

5 Les outils algorithmiques

Peter Schlagheck

Université de Liège

Ces notes ont pour seule vocation d'être utilisées par les étudiants dans le cadre de leur cursus au sein de l'Université de Liège. Aucun autre usage ni diffusion n'est autorisé, sous peine de constituer une violation de la Loi du 30 juin 1994 relative au droit d'auteur.

5 Les outils algorithmiques

5.1 Le branchement conditionnel: `if`

5.2 Les opérateurs de comparaison

5.3 Les opérateurs logiques

5.4 Les blocs: `{ . . . }`

5.5 Les boucles: `while`

5.6 Les itérations: `for`

5.1 Le branchement conditionnel

Exemple: éviter la division par zéro

```
int n, m;  
cout << "entrez deux nombres entiers:";  
cin >> n >> m;  
cout << "voila leur division: " << n / m << endl;  
cout << "voila leur produit: " << n * m << endl;
```

5.1 Le branchement conditionnel

Exemple: éviter la division par zéro

```
int n, m;  
cout << "entrez deux nombres entiers:";  
cin >> n >> m;  
cout << "voila leur division: " << n / m << endl;  
cout << "voila leur produit: " << n * m << endl;
```

Exécution:

```
entrez deux nombres entiers: 3  
2  
voila leur division: 1  
voila leur produit: 6
```

5.1 Le branchement conditionnel

Exemple: éviter la division par zéro

```
int n, m;  
cout << "entrez deux nombres entiers:";  
cin >> n >> m;  
cout << "voila leur division: " << n / m << endl;  
cout << "voila leur produit: " << n * m << endl;
```

Exécution:

```
entrez deux nombres entiers: 3  
0  
division by zero
```

→ **run-time error** (pas détectable par le compilateur):
le programme est avorté juste après l'opération n/m :

5.1 Le branchement conditionnel

Exemple: éviter la division par zéro

```
int n, m;  
cout << "entrez deux nombres entiers:";  
cin >> n >> m;  
cout << "voila leur division: " << n / m << endl;  
cout << "voila leur produit: " << n * m << endl;
```

Solution:

faire le premier affichage seulement **si m ne vaut pas zéro**

→ l'opérateur **if**

5.1 Le branchement conditionnel

Exemple: éviter la division par zéro

```
int n, m;  
cout << "entrez deux nombres entiers:";  
cin >> n >> m;  
if ( m )  
    cout << "voila leur division: " << n / m << endl;  
cout << "voila leur produit: " << n * m << endl;
```

5.1 Le branchement conditionnel

Exemple: éviter la division par zéro

```
int n, m;  
cout << "entrez deux nombres entiers:";  
cin >> n >> m;  
if ( m )  
    cout << "voila leur division: " << n / m << endl;  
cout << "voila leur produit: " << n * m << endl;
```

Exécution:

```
entrez deux nombres entiers: 3  
2  
voila leur division: 1  
voila leur produit: 6
```


5.1 Le branchement conditionnel

Exemple: éviter la division par zéro

```
int n, m;  
cout << "entrez deux nombres entiers:";  
cin >> n >> m;  
if ( m )  
    cout << "voila leur division: " << n / m << endl;  
cout << "voila leur produit: " << n * m << endl;
```

Exécution:

```
entrez deux nombres entiers: 3  
0  
voila leur produit: 0
```

5.1 Le branchement conditionnel

Exemple: éviter la division par zéro

```
int n, m;  
cout << "entrez deux nombres entiers:";  
cin >> n >> m;  
if ( m )  
    cout << "voila leur division: " << n / m << endl;  
cout << "voila leur produit: " << n * m << endl;
```

Structure générale:

```
if ( <expression> ) <instruction>;  
→ <instruction> sera effectuée seulement  
   si <expression> ne vaut pas zéro  
   (<expression> peut être de n'importe quel type)
```

5.1 Le branchement conditionnel

Exemple: éviter la division par zéro

```
int n, m;  
cout << "entrez deux nombres entiers:";  
cin >> n >> m;  
if ( m )  
    cout << "voila leur division: " << n / m << endl;  
cout << "voila leur produit: " << n * m << endl;
```

...et que faire sinon ?

