

Chapitre 4.1

STRUCTURES DE PROGRAMMATION

POLYTECHNIQUE
MONTRÉAL

LE GÉNIE
EN PREMIÈRE CLASSE



Instruction de décision

```
if (expression_booléenne)  
    instruction;
```

```
int a, b;  
cout << " Entrer deux entiers: ";  
cin >> a >> b;  
if (a == b)  
    cout << a << " est égal à " << b;
```

Instruction de décision

if (expression_booléenne)

instruction_if;

else

instruction_else;

```
int a, b;
```

```
cout << " Entrer deux entiers: ";
```

```
cin >> a >> b;
```

```
if (a == b)
```

```
    cout << a << " est égal à " << b;
```

```
else
```

```
    if (a < b)
```

```
        cout << a << " est plus petit que " << b;
```

```
    else
```

```
        cout << a << " est plus grand que " << b;
```

IV_if-else.cpp

Instruction de décision

```
int note;

cout << "Donner une note sur 100: ";
cin >> note;

if (note >= 90)
    cout << "A" << endl;
else if( note >= 70)
    cout << "B" << endl;
else if (note >= 60)
    cout << "C" << endl;
else if (note >= 50)
    cout << "D" << endl;
else
    cout << "F" << endl;
```

IV_elseif.cpp

Instruction de décision

```
switch (expression) {  
    case constante_1 : instruction_1;  
                        break;  
  
    case constante_2 :  
    case constante_3 : instruction_2_3;  
                        break;  
  
    ...  
  
    case constante_x : instruction_x;  
                        break;  
  
    ...  
  
    default           : instruction;  
}
```

Instruction de décision

```
int  points = 0;
char note;
cout << "Note = "; cin >> note;
switch (note) {
    case 'A' : points +=4;
    case 'B' : points +=3;
    case 'C' : points +=2;
                break;
    case 'D' : points = 1;
                break;
    default  : points = 0;
}
cout << " Cette note vaut " << points;
```

IV_switch.cpp

Imbrication d'instructions de décision

```
int main()
{
    const string BISSEXTILE      = " est une année bissextile. ";
    const string PAS_BISSEXTILE = " n'est pas une année bissextile. ";
    cout << "Inscrire l'année dont vous désirez" << endl;
    cout << "connaître la nature (bissextile ou non) => ";
    int annee;
    cin >> annee;
    string statut;
    if (annee % 4 != 0)           // Année non un multiple de 4.
        statut = PAS_BISSEXTILE;
    else                         // Année multiple de 4 mais
        if (annee % 100 != 0)    // pas un multiple de 100.
            statut = BISSEXTILE;
        else                   // Année multiple de 4 et de 100
            if (annee % 400 != 0) // mais pas de 400.
                statut = PAS_BISSEXTILE;
            else                // Multiple de 4, 100 et 400.
                statut = BISSEXTILE;
    cout << endl << annee << statut;
}
```

IV_bissextile.cpp

Algorithme

Structures décisionnelles en pseudo-code

SI Expression Booléenne ALORS

Opération 1

Opération 2

SI Expression Booléenne ALORS

Opération(s) dans le cas Expression VRAIE

SINON

Opération(s) dans le cas Expression FAUSSE

Algorithme

Structures décisionnelles en pseudo-code

SI Expr Booléenne1 ALORS

Opération(s) dans le cas Expr.1 VRAIE

SINON SI Expr_Booléenne2 ALORS

Opération(s) dans le cas Expr1 FAUSSE et Expr2 VRAIE

Algorithme

Structures décisionnelles en pseudo-code

SELON la valeur de l'expression

= Constante1

Opération(s) dans le cas expression=constante1

= Constante2 ou Constante3

Opération(s) dans le cas expression = constante2 ou 3

= ConstanteX

Opération(s) dans le cas expression=constanteX

SINON

Opération(s) pour aucun des autres choix

Algorithme

- Lire un nombre et afficher la valeur absolue de ce nombre

Demander un nombre

Lire le nombre

SI le nombre < 0 ALORS

Afficher $-1 * \text{Nombre}$

SINON

Afficher le Nombre

Instruction de répétition

```
while (expression_booléenne)  
    instruction;
```

Points clés: une initialisation
 critère d'arrêt
 instruction affectant l'expression booléenne

Instruction de répétition

```
do  
    instruction;  
while (expression_booléenne);
```

Points clés :

- critère d'arrêt
- instruction affectant l'expression booléenne
- le corps de la boucle est exécuté au moins une fois

Instruction de répétition

for (initialisation;^① expression_booléenne;^② actualisation)^④
instruction;^③

- ① L'initialisation est effectuée qu'une seule fois au départ.
 - ② L'expression booléenne est vérifiée, si elle est vraie l'instruction ③ est exécutée, autrement la boucle est terminée.
 - ③ L'instruction à répéter; une fois terminée on passe à l'actualisation.
 - ④ L'actualisation doit permettre la modification de l'expression booléenne.
- ensuite....on boucle 2-3-4-2-3-4-2-3-4-...

