

Chapitre 4

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

```
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

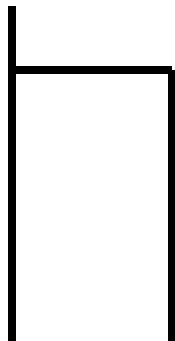
Imbrication d'instructions de décision

```
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

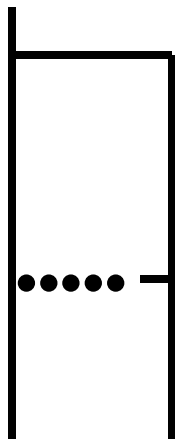
Structures décisionnelles en pseudo-code schématique



SI Expression Booléenne ALORS

Opération 1

Opération 2



SI Expression Booléenne ALORS

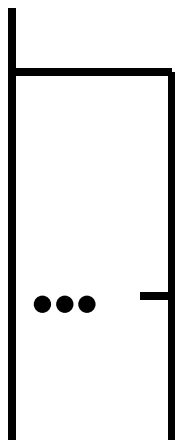
Opération(s) dans le cas Expression VRAIE

.....
SINON

Opération(s) dans le cas Expression FAUSSE

Algorithme

Structures décisionnelles en pseudo-code schématique



SI Expr Booléenne1 ALORS

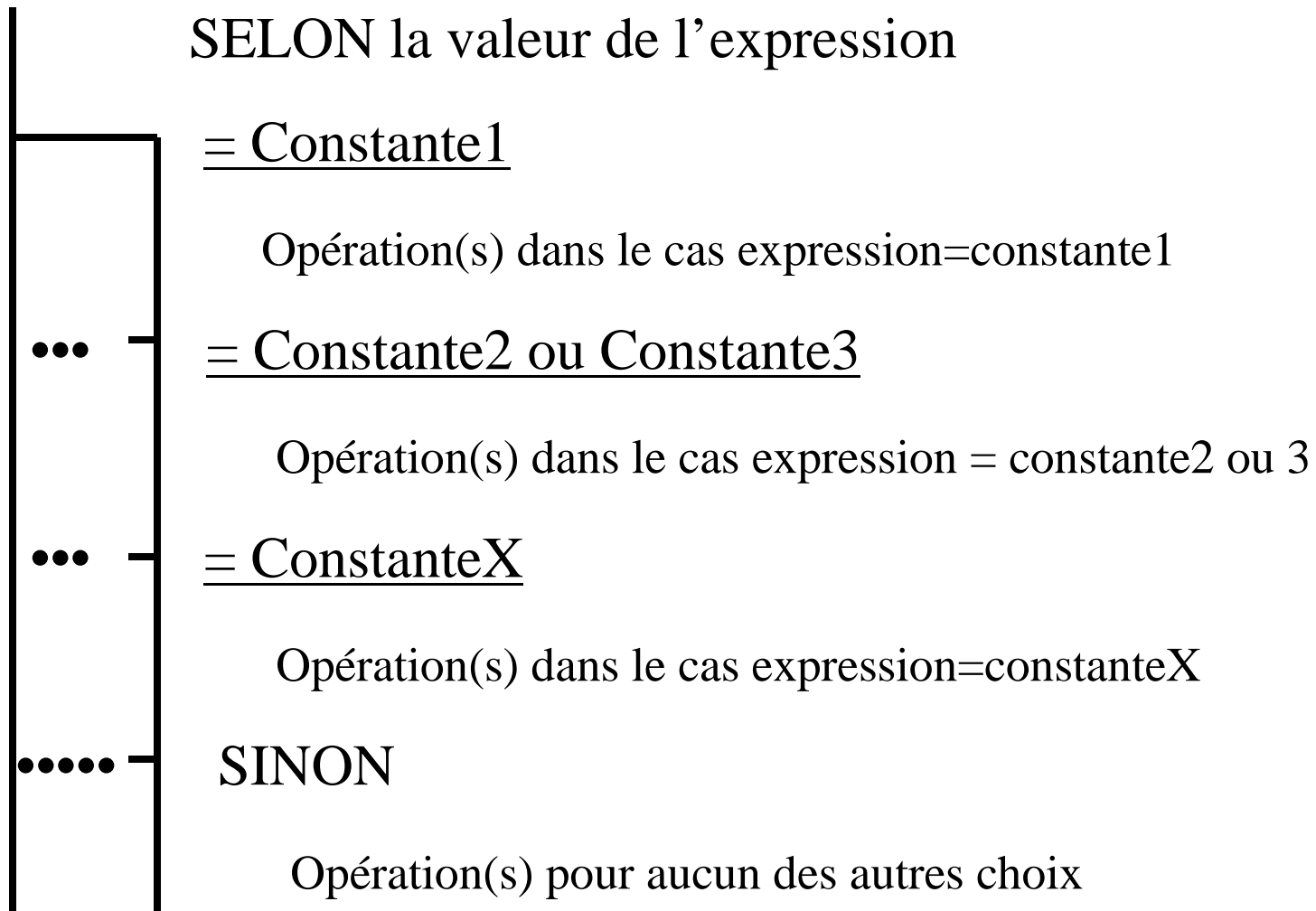
Opération(s) dans le cas Expr.1 VRAIE

SINON SI Expr Booléenne2 ALORS

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

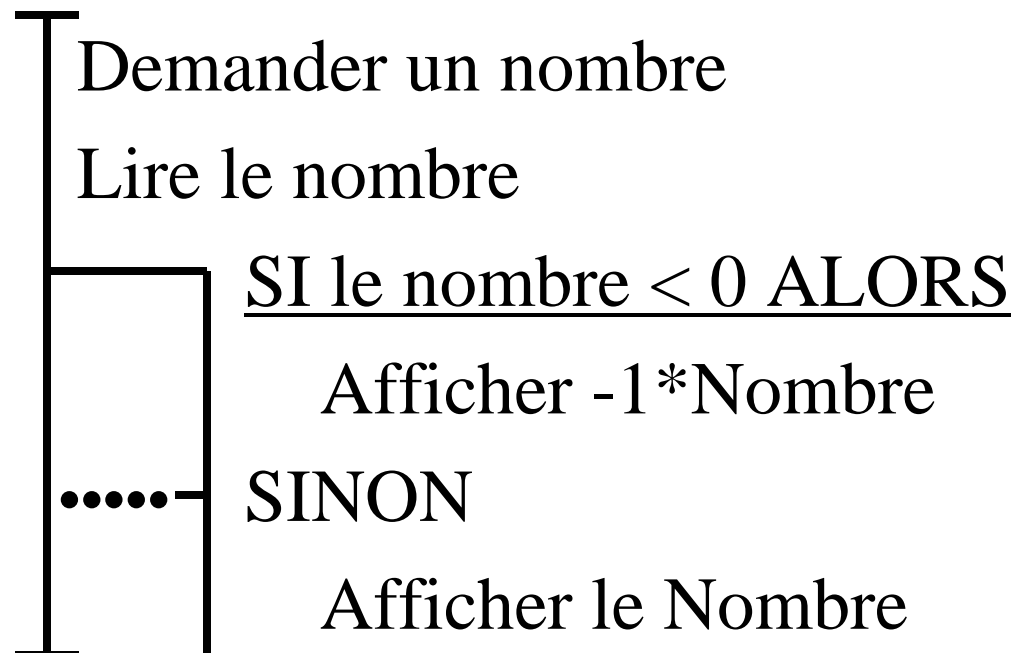