→ l'opérateur **else**

5.1 Le branchement conditionnel

Exemple: éviter la division par zéro

```
int n, m;  
cout << "entrez deux nombres entiers:";  
cin >> n >> m;  
if ( m )  
    cout << "voila leur division: " << n / m << endl;  
else  
    cout << "division par zero!" << endl;  
cout << "voila leur produit: " << n * m << endl;
```

5.1 Le branchement conditionnel

Exemple: éviter la division par zéro

```
int n, m;  
cout << "entrez deux nombres entiers:";  
cin >> n >> m;  
if ( m )  
    cout << "voila leur division: " << n / m << endl;  
else  
    cout << "division par zero!" << endl;  
cout << "voila leur produit: " << n * m << endl;
```

Exécution:

```
entrez deux nombres entiers: 3  
0  
division par zero!  
voila leur produit: 0
```

5.1 Le branchement conditionnel

Exemple: éviter la division par zéro

```
int n, m;  
cout << "entrez deux nombres entiers:";  
cin >> n >> m;  
if ( m )  
    cout << "voila leur division: " << n / m << endl;  
else  
    cout << "division par zero!" << endl;  
cout << "voila leur produit: " << n * m << endl;
```

Exécution:

```
entrez deux nombres entiers: 3  
2  
voila leur division: 1  
voila leur produit: 6
```

5.1 Le branchement conditionnel

Exemple: éviter la division par zéro

```
int n, m;
cout << "entrez deux nombres entiers:";
cin >> n >> m;
if ( m )
    cout << "voila leur division: " << n / m << endl;
else
    cout << "division par zero!" << endl;
cout << "voila leur produit: " << n * m << endl;
```

Structure générale:

```
if ( <expression> ) <instruction>;
else <instruction2>;
```

→ <instruction> sera effectuée seulement
si <expression> ne vaut pas zéro

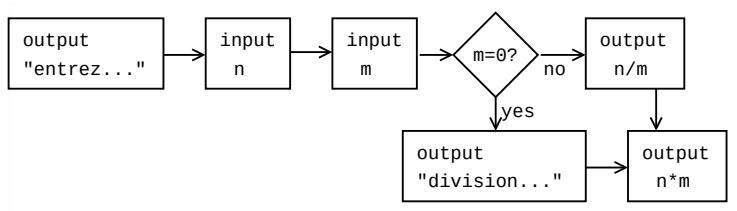
→ dans l'autre cas, <instruction2> sera effectuée au lieu

5.1 Le branchement conditionnel

Exemple: éviter la division par zéro

```
int n, m;  
cout << "entrez deux nombres entiers:";  
cin >> n >> m;  
if ( m )  
    cout << "voila leur division: " << n / m << endl;  
else  
    cout << "division par zero!" << endl;  
cout << "voila leur produit: " << n * m << endl;
```

Représentation diagrammatique de l'algorithme:



Comment réaliser des branchements à des conditions plus compliquées ?

p. ex.: effectuer l'instruction `cout << n / m << endl;`
si `m` est positif

Comment réaliser des branchements à des conditions plus compliquées ?

→ utilisation des **opérateurs de comparaison**

5.2 Les opérateurs de comparaison

`==` égalité: `a == b`

`>` plus grand: `a > b`

`<` plus petit: `a < b`

5.2 Les opérateurs de comparaison

`==` égalité: `a == b`
`>` plus grand: `a > b`
`<` plus petit: `a < b`

```
int a;  
cout << "entrez un nombre:" << endl;  
cin >> a;  
if ( a == 3 )  
    cout << "a est egal a 3" << endl;  
if ( a < 3 )  
    cout << "a est plus petit que 3" << endl;  
if ( a > 3 )  
    cout << "a est plus grand que 3" << endl;
```

5.2 Les opérateurs de comparaison

`==` égalité: `a == b`
`>` plus grand: `a > b`
`<` plus petit: `a < b`

```
int a;  
cout << "entrez un nombre:" << endl;  
cin >> a;  
if ( a == 3 )  
    cout << "a est egal a 3" << endl;  
if ( a < 3 )  
    cout << "a est plus petit que 3" << endl;  
if ( a > 3 )  
    cout << "a est plus grand que 3" << endl;
```

Exécution:

entrez un nombre: 2

5.2 Les opérateurs de comparaison

`==` égalité: `a == b`
`>` plus grand: `a > b`
`<` plus petit: `a < b`

```
int a;  
cout << "entrez un nombre:" << endl;  
cin >> a;  
if ( a == 3 )  
    cout << "a est egal a 3" << endl;  
if ( a < 3 )  
    cout << "a est plus petit que 3" << endl;  
if ( a > 3 )  
    cout << "a est plus grand que 3" << endl;
```