Instructions de répétition

while	do-while	for
Il faut tester la condition avant l'exécution de l'instruction.	Il faut exécuter l'instruction au moins une fois avant que la condition soit testée.	Idéalement l'instruction n'affecte pas la condition.
Le nombre de répétitions de l'instruction n'est pas connu d'avance.	Le nombre de répétitions de l'instruction n'est pas connu d'avance.	Le nombre de répétitions de l'instruction est généralement connu d'avance.

Exemple While

```
char reponse = 'n';  
cout << "Voulez vous entrer un nombre ? (o/n) ";  
cin >> reponse;
```

```
int somme = 0;  
while (reponse == 'o') {  
    cout << "Entrer un nombre entier: ";  
    int nombre;  
    cin >> nombre;  
    somme += nombre;
```

```
    cout << "Voulez vous entrer un autre nombre ? (o/n) ";  
    cin >> reponse;  
}  
cout << "La somme est: " << somme << endl;
```

Si la réponse est autre chose que 'o', la boucle ne sera jamais exécutée.

Changement de la valeur utilisée dans l'expression booléenne.

IV_while.cpp

Exemple Do While

```
char reponse = 'n';
```

```
int somme = 0;
```

```
do {
```

```
    cout << "Entrer un nombre entier: ";
```

```
    int nombre;
```

```
    cin >> nombre;
```

```
    somme += nombre;
```

```
    cout << "Voulez vous entrer un autre nombre ? (o/n) ";
```

```
    cin >> reponse;
```

```
} while (reponse == 'o');
```

```
cout << "La somme est: " << somme << endl;
```

La boucle est exécutée au moins une fois.

L'expression est vérifiée à la fin.

IV_dowhile.cpp

Exemple For

```
string phrase = "il pleut, il vente, il neige et il gresille";  
char lettre = 'e';  
int occurrences = 0;  
for (unsigned i = 0; i < phrase.size(); i++) {  
    if (phrase[i] == lettre)  
        occurrences++;  
}  
cout << "Le nombre occurrence de la lettre "  
    << lettre << "' est " << occurrences  
    << endl;
```

On connaît le nombre
d'itérations au moment
où la boucle commence.

Exemple

```
static const int JOUR_INVALIDE = -1;

int main()
{
    ifstream entree("IV_meteo.txt");
    if (entree.fail())
        cout << " Impossible d'ouvrir le fichier" << endl;
    else {
        // Lecture du fichier.
        double temperatures[31]; // temperatures[0] est la température du premier jour.
        int nbJours = 0;
        entree.exceptions(ios::failbit); // Arrête le programme si une erreur se produit.
        while (!ws(entree).eof()) {
            entree >> temperatures[nbJours];
            nbJours++;
        }

        // Déterminer la température maximale du mois et la moyenne des températures.
        int jourMax = 0;
        double temperatureMax = -1.0E2;
        double somme = 0.0;
        for (int jour = 0; jour < nbJours; jour++) {
            somme += temperatures[jour];
            if (temperatures[jour] > temperatureMax) {
                temperatureMax = temperatures[jour];
                jourMax = jour;
            }
        }
    }
}
```

voir la suite dans le fichier IV_meteo.cpp

Lecture d'un fichier

Un fichier contient des valeurs entières. Sur chaque ligne sont inscrites 3 valeurs. Nous ne connaissons pas le nombre de lignes du fichier.

12 34 -5

2 5 7

0 -23 65

95 67 -2

-8 4 76

12 -3 ...