Algorithme

Structures décisionnelles en pseudo-code schématique



Algorithme

- Lire un nombre et afficher la valeur absolue de ce 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 de 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 + 1;
            }
        }
    }
}
```

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

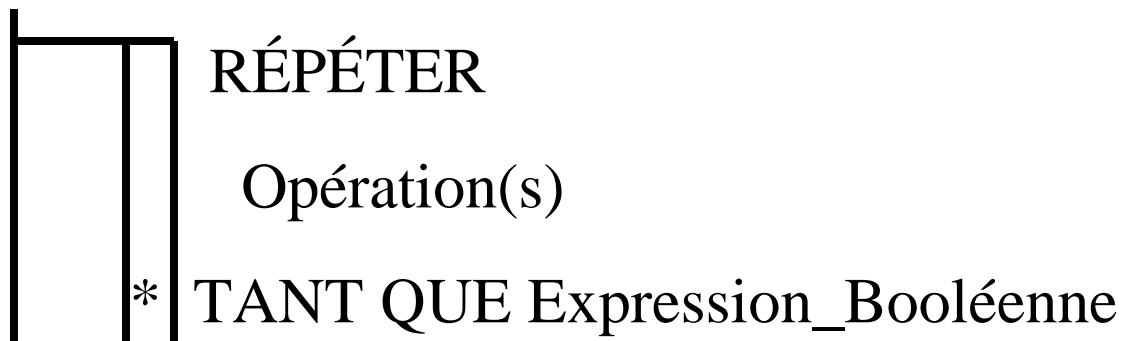
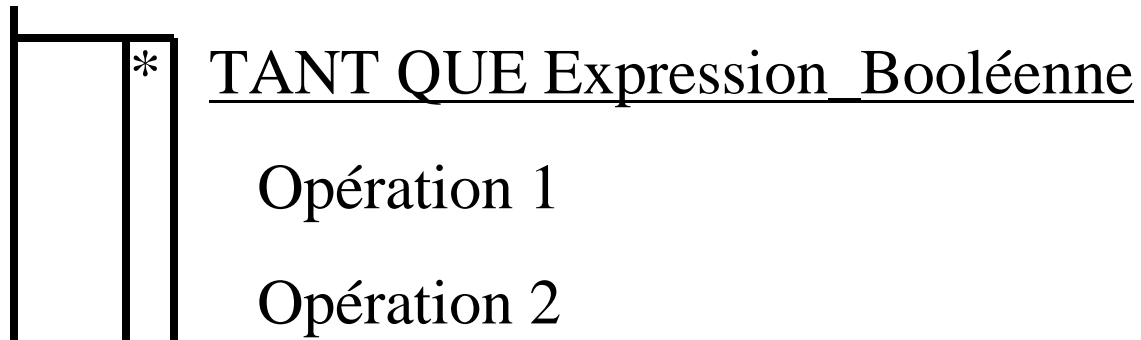
Validation d'une entrée

