tout le texte (même sur plusieurs lignes) entre /* et */ est ignoré par le compilateur (ce mode de commentaire est déconseillé car source d'erreurs).

Exemple :

```
int a; // déclaration d'une variable entière
int b /* exemple de commentaire dangereux */ ; // attention au ;
```

Bien que le commentaire soit ignoré par la machine, il est indispensable à l'être humain ! On peut considérer que tous les algorithmes écrits avant la naissance des premiers langages machines (dans les années 1940, ou au mieux 1880 pour la machine de Babbage) étaient des commentaires. Dans ce cours le schéma de l'algorithme sera **toujours** exprimé par des commentaires dans le code source (voir exemples de code précédents), sauf exception bien délimitée (exemple d'erreur, simple fragment de code). L'absence de commentaire indiquant le schéma de l'algorithme dans votre code sera sanctionnée comme une erreur.

3.5 Opérations arithmétiques

Nom	Codage	type d'opérandes	type du résultat
addition	$+$	entier ou réel	entier si opérandes entiers, sinon réel
soustraction	$-$	entier ou réel	entier si opérandes entiers, sinon réel
multiplication	$*$	entier ou réel	entier si opérandes entiers, sinon réel
division	$/$	entier ou réel	entier si opérandes entiers, sinon réel
modulo (reste de la division euclidienne)	`%`	2 entiers	1 entier

3.6 Entrées/sorties

On désigne ainsi traditionnellement les opérations de communications de l'unité centrale de l'ordinateur avec les unité périphériques (écran, clavier, mémoires de masse : disque dur, clé usb, ...). Ici on s'intéresse uniquement aux clavier et écran.

Nom	Symbole	commentaire
saisie	`cin >>` variable	affecte la valeur saisie au clavier dans la variable
affichage	`cout <<` texte ou variable	affiche le texte ou la valeur de la variable
affichage d'un saut de ligne	`cout << endl`	

4 Structure de contrôle : alternative ou "SI"

4.1 Principe

L'alternative est une structure de contrôle qui permet de différencier les traitements (le travail) à exécuter selon la valeur d'une condition (d'un test). Le schéma général est l'un des deux suivants, selon les besoins :

si-alors-sinon

```
// si une condition est vraie
//      bloc d'instructions à exécuter dans ce cas
// sinon
//      bloc d'instructions à exécuter en cas de condition fausse
// instructions suivantes dans tous les cas
```

si-alors

```
// si une condition est vraie
//      bloc d'instructions à exécuter dans ce cas
// instructions suivantes dans tous les cas
```

En C++, l'alternative se code par les mots-clés **if** (si) et **else** (sinon), et la condition doit obligatoirement être entre parenthèses. Le bloc d'instructions est soit un bloc C++ mis entre accolades, soit (déconseillé) une unique instruction qui n'a pas besoin d'accolades.

Exemple :

```cpp
// mineurmajeur
// et al., le 16/06/2015

#include <iostream>
using namespace std;

int main(void)
{

  int age;

  // saisie
  cout << "Saisir votre age s.v.p. : ";
  cin >> age;

  // test et affichage majeur ou pas
  // si agee de plus de est plus grand que 18, alors la personne est majeure
  // sinon la personne est mineure
  if (age >= 18)
  {
        cout << "Vous etes majeur" << endl;
  }
  else
  {
        cout << "Vous etes mineur" << endl;
  }

  // message de fin
  cout << "Bonjour chez vous." << endl;

  return 0;
} // main
```

4.2 Conditions

La condition d'une alternative est une expression conditionnelle, c'est-à-dire un calcul dont le résultat est un booléen (vrai ou faux). À l'exécution du programme, l'ordinateur calcule la valeur de la condition quand il arrive à cette étape du traitement. Si cette valeur vaut vrai (`true`) alors il exécute le bloc de traitement indiqué, sinon (valeur `false`) il exécute le second bloc de traitement (s'il existe). Puis on reprend séquentiellement la suite des instructions situées **après** la structure d'alternative.