Exécution:

```
entrez un nombre: 2  
a est plus petit que 3
```

5.2 Les opérateurs de comparaison

`==` égalité: `a == b`
`>` plus grand: `a > b`
`<` plus petit: `a < b`

```
int a;  
cout << "entrez un nombre:" << endl;  
cin >> a;  
if ( a == 3 )  
    cout << "a est egal a 3" << endl;  
if ( a < 3 )  
    cout << "a est plus petit que 3" << endl;  
if ( a > 3 )  
    cout << "a est plus grand que 3" << endl;
```

Exécution:

entrez un nombre: 3

5.2 Les opérateurs de comparaison

`==` égalité: `a == b`
`>` plus grand: `a > b`
`<` plus petit: `a < b`

```
int a;  
cout << "entrez un nombre:" << endl;  
cin >> a;  
if ( a == 3 )  
    cout << "a est egal a 3" << endl;  
if ( a < 3 )  
    cout << "a est plus petit que 3" << endl;  
if ( a > 3 )  
    cout << "a est plus grand que 3" << endl;
```

Exécution:

```
entrez un nombre: 3  
a est egal a 3
```


5.2 Les opérateurs de comparaison

`==` égalité: `a == b`
`>` plus grand: `a > b`
`<` plus petit: `a < b`

```
int a;  
cout << "entrez un nombre:" << endl;  
cin >> a;  
if ( a == 3 )  
    cout << "a est egal a 3" << endl;  
if ( a < 3 )  
    cout << "a est plus petit que 3" << endl;  
if ( a > 3 )  
    cout << "a est plus grand que 3" << endl;
```

Exécution:

entrez un nombre: 4

5.2 Les opérateurs de comparaison

`==` égalité: `a == b`
`>` plus grand: `a > b`
`<` plus petit: `a < b`

```
int a;  
cout << "entrez un nombre:" << endl;  
cin >> a;  
if ( a == 3 )  
    cout << "a est egal a 3" << endl;  
if ( a < 3 )  
    cout << "a est plus petit que 3" << endl;  
if ( a > 3 )  
    cout << "a est plus grand que 3" << endl;
```

Exécution:

```
entrez un nombre: 4  
a est plus grand que 3
```

5.2 Les opérateurs de comparaison

`==` égalité: `a == b`
`>` plus grand: `a > b`
`<` plus petit: `a < b`

```
int a;  
cout << "entrez un nombre:" << endl;  
cin >> a;  
if ( a == 3 )  
    cout << "a est egal a 3" << endl;  
if ( a < 3 )  
    cout << "a est plus petit que 3" << endl;  
if ( a > 3 )  
    cout << "a est plus grand que 3" << endl;
```

→ ça fonctionne aussi pour des nombres flottants
et des caractères

5.2 Les opérateurs de comparaison

`==` égalité: `a == b`
`>` plus grand: `a > b`
`<` plus petit: `a < b`

```
int a;  
cout << "entrez un nombre:" << endl;  
cin >> a;  
if ( a == 3 )  
    cout << "a est egal a 3" << endl;  
if ( a < 3 )  
    cout << "a est plus petit que 3" << endl;  
if ( a > 3 )  
    cout << "a est plus grand que 3" << endl;
```

Attention: n'écrivez pas `if ( a = 3 )` !

5.2 Les opérateurs de comparaison

`==` égalité: `a == b`
`>` plus grand: `a > b`
`<` plus petit: `a < b`

```
int a;  
cout << "entrez un nombre:" << endl;  
cin >> a;  
if ( a = 3 )  
    cout << "a est egal a 3" << endl;  
if ( a < 3 )  
    cout << "a est plus petit que 3" << endl;  
if ( a > 3 )  
    cout << "a est plus grand que 3" << endl;
```

5.2 Les opérateurs de comparaison

`==` égalité: `a == b`
`>` plus grand: `a > b`
`<` plus petit: `a < b`

```
int a;  
cout << "entrez un nombre:" << endl;  
cin >> a;  
if ( a = 3 )  
    cout << "a est egal a 3" << endl;  
if ( a < 3 )  
    cout << "a est plus petit que 3" << endl;  
if ( a > 3 )  
    cout << "a est plus grand que 3" << endl;
```