Lecture d'un fichier

```
int main()
{
    ifstream fichier("IV_lecture_fichier.txt");
    fichier.exceptions(ios::failbit);    // Arrête s'il y a erreur.

    int somme = 0;
    // Boucle tant qu'il reste autre chose que des espaces.
    while (!ws(fichier).eof()) {        //-
        int val1, val2, val3;
        fichier >> val1 >> val2 >> val3; //- Lire les données.

        somme += val1 + val2 + val3;    //- Traitement.
    }
    cout << "La somme est " << somme;
    fichier.close();
}
```

IV_lecture_fichier.cpp

Lecture d'un fichier

Lecture du fichier Test.txt

- ☐ Caractère par caractère avec l'opérateur >>
- ☐ Caractère par caractère avec la fonction `get()`
- ☐ Mot par mot avec l'opérateur >>
- ☐ Ligne par ligne avec la fonction `getline()`
- ☐ Lecture robuste teste toujours `fail()` avant d'utiliser la valeur

Fichier: Test.txt

Il etait un petit navire
qui n'avait jamais navigue

OH! HE! OH! HE!

14 + 17 = 31

FIN FIN

Lecture du fichier caractère par caractère avec l'opérateur >>

```
int main()
{
    ifstream ficLu("IV_lecture.txt"); // L'ouverture du fichier
    if (ficLu.fail())                  // est incorrecte
        cout << " Probleme d'ouverture ";
    else {                             // est correcte.
        // Boucle de lecture du fichier, sautant les espaces.
        while (!ws(ficLu).eof()) {
            char carLu;
            // Lecture d'un caractère avec l'opérateur >>.
            ficLu >> carLu;
            cout << carLu;
        }
        ficLu.close();
    }
}
```

IV_lecture_opérateur_char.cpp

Lecture du fichier caractère par caractère avec la fonction get()

```
int main()
{
    ifstream ficLu("IV_lecture.txt"); // L'ouverture du fichier
    if (ficLu.fail())                  // est incorrecte
        cout << " Probleme d'ouverture ";
    else {                             // est correcte.
        // Boucle de lecture du fichier, incluant les espaces.
        while (ficLu.peek() != EOF) {
            char carLu;
            // Lecture d'un caractère avec get.
            ficLu.get(carLu);
            cout << carLu;
        }
        ficLu.close();
    }
}
```

IV_lecture_get.cpp

Lecture du fichier mot par mot à l'aide de l'opérateur >>

```
int main()
{
    ifstream ficLu("IV_lecture.txt"); // L'ouverture du fichier
    if (ficLu.fail())                 // est incorrecte
        cout << " Probleme d'ouverture ";
    else {                             // est correcte.
        // Boucle de lecture du fichier, sautant les espaces.
        while (!ws(ficLu).eof()) {
            string motLu;
            // Lecture d'une chaîne avec l'opérateur >>.
            ficLu >> motLu;
            cout << motLu;
        }
        ficLu.close();
    }
}
```

IV_lecture_opérateur_string.cpp

Lecture du fichier ligne par ligne à l'aide de la fonction getline()

```
int main()
{
    ifstream ficLu("IV_lecture.txt"); // L'ouverture du fichier
    if (ficLu.fail())                 // est incorrecte
        cout << " Probleme d'ouverture ";
    else {                             // est correcte.
        // Boucle de lecture du fichier, incluant les espaces.
        while (ficLu.peek() != EOF) {
            string ligneLue;
            // Lecture d'une chaîne avec la fonction getline.
            getline(ficLu, ligneLue);
            cout << ligneLue << endl;
        }
        ficLu.close();
    }
}
```

IV_lecture_getline.cpp

Comparaison des affichages obtenus

- ☐ Caractère par caractère avec >>

Iletaitunpetitnavirequin'avaitjamaisnavigueOH!HE!OH!HE!14+17=31FINFIN

- ☐ Caractère par caractère avec get()

Il etait un petit navire
qui n'avait jamais navigue
OH! HE! OH! HE!
14 + 17 = 31
FIN FIN

- ☐ Mot par mot avec >>

Iletaitunpetitnavirequin'avaitjamaisnavigueOH!HE!OH!HE!14+17=31FINFIN

- ☐ Ligne par ligne avec getline()

Il etait un petit navire
qui n'avait jamais navigue
OH! HE! OH! HE!
14 + 17 = 31
FIN FIN

Validation d'une entrée

```
int main()
{
    bool erreur = false;
    do {
        cout << "Entrer un entier: ";
        int entier;
        cin >> entier;
        if (cin.fail()) {
            erreur = true;
            cout << "Erreur fatale, cin est inutilisable" << endl;
            cin.clear();
            cin.ignore(80, '\n');
            cout << "cin est maintenant réinitialisé" << endl;
        }
        else {
            erreur = false;
            cout << entier << " lu correctement" << endl;
        }
    } while (erreur);
}
```