Les expressions conditionnelles s'écrivent avec les opérateurs C++ suivants :

Opérateurs de comparaison

Nom	Symbole	S'applique sur	Produit comme résultat
égalité	==	2 valeurs de même type	1 booléen
différence	!=	2 valeurs de même type	1 booléen
inférieur strict	<	2 entiers ou 2 réels	1 booléen
inférieur ou égal	<=	2 entiers ou 2 réels	1 booléen
supérieur strict	>	2 entiers ou 2 réels	1 booléen
supérieur ou égal	>=	2 entiers ou 2 réels	1 booléen

Il est souvent utile de combiner ensemble plusieurs comparaisons, par exemple si on veut tester qu'une variable entière a est comprise entre 1 et 10 inclus, la condition pourrait être : (a >= 1 et a <= 10). On utilise alors les opérateurs logiques suivants :

Opérateurs logiques

Nom	Symbole	S'applique sur	Produit comme résultat
ET (conjonction)	`&&`	2 booléens	1 booléen
OU (disjonction)	`||`	2 booléens	1 booléen
NON (négation)	`!`	1 booléen	1 booléen

4.3 Exemples

Valeur absolue d'un nombre : Comment afficher la valeur absolue d'un nombre ? (sans faire appel à la fonction prédéfinie `abs(n)`).

Méthode :

```
// valeur absolue
// et al., le 16/06/2015

#include <iostream>
using namespace std;

int main(void)
{

  int n;

  // saisir le nombre
  cout << "Saisir le nombre : ";
  cin >> n;

  // si le nombre est positif, alors on l'affiche
  // sinon on affiche son opposé
  if (n >= 0)
  {
        cout << "La valeur absolue de " << n << " est " << n << endl;
  }
  else
  {
         cout << "La valeur absolue de " << n << " est " << -n << endl;
  }

  return 0;
} // main
```

Équation du 1er degré : Écrivez un programme (sans oublier le schéma de l'algorithme) permettant de résoudre une équation $ax + b = 0$, avec a et b 2 réels à saisir, et x un réel inconnu.

Méthode :

```
// saisir a et b
// si a != 0
//    il y a une solution de valeur x = -b/a
// si a == 0 et b != 0
//    il n'y a pas de solution
// si a == 0 et b == 0
//    infinité de solutions (tout réel x est solution)
```

Faites les traces de l'algorithme lorsque l'on saisit $(-0.5, 21)$, $(0, 37)$ et $(0, 0)$ comme valeurs pour (a, b).

4.4 Exercices

Exercice 7 : Comparaison de deux entiers

1. Écrivez un programme (sans oublier le schéma de l'algorithme) qui saisit deux entiers et affiche le plus grand.

   Exemple d'exécution avec les entrées (2, 7) :

   ```
   Entrez le premier entier : 2
   Entrez le second entier : 7
   Le plus grand est 7.
   ```

2. Réécrivez votre programme pour prendre en compte l'égalité.

   Exemple d'exécution avec les entrées (12, 12) :

   ```
   Entrez le premier entier : 12
   Entrez le second entier : 12
   Les deux entiers sont egaux (12).
   ```

3. Faites la trace du dernier programme avec les entrées (42, 37), (37, 42) et (1, 1).

Exercice 8 : Produit positif, négatif ou nul

1. Écrivez un programme (sans oublier le schéma de l'algorithme) qui saisit deux réels et affiche si leur produit est positif, négatif ou nul.

   Exemple d'exécution :

   ```
   Entrez le premier nombre : -2.0
   Entrez le second nombre : 13.37
   Le produit est negatif.
   ```

2. Faites la trace avec l'entrée (-2.0, 13.37).
3. Écrivez maintenant un programme (sans oublier le schéma de l'algorithme) qui saisit deux réels et affiche si leur produit est positif, négatif ou nul, sans calculer le produit des 2 nombres.

Exercice 9 : Pair ou impair

Écrivez un programme (sans oublier le schéma de l'algorithme) qui saisit un entier et affiche s'il est pair ou impair.

Exemple d'exécution :

```
Entrez un entier : 13
13 est impair
```

Exercice 10 : Année bissextile

Écrivez un programme (sans oublier le schéma de l'algorithme) qui permet de dire si une année est bissextile ou non. Votre algorithme doit être le plus concis possible.

Rappel : Une année est bissextile si elle est multiple de 4 sans être multiple de 100 ou si elle est multiple de 400.

Exemple d'exécution :

```
Annee ? 1900
1900 n'est pas bissextile
```

5 Algèbre de Boole

5.1 Motivation

Lorsqu'on utilise des conditions (par exemple, avec la structure d'alternative), on utilise le type booléen et donc l'algèbre de Boole. Celle-ci est intéressante pour formaliser des conditions, les simplifier...