Exécution:

entrez un nombre: 2

5.2 Les opérateurs de comparaison

`==` égalité: `a == b`
`>` plus grand: `a > b`
`<` plus petit: `a < b`

```
int a;  
cout << "entrez un nombre:" << endl;  
cin >> a;  
if ( a = 3 )  
    cout << "a est egal a 3" << endl;  
if ( a < 3 )  
    cout << "a est plus petit que 3" << endl;  
if ( a > 3 )  
    cout << "a est plus grand que 3" << endl;
```

Exécution:

```
entrez un nombre: 2  
a est egal a 3
```

5.2 Les opérateurs de comparaison

`==` égalité: `a == b`
`>` plus grand: `a > b`
`<` plus petit: `a < b`

```
int a;  
cout << "entrez un nombre:" << endl;  
cin >> a;  
if ( a = 3 )  
    cout << "a est egal a 3" << endl;  
if ( a < 3 )  
    cout << "a est plus petit que 3" << endl;  
if ( a > 3 )  
    cout << "a est plus grand que 3" << endl;
```

`if ( a = 3 )` effectue d'abord l'opération de copie `a = 3`
et verifie ensuite si `a` vaut zéro ou non

5.2 Les opérateurs de comparaison

`==` égalité: `a == b`

`>` plus grand: `a > b`

`<` plus petit: `a < b`

`!=` ne pas égal

`<=` plus petit ou égal

`>=` plus grand ou égal

5.2 Les opérateurs de comparaison

<code>==</code>	égalité:	<code>a == b</code>	<code>!=</code>	ne pas égal
<code>></code>	plus grand:	<code>a > b</code>	<code><=</code>	plus petit ou égal
<code><</code>	plus petit:	<code>a < b</code>	<code>>=</code>	plus grand ou égal

```
int a;  
cout << "entrez un nombre:" << endl;  
cin >> a;  
if ( a != 3 )  
    cout << "a n'est pas egal a 3" << endl;  
if ( a >= 3 )  
    cout << "a est egal a ou plus grand que 3" << endl;  
if ( a <= 3 )  
    cout << "a est egal a ou plus petit que 3" << endl;
```

5.2 Les opérateurs de comparaison

<code>==</code>	égalité:	<code>a == b</code>	<code>!=</code>	ne pas égal
<code>></code>	plus grand:	<code>a > b</code>	<code><=</code>	plus petit ou égal
<code><</code>	plus petit:	<code>a < b</code>	<code>>=</code>	plus grand ou égal

```
int a;  
cout << "entrez un nombre:" << endl;  
cin >> a;  
if ( a != 3 )  
    cout << "a n'est pas egal a 3" << endl;  
if ( a >= 3 )  
    cout << "a est egal a ou plus grand que 3" << endl;  
if ( a <= 3 )  
    cout << "a est egal a ou plus petit que 3" << endl;
```

Exécution:

entrez un nombre: 2

5.2 Les opérateurs de comparaison

<code>==</code>	égalité:	<code>a == b</code>	<code>!=</code>	ne pas égal
<code>></code>	plus grand:	<code>a > b</code>	<code><=</code>	plus petit ou égal
<code><</code>	plus petit:	<code>a < b</code>	<code>>=</code>	plus grand ou égal

```
int a;
cout << "entrez un nombre:" << endl;
cin >> a;
if ( a != 3 )
    cout << "a n'est pas egal a 3" << endl;
if ( a >= 3 )
    cout << "a est egal a ou plus grand que 3" << endl;
if ( a <= 3 )
    cout << "a est egal a ou plus petit que 3" << endl;
```

Exécution:

```
entrez un nombre: 2
a n'est pas egal a 3
a est egal a ou plus petit que 3
```

5.2 Les opérateurs de comparaison

<code>==</code>	égalité:	<code>a == b</code>	<code>!=</code>	ne pas égal
<code>></code>	plus grand:	<code>a > b</code>	<code><=</code>	plus petit ou égal
<code><</code>	plus petit:	<code>a < b</code>	<code>>=</code>	plus grand ou égal

```
int a;
cout << "entrez un nombre:" << endl;
cin >> a;
if ( a != 3 )
    cout << "a n'est pas egal a 3" << endl;
if ( a >= 3 )
    cout << "a est egal a ou plus grand que 3" << endl;
if ( a <= 3 )
    cout << "a est egal a ou plus petit que 3" << endl;
```