IV_lecture_validation.cpp

Chapitre 4.2

STRUCTURES DE DONNÉES

POLYTECHNIQUE
MONTRÉAL

LE GÉNIE
EN PREMIÈRE CLASSE



Les tableaux

- ☐ Une structure homogène constituée d'un nombre déterminé d'éléments de même type.
- ☐ On peut repérer chaque élément à l'aide d'un indice qui sert à indiquer sa position.
- ☐ Déclaration
 - `TypeDesElements nomVariable[dim1][dim2]...[dimN];`

Exemple déclaration d'un tableau

// Accolades vides pour un tableau rempli de zéros:

```
int liste1[3] = { };  
for (int i = 0; i < 3; i++)  
    cout << liste1[i] << ' ';  
// Résultat: 0 0 0
```

// Accolades avec valeurs pour spécifier le contenu:

```
int liste2[3] = { 2, 7, 8 };  
for (int i = 0; i < 3; i++)  
    cout << liste2[i] << ' ';  
// Résultat: 2 7 8
```

Les tableaux

```
int unTableau [10] [5] [15];
```

- ☐ On ne peut pas effectuer de lecture, d’affichage, de comparaison ni d’autres opérations sur des tableaux complets (sauf les chaînes de caractères)
- ☐ Le premier élément d’un tableau est toujours à l’indice 0
- ☐ Accès aux éléments d’un tableau
 - `unTableau[3][2][5] = 12;`

Les tableaux

- `int classe[6] [75];`

Ce tableau contient 450 éléments (6×75). Cependant pour accéder à un élément de ce tableau, l'indice de la première dimension doit être compris entre 0 et 5, l'indice de la deuxième dimension doit être compris entre 0 et 74.

**I
M
P
O
R
T
A
N
T**

Tableau

```
int matrice[6][7];
```

A 6x7 matrix of integers is shown. The elements are as follows:

0	1	0	1	0	1	0
0	1	0	1	1	1	1
1	1	1	1	0	0	0
0	0	0	0	0	0	1
1	1	1	1	1	0	0
0	0	0	1	1	1	1

Arrows indicate the following elements:

- `matrice[2][2]` points to the element 1 in the third row, third column.
- `matrice[1][6]` points to the element 1 in the second row, seventh column.
- `matrice[5][0]` points to the element 0 in the sixth row, first column.

Les tableaux

- ☐ Il n'est pas possible de manipuler le tableau dans son entité.
- ☐ Pour ces opérations, il faut accéder à chacun de ses éléments.
 - ⇒ Affectation d'un tableau à un autre
 - ⇒ Comparaison de deux tableaux
 - ⇒ Lecture et écriture

Traitement des éléments d'un tableau

```
double vecteurA[10], vecteurB[10];
```

```
//- Initialisation du vecteur -  
for (int i = 0; i < 10; i++)  
    vecteurA[i] = 1.0;
```

```
//- Affectation d'un tableau à un autre -  
for (int i = 0; i < 10; i++)  
    vecteurB[i] = vecteurA[i];
```

IV_tableau_vecteur.cpp

Traitement des éléments d'un tableau

```
//- Comparaison de deux tableaux -  
bool estPareil = true;  
for (int i = 0; i < 10 && estPareil; i++)  
    if (vecteurA[i] != vecteurB[i])  
        estPareil = false;
```

```
//- Calculer la norme d'un vecteur -  
double norme = 0.0;  
for (int i = 0; i < 10; i++)  
    norme += pow(vecteurA[i], 2);  
norme = sqrt(norme);
```

IV_tableau_vecteur.cpp

Tableau à 2 dimensions

```
const int DIMENSION1 = 5, DIMENSION2 = 5;
int matrice [DIMENSION1][DIMENSION2];

// Initialisation de la matrice avec valeurs aléatoires.
// Pour passer chaque cases, 2 boucles imbriquées:
for (int i = 0; i < DIMENSION1; i++)      // ligne
    for (int j = 0; j < DIMENSION2; j++) // colonne
        matrice[i][j] = rand();

// Afficher le contenu du tableau 2D.
for (int i = 0; i < DIMENSION1; i++) {
    for (int j = 0; j < DIMENSION2; j++)
        cout << matrice[i][j] << "\t" ;
    cout << endl;
}
```