5.2 Principe

Définitions :
L'algèbre de Boole est composée de l'ensemble à deux éléments {`Vrai`,`Faux`} et de quelques opérateurs (`ET`, `OU`, `NON` ...) définis sur cet ensemble par la table de vérité suivante.

a	b	a `ET` b	a `OU` b	`NON` a
`Vrai`	`Vrai`	`Vrai`	`Vrai`	`Faux`
`Vrai`	`Faux`	`Faux`	`Vrai`	`Faux`
`Faux`	`Vrai`	`Faux`	`Vrai`	`Vrai`
`Faux`	`Faux`	`Faux`	`Faux`	`Vrai`

Priorités des opérateurs :
- `NON` est prioritaire sur `ET`
- `ET` est prioritaire sur `OU`
- ne pas hésiter à mettre des parenthèses en cas de doute

Élement neutre, absorbant, complémentarité :
- `Vrai ET` $a = a$
- `Vrai OU` $a =$ `Vrai`
- `Faux ET` $a =$ `Faux`
- `Faux OU` $a = a$
- `NON`$(a$ `OU` $b) =$ `NON` a `ET NON` b
- `NON`$(a$ `ET` $b) =$ `NON` a `OU NON` b

Commutativité-Distributivité :
- $(a$ `OU` $b) = (b$ `OU` $a)$
- $(a$ `ET` $b) = (b$ `ET` $a)$
- $(a$ `ET` $(b$ `OU` $c)) = (a$ `ET` $b)$ `OU` $(a$ `ET` $c)$
- $(a$ `OU` $(b$ `ET` $c)) = (a$ `OU` $b)$ `ET` $(a$ `OU` $c)$

Autres notations :
- le `OU` est aussi noté $+$
- le `ET` est aussi noté . ou $*$
- le `NON` est aussi noté $\neg$

5.3 Exemples

Résultats d'expressions logiques

	a	b	a `ET` b	a `OU` b
x	$x < 12$	$x > -4$	$(x < 12)$ `ET` $(x > -4)$	$(x < 12)$ `OU` $(x > -4)$
0				
-5				
15				
12				
-4				

NON $(x < 12) =$

NON $(x > -4) =$

	a	b	NON a OU NON b	NON a ET NON b
x	$x < 12$	$x > -4$	$(x \geq 12)$ OU $(x \leq -4)$	$(x \geq 12)$ ET $(x \leq -4)$
0				
-5				
15				
12				
-4				

Autre manière de calculer cette dernière table ?

5.4 Exercices

Exercice 11 : Applications : calcul d'expressions logiques

On pose : $a = 2, b = 3, c = 5, d = 10, X = \text{Vrai}, Y = \text{Faux}$.
Calculez les expressions suivantes (en développant vos calculs).

1. $(a < b)$ ET $(a < c)$
2. NON $((a < b)$ ET $(a < c))$
3. NON $(a < b)$ ET $(a < c)$
4. $(a < c)$ ET $(c == d/2)$
5. $(d/a == c) == Y$
6. $(d/c == b) == Y$
7. $(d/c == b) == X$
8. $(a < b)$ ET $(d < c)$
9. $(a < b)$ ET $(d < c) == Y$

Exercice 12 : Démonstration de la distributivité

Montrez que les expressions booléennes suivantes sont vérifiées :

1. (a + b).(a + c) = a + b.c
2. (a.b) + (a.c) = a.(b + c)

Exercice 13 : Simplification d'expressions

Simplifiez les expressions booléennes suivantes :

1. (a.b +c).(b.c + a) + ¬a.(b + c)
2. (a + b).(c + ¬a) + b.(¬c + ¬b)
3. (a + b.c) + (c + ¬b.c + a.b).(b + ¬c).b + (a + ¬a.¬b)

Exercice 14 : Quel champignon ?