```
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

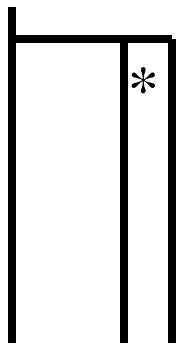
Algorithme

Structures décisionnelles en pseudo-code schématique



Algorithme

Structures décisionnelles en pseudo-code schématique



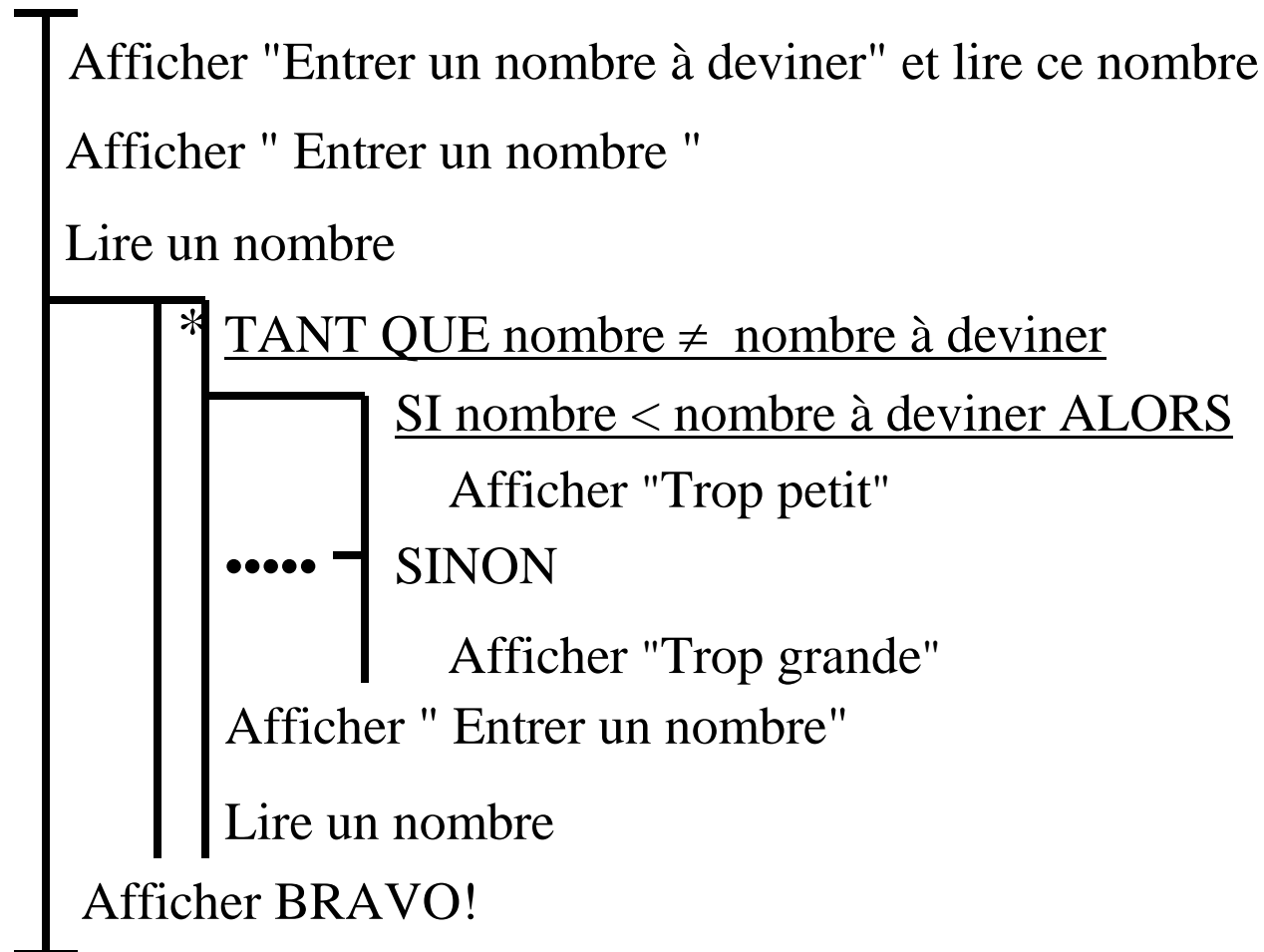
POUR les valeurs à considérer

Opération 1

Opération 2

Algorithme

Deviner un nombre (Version 1)



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 point from the following labels to specific elements in the matrix:

- `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];
```

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

Type énumération

 Permet d'expliciter les valeurs que pourra prendre une variable

 Augmente la lisibilité d'un programme par l'utilisation d'identificateur significatif

 Déclaration

⇒ `enum TypeEnum { Id1, Id2, Id3,..., IdN };`


⇒ `TypeEnum varEnum;`

 Exemple

⇒ `enum Direction { NORD, SUD, EST, OUEST };`

⇒ `Direction direction;`

Type énumération

 Le compilateur attribue une valeur ordinale à chaque identificateur de l'énumération

– enum Direction { NORD, SUD, EST, OUEST };

- NORD vaut 0
- SUD vaut 1
- EST vaut 2
- OUEST vaut 3




Ces identificateurs
sont des constantes

Type énumération

 Il est possible de préciser la valeur entière associée à un identificateur de l'énumération



– enum Couleur { BLEU=1, VERT, ROUGE=4, JAUNE=14 };

- BLEU vaut 1
- VERT vaut 2
- ROUGE vaut 4
- JAUNE vaut 14


 Comme il existe une relation d'ordre, les opérateurs relationnels s'appliquent sur des variables de type énumération


- ==, !=, >, <, >=, <=

Type énumération

-  Il n'est pas possible de lire du clavier ou d'un fichier texte une valeur appartenant à un type énumération.
-  L'affichage correspond à la valeur entière associée à l'identificateur de l'énumération.

Exemple avec le type énumération

 Un point est positionné aléatoirement dans une région spécifique du plan $[0..20, 0..20]$

 Un déplacement est effectué uniquement si le point demeure dans la région spécifique

Exemple avec le type énumération