IV_tableau_2D.cpp

Maximum d'un tableau à 2 dimensions

```
// Trouver la position i,j de la valeur maximum.
int maximum = matrice[0][0];
int positionIMax = 0, positionJMax = 0;
for (int i = 0; i < DIMENSION1; i++) {
    for (int j = 0; j < DIMENSION2; j++) {
        if (matrice[i][j] > maximum) {
            // Conserver la valeur et indices i,j du maximum.
            maximum = matrice[i][j];
            positionIMax = i;
            positionJMax = j;
        }
    }
}
cout << "maximum " << maximum << endl;
cout << "indiceMax Ligne " << positionIMax << endl;
cout << "indiceMax Colonne " << positionJMax << endl;
```

Les enregistrements / structures

- ❏ Un enregistrement est une structure de données formée d'un certain nombre de champs portant chacun un nom et pouvant être de types différents.
- ❏ L'enregistrement permet donc de regrouper, sous un même identificateur, des données diverses, mais logiquement interreliées.

```
❏ struct NomDuType {  
    déclaration des champs;  
};
```



```
struct Adresse { // Définition d'un type Adresse étant un enregistrement
    int    numero;
    string rue, ville, codePostal;
};

struct Membre { // Définition d'un type Membre étant un enregistrement
    string  nom, prenom;
    int     age;
    char    sexe;
    Adresse adresse;
    string  telephone;
    double  montantDu;
};

int main()
{
    // Définition de variables de type Membre
    Membre etudiant, clubSocial[50];
}
```

Les enregistrements

```
struct Point {  
    double x, y;  
};
```

 Déclaration et initialisation d'un enregistrement

```
Point point = { 12.3, -1.34 };
```

 Accès aux champs d'un enregistrement

variable.champ

```
point.x = -2.5; point.y = -6.78;
```

IV_enregistrements_point.cpp

Les enregistrements

- ✎ La seule opération agissant sur l'entité enregistrement est l'affectation entre deux enregistrements de même type.
- ✎ Toutes les autres opérations doivent être définies à l'aide des champs.
- ✎ Peuvent être utiles pour transmettre plusieurs informations en paramètre.

Traitement des éléments d'un enregistrement

```
Point pointA, pointB, vecteur[10];
```

- Initialisation d'un enregistrement

```
pointA.x = 12.3;  
pointA.y = -5.11;
```

- Affectation d'un enregistrement à un autre

```
pointB = pointA;
```

IV_enregistrements_point.cpp

Traitement des éléments d'un enregistrement

- Comparaison de deux enregistrements

```
bool pareil = false;  
if (pointA.x == pointB.x && pointA.y == pointB.y)  
    pareil = true;
```

- Le plus petit x du vecteur

```
double minX = vecteur[0].x;  
for (int i = 1; i < 10; i++)  
    if (vecteur[i].x < minX)  
        minX = vecteur[i].x;
```

IV_enregistrements_point.cpp

Enregistrement et tableau

```
const int MAX_ORDIS = 50;
struct Ordinateur
{
    string format;
    string compagnie;
    double tailleEcranPouces;
    double cpuGHz;
    int memoireRamGiB;
    int memoireDisqueGB;
};
struct ListeOrdi
{
    Ordinateur liste[MAX_ORDIS];
    int nombreOrdi;
};
```

Définit les
caractéristiques d'un ordi

liste: tableau de taille MAX_ORDIS
(nombre maximum d'ordis).
nombreOrdi: nombre d'ordis
actuellement dans la liste.

IV_structTab.cpp

Enregistrement et Tableau

```
ListeOrdi tousOrdi;
fstream fichier;
fichier.open("IV_ordi.txt", ios::in);
if (!fichier.fail()) {
    tousOrdi.nombreOrdi = 0;

    while (!ws(fichier).eof()){
        fichier >> tousOrdi.liste[tousOrdi.nombreOrdi].format
        >> tousOrdi.liste[tousOrdi.nombreOrdi].compagnie
        >> tousOrdi.liste[tousOrdi.nombreOrdi].tailleEcranPouces
        >> tousOrdi.liste[tousOrdi.nombreOrdi].cpuGHz
        >> tousOrdi.liste[tousOrdi.nombreOrdi].memoireRamGiB
        >> tousOrdi.liste[tousOrdi.nombreOrdi].memoireDisqueGB;
        tousOrdi.nombreOrdi++;
    }
}
else
    cout << " fichier introuvable" << endl;
```

IV_structTab.cpp