Qu'affiche le programme suivant (1 :VRAI, 0 :FAUX) lorsque les entrées sont (1), puis (0,1), puis (0,0) ?

```cpp
// Affichages guidés
// et al., le 11/06/2015

#include <iostream>
using namespace std;

int main(void)
{
  int rep;

  // saisie forme 1
  cout << "A-t-il une forme de buisson ? <1/0> ";
  cin >> rep;

  if (rep) //forme de buisson
  {
          cout << "c'est un clavaire" << endl;
  }
  else
  {
          // saisie forme 2
          cout << "A-t-il une forme de toupie ? <1/0> ";
          cin >> rep;
          if (rep) //forme de toupie
          {
                  cout << "c'est un vesse" << endl;
          }
          else
          {
                  cout << "je ne sais pas" << endl;
          }
  }

  return 0;
} // main
```

Exercice 15 : Température

Pour le programme suivant :

1. indiquez les affichages produits lorsqu'on saisit $temp = -15$, $temp = 3$, $temp = 6$ et $temp = 12$;
2. indiquez, pour chaque action d'affichage, la condition complète pour que celle-ci soit réalisée.

```cpp
// Affichages temperatures
// et al., le 11/06/2015

#include <iostream>
using namespace std;

int main(void)
{
        float temp;

        // saisie temperature
        cout << "Quelle température fait-il ? ";
        cin >> temp;

        // Affichages 1
        cout << "Affichage 1" << endl;
        if (temp < 10 && temp > -10) //froid
        {
                cout << "froid" << endl;
        }
        if (temp < 5) //hiver
        {
                cout << "hiver" << endl;
        }
        if (temp > 0) //degel
        {
                cout << "degel" << endl;
        }

        // Affichages 2
        cout << "Affichage 2" << endl;
        if (temp < 10 && temp > -10) //froid
        {
                cout << "froid" << endl;
        }
        if (temp < 5) //hiver
        {
                cout << "hiver" << endl;
        }
        else
        {
                if (temp > 0) //degel
                {
                        cout << "degel" << endl;
                }
        }

        // Affichages 3 - a la suite
```

```
  // Affichages 3 - suite
        cout << "Affichage 3" << endl;
        if (temp < 10 && temp > -10) //froid
        {
                cout << "froid" << endl;
        }
        else
        {
                if (temp < 5) //hiver
                {
                        cout << "hiver" << endl;
                }
                else
                {
                        if (temp > 0) //degel
                        {
                                cout << "degel" << endl;
                        }
                }
        }

  return 0;
} // main
```

Exercice 16 : État de l'eau

1. Écrivez un programme (sans oublier le schéma de l'algorithme) permettant de saisir la température de l'eau et d'afficher son état :
   — "glace" si la température est inférieure à 0,
   — "eau" si la température est supérieure stricte à 0 et inférieure à 100,
   — "vapeur" si la température est supérieure stricte à 100.

   Exemple d'exécution avec l'entrée (42) :

   ```
   Temperature ? 42
   eau
   ```

2. Faites la trace pour les entrées (0) et (100).

Exercice 17 : Manipulation de date

1. Écrivez un programme (sans oublier le schéma de l'algorithme) (concis) qui permet de dire si une date (représentée par trois entiers : jour, mois, annee) est valide.
   Indications :
   — listez tout ce qui rend une date invalide ;
   — pour faciliter le programme trouvez une formule permettant de convertir le mois en son nombre de jours, sinon faites sans ;
   — vous donnerez, pour chaque "branche" conditionnelle (if ou else), l'expression logique qui lui correspond, expression que vous simplifierez s'il y a lieu.

   Exemple d'exécution :

```
Jour ? 31
Mois ? 1
Annee ? 2042
Date valide !
```

6 Structure de contrôle : répétition "tant que"

6.1 Principe

La structure de contrôle "tant que" permet de répéter un traitement tant qu'une condition est vérifiée. On parle aussi de structure de **boucle** "tant que". Le schéma général est le suivant :

tant-que

```
// tant que une condition est vraie
//      bloc d'instructions à exécuter
// instructions suivantes une fois la boucle terminée
```

Si la condition est fausse lorsqu'elle est évaluée, alors le bloc d'instructions n'est pas exécuté et on passe à la suite. Si la condition est vraie, on exécute le bloc, **puis on ré-évalue la condition** pour savoir s'il faut répéter son exécution. Tant que la condition est vraie, l'exécution du bloc est répétée et la condition ré-évaluée. Typiquement le contenu du bloc doit, à un moment donné, changer la valeur de la condition afin qu'on puisse quitter la boucle et passer à la suite, sinon on boucle à l'infini. Au sortir de la boucle, on est sûr que la condition est fausse.