Exécution:

entrez un nombre: 3

5.2 Les opérateurs de comparaison

<code>==</code>	égalité:	<code>a == b</code>	<code>!=</code>	ne pas égal
<code>></code>	plus grand:	<code>a > b</code>	<code><=</code>	plus petit ou égal
<code><</code>	plus petit:	<code>a < b</code>	<code>>=</code>	plus grand ou égal

```
int a;  
cout << "entrez un nombre:" << endl;  
cin >> a;  
if ( a != 3 )  
    cout << "a n'est pas egal a 3" << endl;  
if ( a >= 3 )  
    cout << "a est egal a ou plus grand que 3" << endl;  
if ( a <= 3 )  
    cout << "a est egal a ou plus petit que 3" << endl;
```

Exécution:

```
entrez un nombre: 3  
a est egal a ou plus grand que 3  
a est egal a ou plus petit que 3
```

5.2 Les opérateurs de comparaison

<code>==</code>	égalité:	<code>a == b</code>	<code>!=</code>	ne pas égal
<code>></code>	plus grand:	<code>a > b</code>	<code><=</code>	plus petit ou égal
<code><</code>	plus petit:	<code>a < b</code>	<code>>=</code>	plus grand ou égal

```
int a;  
cout << "entrez un nombre:" << endl;  
cin >> a;  
if ( a != 3 )  
    cout << "a n'est pas egal a 3" << endl;  
if ( a >= 3 )  
    cout << "a est egal a ou plus grand que 3" << endl;  
if ( a <= 3 )  
    cout << "a est egal a ou plus petit que 3" << endl;
```

Exécution:

entrez un nombre: 4

5.2 Les opérateurs de comparaison

<code>==</code>	égalité:	<code>a == b</code>	<code>!=</code>	ne pas égal
<code>></code>	plus grand:	<code>a > b</code>	<code><=</code>	plus petit ou égal
<code><</code>	plus petit:	<code>a < b</code>	<code>>=</code>	plus grand ou égal

```
int a;  
cout << "entrez un nombre:" << endl;  
cin >> a;  
if ( a != 3 )  
    cout << "a n'est pas egal a 3" << endl;  
if ( a >= 3 )  
    cout << "a est egal a ou plus grand que 3" << endl;  
if ( a <= 3 )  
    cout << "a est egal a ou plus petit que 3" << endl;
```

Exécution:

```
entrez un nombre: 4  
a n'est pas egal a 3  
a est egal a ou plus grand que 3
```


5.2 Les opérateurs de comparaison

<code>==</code>	égalité:	<code>a == b</code>	<code>!=</code>	ne pas égal
<code>></code>	plus grand:	<code>a > b</code>	<code><=</code>	plus petit ou égal
<code><</code>	plus petit:	<code>a < b</code>	<code>>=</code>	plus grand ou égal

```
int a;  
cout << "entrez un nombre:" << endl;  
cin >> a;  
if ( a != 3 )  
    cout << "a n'est pas egal a 3" << endl;  
if ( a >= 3 )  
    cout << "a est egal a ou plus grand que 3" << endl;  
if ( a <= 3 )  
    cout << "a est egal a ou plus petit que 3" << endl;
```

→ formellement, `==`, `<`, `>`, `!=`, `>=`, `<=` sont des opérateurs qui rendent la valeur 1 si la condition est satisfaite, et la valeur 0 sinon

5.2 Les opérateurs de comparaison

==	égalité:	a == b	!=	ne pas égal
>	plus grand:	a > b	<=	plus petit ou égal
<	plus petit:	a < b	>=	plus grand ou égal

```
int a;  
cout << "entrez un nombre:" << endl;  
cin >> a;  
cout << ( a > 3 ) << " " << ( a == 3 ) << " "  
    << ( a < 3 ) << " " << ( a <= 3 ) << " "  
    << ( a != 3 ) << " " << ( a >= 3 ) << endl;
```

5.2 Les opérateurs de comparaison

==	égalité:	a == b	!=	ne pas égal
>	plus grand:	a > b	<=	plus petit ou égal
<	plus petit:	a < b	>=	plus grand ou égal

```
int a;  
cout << "entrez un nombre:" << endl;  
cin >> a;  
cout << ( a > 3 ) << " " << ( a == 3 ) << " "  
      << ( a < 3 ) << " " << ( a <= 3 ) << " "  
      << ( a != 3 ) << " " << ( a >= 3 ) << endl;
```