```
int main()
{
    enum Direction { NORD, SUD, EST, OUEST };

    srand(unsigned(time(0)));
    int x = rand() % 21;
    int y = rand() % 21;

    cout << "Positon initiale du point: ";
    cout << '(' << x << ',' << y << ')' << endl;

    Direction direction = Direction(rand() % 4);
    switch (direction) {
        case NORD : if (y < 20) ++y; break;
        case SUD  : if (y > 0)  --y; break;
        case EST  : if (x < 20) ++x; break;
        case OUEST : if (x > 0)  --x;
    }

    cout << "Positon finale du point  : ";
    cout << '(' << x << ',' << y << ')' << endl;
}
```

Énumérations et tableaux

- ❑ L'utilisation d'un indice de type énumération peut rendre très explicite l'accès à un élément du tableau.

```
enum Departement { CIVIL, MECA , ELEC,  
                  INFO, NBDEPARTEMENTS };  
enum Grade { BING, MSCA, PHD, NBGRADES };  
enum Sexe { FEMME, HOMME, NBSEXES };  
int gradues[NBDEPARTEMENTS][NBGRADES][NBSEXES];
```

Énumérations et tableaux

- Pour déterminer le nombre de Ph.D. décernés par le département de génie informatique:

```
int nbrePhdInformatique =  
    gradues[INFO][PHD][FEMME] +  
    gradues[INFO][PHD][HOMME];
```

IV_tableau_enum.cpp

Énumérations, tableaux et boucles

☐ Utilisation du type énumération comme indice

```
enum Jour { LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI,  
            SAMEDI, DIMANCHE, NBJOURS };  
enum Periode { AM, PM, SOIR, NBPERIODES };  
  
string agenda[NBJOURS][NBPERIODES];  
static const string LIBRE = "----";  
  
for (Jour jour = LUNDI; jour <= DIMANCHE; jour = Jour(jour+1))  
    for (Periode periode = AM; periode <= SOIR;  
         periode = Periode(periode+1))  
        agenda[jour][periode] = LIBRE;
```

IV_tableau_enum_for.cpp

Lecture/affichage d'un type énuméré

```
// Lecture d'un type énuméré en utilisant un tableau:
static const string nomJours[] = {
    "lundi", "mardi", "mercredi", "jeudi", "vendredi",
    "samedi", "dimanche"
};
cout << "Pour quel jour voulez-vous l'horaire? (lundi .. dimanche) ";
string jourTexte;
cin >> jourTexte;
Jour jour = LUNDI;
while (jour <= DIMANCHE && jourTexte != nomJours[jour])
    jour = Jour(jour+1);

// Affichage d'un type énuméré en utilisant un tableau de string:
assert(jour >= LUNDI && jour <= DIMANCHE); // Erreur si jour pas valide.
cout << "Le jour est " << nomJours[jour] << endl;
```


Lecture/affichage d'un type énuméré

```
// Lecture d'un type énuméré en utilisant un switch/case:
switch (jourTexte[0]) { // Teste la première lettre.
    case 'l': jour = LUNDI; break;
    case 'm': // Pour distinguer "mardi" et "mercredi",
        switch (jourTexte[1]) { // teste la deuxième lettre.
            case 'a': jour = MARDI; break;
            case 'e': jour = MERCREDI; break;
        }
        break;
    case 'j': jour = JEUDI; break;
    case 'v': jour = VENDREDI; break;
    case 's': jour = SAMEDI; break;
    case 'd': jour = DIMANCHE; break;
}

// Affichage d'un type énuméré en utilisant un switch/case:
switch (jour) {
    case LUNDI: cout << "lundi"; break;
    case MARDI: cout << "mardi"; break;
    case MERCREDI: cout << "mercredi"; break;
    case JEUDI: cout << "jeudi"; break;
    case VENDREDI: cout << "vendredi"; break;
    case SAMEDI: cout << "samedi"; break;
    case DIMANCHE: cout << "dimanche"; break;
    default: cout << "invalidé"; break;
}
```

IV_tableau_enum_for.cpp