En C++, cette boucle se code par le mot-clé **while** (tant que), et la condition doit obligatoirement être entre parenthèses. Le bloc d'instructions est soit un bloc C++ mis entre accolades, soit (déconseillé) une unique instruction qui n'a pas besoin d'accolades.

Exemple :

```
// saisir un entier strictement positif
// et al., le 16/06/2015

#include <iostream>
using namespace std;

int main(void)
{

  int n;

  // saisie initiale
  cout << "Saisir un entier strictement positif s.v.p. : ";
  cin >> n;

  // recommencer la saisie tant que l'entier n'est pas correct
  // entier incorrect <=> entier négatif ou nul
  while (n <= 0)
  {
    cout << "le nombre n'est pas positif, redonnez-le s.v.p. : ";
    cin >> n;
  }
  // on est sorti de la boucle <=> entier correct <=> entier strictement positif
  // message de fin
  cout << n << " > 0" << endl;

  return 0;
} // main
```

Faites la trace pour les saisies suivantes : 0, -1, 3

6.2 Exercices

Exercice 18 : Inverse d'un nombre

Écrivez un programme (sans oublier le schéma de l'algorithme) qui saisit un réel et affiche son inverse. Attention comme on ne peut pas diviser par 0, la saisie doit être recommencée tant qu'elle n'est pas valide. Faites une trace pour 2 saisies permettant de vérifier que votre programme rend un résutat correct.

Exercice 19 : Division et diviseur non nul

1. Écrivez un programme (sans oublier le schéma de l'algorithme) qui saisit deux entiers et affiche leur division (euclidienne).

   Exemple d'exécution :

   ```
   Dividende ? 42
   Diviseur ? 2
   42 / 2 = 21
   ```

2. Que se passe-t-il avec votre programme si vous entrez la valeur 0 pour le diviseur ?
3. Réécrivez votre programme pour qu'il force l'utilisateur à saisir un diviseur valide.

   Exemple d'exécution :

   ```
   Dividende ? 42
   Diviseur ? 0
   Erreur !
   Diviseur ? 10
   42 / 10 = 4
   ```

4. Faites la trace de votre programme pour l'entrée (7, 0, 0, 2).

Exercice 20 : sommes

1. Écrivez un programme (sans oublier le schéma de l'algorithme) qui saisit des entiers puis affiche leur somme. La saisie termine quand on entre la valeur 0.

   Exemple d'exécution :

   ```
   Valeur ? 5
   Valeur ? 37
   Valeur ? 0
   Somme = 42
   ```

2. Faites la trace pour l'entrée (12, 13, 1, 0).
3. Écrivez un programme qui saisit des entiers puis affiche leur somme. La saisie termine quand la somme est supérieure à 40.

   Exemple d'exécution :

   ```
   Valeur ? 5
   Valeur ? 37
   Somme = 42
   ```

4. Faites la trace pour l'entrée (12, 25, 8).

Exercice 21 : Programme d'accueil

1. Écrivez un programme (sans oublier le schéma de l'algorithme) qui permet de saisir le nom d'un utilisateur (exemple : "toto") et de lui afficher un message d'accueil (exemple : "Bonjour toto"). Ce programme permettra de recommencer la saisie et l'affichage du message jusqu'à ce que le nom de l'utilisateur soit un texte vide. Pour saisir un texte, vous utiliserez la fonction **getline(cin, s)** qui permet de lire la ligne à partir du canal d'entrée et de la sauvegarder dans "s" (de type string).

   Exemple d'exécution pour les entrées ("Jake", "Elwood", "") :

   ```
   Nom ? Jake
   Bonjour Jake !
   Nom ? Elwood
   Bonjour Elwood !
   Nom ?
   Au revoir !
   ```

2. Faites la trace pour l'entrée (Elsa, Antoine, Gérard).

7 Structure de contrôle : répétition "pour"

7.1 Principe

La structure de contrôle "pour" permet de répéter un traitement un certain nombre de fois. Le nombre de répétitions est connu au moment où on commence la boucle "pour", il est controlé par une variable compteur qui va compter le nombre de répétitions. Le schéma général est le suivant :

pour

```
// pour compteur de i à j
//      bloc d'instructions à exécuter
// instructions suivantes une fois la boucle terminée
```

À l'arrivée sur la boucle, la variable compteur est initialisée à la première valeur i. L'algorithme teste alors si le compteur a **dépassé** la valeur finale j : si oui le bloc d'instructions n'est pas exécuté, on "sort de la boucle" et on passe aux instructions suivantes ; si par contre la valeur finale du compteur n'est pas dépassée, on exécute le bloc d'instructions, on incrémente le compteur (il augmente de 1) et on ré-évalue le test de dépassement de la valeur finale pour savoir s'il faut répéter le traitement.
Comme le nombre de répétitions est connu, si on veut faire N répétitions j devra être égal à i+N-1.

Le mot-clé C++ est **for** (pour). Il s'utilise sur le modèle suivant :

```
for ( initialisation_compteur ; condition_dépassement ; incrémentation_compteur )
```

Exemple :

```
// afficher les 5 premiers entiers naturels pairs : 0, 2, 4, 6, 8
// et al., le 16/06/2015

#include <iostream>
using namespace std;

int main(void)
{

  int nat, cpt; // cpt sera le compteur de boucle pour

  // initialisation
  nat = 0; // 1er naturel pair

  // repeter 5 fois (5 entiers a afficher) :
  // afficher le pair courant et préparer le suivant
  for (cpt = 1; cpt <= 5; cpt++)
  {
    cout << nat << endl; // afficher pair courant
    nat = nat + 2; // préparer pair suivant
  }

  return 0;
} // main
```

À noter que la variable compteur étant une variable ordinaire, il est possible de la modifier dans le bloc d'instructions à répéter, mais on s'interdira de le faire pour préserver la lisibilité de l'algorithme : une boucle "pour" est une boucle basée sur une énumération simple dans un compteur, toute boucle plus compliquée sera implantée avec un "tant que".

7.2 Équivalence boucle "pour" / boucle "tant que"

Pour des raisons historiques, en C++ la boucle "pour" est en fait un "tant que" masqué, qui permet de s'écarter largement du schéma présenté ci-dessus. On se limitera cependant à ce schéma simple, qui nous suffira pour toutes les applications de ce cours.

L'exemple précédent pourra donc être ré-écrit comme suit :

Exemple :

```
// afficher les 5 premiers entiers naturels pairs
// et al., le 16/06/2015

#include <iostream>
using namespace std;

int main(void)
{

  int nat, cpt; // cpt sera le compteur de boucle pour

  // initialisation
  nat = 0; // 1er naturel pair

  // repeter 5 fois : afficher le pair courant et préparer le suivant
  cpt = 1; // initialisation du compteur
  while (cpt <= 5) // test si dépassement
  {
    cout << nat << endl; // afficher pair courant
    nat = nat + 2; // préparer pair suivant
    cpt++; // incrémentation du compteur
  }

  return 0;
} // main
```

7.3 Exemple : étoiles

Afficher sur une même ligne autant d'étoiles "*" que demandées par l'utilisateur.

Méthode :

```
// saisir un entier n
// repeter n fois, afficher une étoile
// passer à la ligne
```

7.4 Exercices

Exercice 22 : moyenne

Écrivez un programme (sans oublier le schéma de l'algorithme) qui saisit 4 entiers puis affiche leur moyenne. Pour cela, utilisez une boucle "pour". Faites une trace.

Exercice 23 : le plus grand entier saisi

1. Écrivez un programme qui saisit 10 entiers puis affiche le plus grand.

   Exemple d'exécution :

   ```
   Entier 1 ? 10
   Entier 2 ? 42
   ...
   Entier 10 ? 37
   Le plus grand entier saisi est 42.
   ```

2. Réécrivez le programme pour qu'il saisisse d'abord le nombre d'entiers à traiter.

   Exemple d'exécution :

   ```
   Nombre d'entiers ? 3
   Entier 1 ? 10
   Entier 2 ? 42
   Entier 3 ? 37
   Le plus grand entier saisi est 42.
   ```

3. Faites la trace pour l'entrée (4, 5, 4, 9, 3).

Exercice 24 : Somme de a à b

1. Écrivez un programme qui saisit deux entiers a et b puis affiche la somme des entiers entre a et b. Pour cela, utilisez une boucle "pour".

   Exemple d'exécution :

   ```
   a ? 10
   b ? 30
   La somme des entiers de 10 à 30 est 420.
   ```

2. Faites la trace pour l'entrée (3, 6).

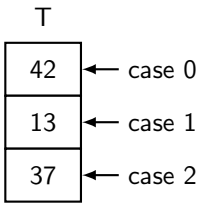
Exercice 25 : Affichage

1. Affichez une ligne d'étoiles (caractère "*") ;
2. affichez une colonne d'étoiles ;
3. affichez un carré de N étoiles, la valeur de N sera demandée à l'utilisateur ;
4. affichez un triangle isocèle de N étoiles de côté.

8 Tableaux

8.1 Notion de tableau

Structure de données composée de données contiguës de même type. Par exemple : T est un tableau de 3 entiers $\iff$ T a 3 éléments qui seront obligatoirement des entiers et chaque élément est mis dans une case numérotée.



8.2 Déclarer un tableau

Pour décrire le type des tableaux on utilisera le type prédéfini "array" de la bibliothèque nommée "array" et qui sera à inclure au même endroit que les autres librairies (cf ci-dessous).

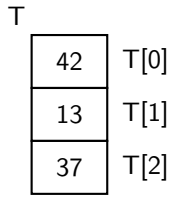
définition d'un tableau Tab à NBELEM éléments de type "type des éléments"

```
const int NBELEM = nombre d'éléments;
array <type des éléments, NBELEM> Tab;
```

Attention, comme pour les variables classiques, on ne connait pas les valeurs initiales d'un tableau.

8.3 Accéder à un élément d'un tableau

On accède à un élément d'un tableau en spécifiant le nom du tableau suivi de l'indice (entre crochets) de la case dans lequel est l'élément. Les indices commencent à 0.



Chaque case du tableau se manipule comme une variable classique, du type correspondant.

Exemple :

```
//  mettre la valeur 0 dans la 2\ieme~case d'un tableau de 3 entiers
// et al., le 16/06/2015

#include <iostream>
#include <array>
using namespace std;

const int NBELEM = 3;

int main(void)
{
  // declaration du tableau T de 3 entiers
  array <int, NBELEM> T;

  // valeur 0 pour le 2e element du tableau
  T[1] = 0;

  // afficher la valeur de ce 2e element du tableau
  cout << "T[1] = " << T[1] << endl;

  return 0;
} // main
```

8.4 Parcourir un tableau

Parcourir un tableau de taille n revient à accéder à chaque case, qu'on appellera la **case courante**, et donc à accéder à la case d'indice 0, puis celle d'indice 1, etc jusqu'à la case d'indice $n-1$. Ceci s'écrit très simplement avec une boucle "pour".

Exemple : initialiser à 0 tous les éléments (réels) du tableau T

```
//  intialiser le tableau T composé de 3 réels à 0
// et al., le 16/06/2015

#include <iostream>
#include <array>
using namespace std;

const int NBELEM = 3;

int main(void)
{
  // declaration du tableau tab de 3 reels
  array <float, NBELEM> T;
  int i; //declaration de l'indice de boucle

  // parcourir le tableau <=> pour tous les elements du tableau
  //    initialiser l'element de la case courante a 0
  for (i=0;i<=2;i++)
  {
        T[i] = 0;
  }

  // affichage de toutes les valeurs du tableau
  // parcourir le tableau
  //    afficher l'element de la case courante
  cout << "Affichage :" << endl;
  for (i=0;i<=2;i++)
  {
        cout << "T[" << i << "] = " << T[i] << endl;
  }

  return 0;
} // main
```

Parcourir le tableau en mettant une valeur à chaque élément correspond à **remplir le tableau**.

8.5 Exemple : sommer des notes

Problème :
On veut saisir 5 notes, calculer leur somme et afficher les 5 notes séparées par des "+" avec le résultat de la somme ensuite.

Exemple d'exécution :

```
Note 0 ? 10.3
Note 1 ? 12.5
Note 2 ? 2
Note 3 ? 5.5
Note 4 ? 4.2
10.3 + 12.5 + 2 + 5.5 + 4.2 = 34.5
```

Méthode :

```
// saisie des notes
// repeter 5 fois
//     saisir une note, la mettre dans la 1ere case libre (non remplie) du tableau

// calcul de la somme
// initialiser la somme a 0
// parcourir le tableau
//     ajouter l'element courant a la somme

// affichage
// parcourir le tableau
//     afficher l'element courant, suivi d'1 '+' sauf le dernier qui sera suivi d'1 '='
// afficher la somme
```

Algorithme :

```
//  calcul de la somme des éléments d'un tableau "notes" composé de 5 réels
// et al., le 16/06/2015

#include <iostream>
#include <array>
using namespace std;

const int NBELEM = 5;

int main(void)
{
        float somme; //declaration de la somme
        array <float, NBELEM> notes; //tableau de 5 reels
        int i; //compteur de boucle


        // saisie des notes
        // repeter 5 fois
        //     saisir une note, la mettre dans la 1ere case libre du tableau
        for (i=0;i<=4;i++) //E0
        {
                cout << "Note " << i << " ? ";
                cin >> notes[i]; //E1
        }

        // calcul de la somme
        // initialiser la somme a 0
        somme = 0; //E2
        // parcourir le tableau
        //     ajouter l'element courant a la somme
        for (i=0;i<=4;i++) //E3
        {
                somme = somme + notes[i]; //E4
        }

        // affichage
        // parcourir le tableau
        //     afficher l'element courant
        for (i=0;i<=4;i++) //E5
        {
          cout << notes[i] << " "; //E6
          if (i!=4) //E7
          {
            cout << "+ "; //E8
          }
          else
          {
            cout << "= "; //E9
          }
        }
        // afficher la somme
        cout << somme << endl; //E10

  return 0;
} // main
```

Autre solution d'affichage :

```
        // affichage
        // parcourir le tableau jusqu'a l'avant dernier element
        //     afficher l'element courant suivi du signe "+"
        for (i=0;i<=3;i++) //E5
        {
          cout << notes[i] << " + "; //E6
         } //remarque : lorsque la boucle se termine, i vaut 4
        // afficher le dernier element du tableau, suivi du signe "=", puis la somme
        cout << notes[i] << " = "<< somme << endl; //E9
```

Option de compilation

Rajouter dans la ligne de compilation l'option -std=c++11.

8.6 Exercices

Exercice 26 : Nombre de notes supérieures à la note moyenne

Écrivez un programme (sans oublier le schéma de l'algorithme) qui saisit un tableau de 5 notes et affiche les notes supérieures strictes à la note moyenne.

Faites la trace pour l'entrée (10.0, 11.0, 8.0, 12.0, 9.0).

Exercice 27 : Écart type

Écrivez un programme (sans oublier le schéma de l'algorithme) qui saisit 5 notes et affiche leur écart-type.

Calculez mentalement l'écart-type pour les entrées (10.0, 11.0, 8.0, 12.0, 9.0) : vérifiez l'exactitude (la "correction") de votre programme.

Rappel :

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2} \text{ où } \bar{x} \text{ est la moyenne des } x_i$$

Exercice 28 : Permutation circulaire

Écrivez un programme (sans oublier le schéma de l'algorithme) qui :

- remplit un tableau Tinit composé des 50 premiers naturels (on commence à 0) ;
- modifie le tableau Tinit en faisant une permutation circulaire des élements à gauche, c'est-à-dire que le naturel 1 se retrouve à la place de 0, que 2 se retrouve à la place de 1, ... que 49 se retrouve à la place de 48 et que 0 se retrouve à la place de 49 ;
- recopie le tableau Tinit ainsi modifié dans le tableau Tdest en remettant les éléments dans l'ordre croissant de façon à ce que Tdest soit le même que Tinit au départ.

Exercice 29 : Manipulations de tableau

Écrivez un programme (sans oublier le schéma de l'algorithme) qui :

1. permet de remplir aléatoirement un tableau de 50 entiers ;
2. affiche les éléments du tableau 10 par ligne ;
3. recherche puis affiche le nombre le plus petit du tableau ;
4. trie le tableau suivant le "tri minimum" qui consiste à trouver le plus petit élément du tableau, à le mettre dans la première case du tableau (on fera attention de ne pas écraser des valeurs du tableau), puis à recommencer la recherche du plus petit sur le tableau privé de son premier élément (puisqu'il est bien placé) et de le mettre à sa place soit dans la 2e case du tableau, puis à recommencer la recherche du plus petit sur le tableau privé de ses premier et deuxième éléments (puisqu'ils sont bien placés) et de le mettre à sa place soit dans la 3e case du tableau, ainsi de suite jusqu'à arriver au dernier élément du tableau.

A partir de la même hypothèse que précédemment (tableau rempli aléatoirement), écrivez un programme qui recherche le 2e plus petit élément du tableau (plus grand que le minimum).