Exécution:

```
entrez un nombre: 2  
0 0 1 1 1 0
```

5.2 Les opérateurs de comparaison

==	égalité:	a == b	!=	ne pas égal
>	plus grand:	a > b	<=	plus petit ou égal
<	plus petit:	a < b	>=	plus grand ou égal

```
int a;  
cout << "entrez un nombre:" << endl;  
cin >> a;  
cout << ( a > 3 ) << " " << ( a == 3 ) << " "  
      << ( a < 3 ) << " " << ( a <= 3 ) << " "  
      << ( a != 3 ) << " " << ( a >= 3 ) << endl;
```

Exécution:

```
entrez un nombre: 3  
0 1 0 1 0 1
```

5.2 Les opérateurs de comparaison

==	égalité:	a == b	!=	ne pas égal
>	plus grand:	a > b	<=	plus petit ou égal
<	plus petit:	a < b	>=	plus grand ou égal

```
int a;  
cout << "entrez un nombre:" << endl;  
cin >> a;  
cout << ( a > 3 ) << " " << ( a == 3 ) << " "  
      << ( a < 3 ) << " " << ( a <= 3 ) << " "  
      << ( a != 3 ) << " " << ( a >= 3 ) << endl;
```

Exécution:

```
entrez un nombre: 4  
1 0 0 0 1 1
```

La valeur des instructions

Pour votre information:

aussi des instructions peuvent rendre une valeur

```
int n = 2;  
if ( n = 3 )  
    cout << n << endl;
```

Exécution:

3

La valeur des instructions

Pour votre information:

aussi des instructions peuvent rendre une valeur

`n = 3` effectue une opération de copie
et rend la valeur 3

`n += 3` effectue une opération d'affectation
et rend la valeur de `n` après cette opération

`n++` effectue une opération d'incrément
et rend la valeur de `n` **avant** cette opération

`++n` effectue une opération d'incrément
et rend la valeur de `n` **après** cette opération

La valeur des instructions

Pour votre information:

aussi des instructions peuvent rendre une valeur

```
int a, b, c;  
a = b = c = 3;  
cout << a << " " << b << " " << c << endl;
```

Exécution:

3 3 3

La valeur des instructions

Pour votre information:

aussi des instructions peuvent rendre une valeur

```
int n = 1;  
int m = ( n += 3 );  
cout << n << " " << ( m++ ) << endl;  
cout << m << endl;
```

Exécution:

4 4

5

La valeur des instructions

Pour votre information:

aussi des instructions peuvent rendre une valeur

```
int n = 1;  
int m = ( n += 3 );  
cout << n << " " << ( ++m ) << endl;  
cout << m << endl;
```

Exécution:

4 5

5

La valeur des instructions

Pour votre information:

aussi des instructions peuvent rendre une valeur

```
int n = 1;  
int m = ( n += 3 );  
cout << n << " " << ( m++ ) << endl;  
cout << m << endl;
```

→ ça permet d'écrire des programmes très compacts ...
(...et très illisibles)

5.3 Les opérateurs logiques

`&&` *AND* (et) `a && b`

`||` *OR* (ou) `a || b`

`!` *NOT* (ne pas) `!a`

5.3 Les opérateurs logiques

`&&` *AND* (et) `a && b`

`||` *OR* (ou) `a || b`

`!` *NOT* (ne pas) `!a`

```
int a, b;  
cin >> a >> b;  
cout << a && b << " " << a || b  
     << " " << !a << endl;
```

5.3 Les opérateurs logiques

`&&` *AND* (et) `a && b`

`||` *OR* (ou) `a || b`

`!` *NOT* (ne pas) `!a`

```
int a, b;  
cin >> a >> b;  
cout << a && b << " " << a || b  
    << " " << !a << endl;
```

Exécution:

2

1

1 1 0

5.3 Les opérateurs logiques

`&&` *AND* (et) `a && b`

`||` *OR* (ou) `a || b`

`!` *NOT* (ne pas) `!a`

```
int a, b;  
cin >> a >> b;  
cout << a && b << " " << a || b  
     << " " << !a << endl;
```

Exécution:

```
2  
0  
0 1 0
```

5.3 Les opérateurs logiques

`&&` *AND* (et) `a && b`

`||` *OR* (ou) `a || b`

`!` *NOT* (ne pas) `!a`

```
int a, b;  
cin >> a >> b;  
cout << a && b << " " << a || b  
    << " " << !a << endl;
```

Exécution:

0

0

0 0 1

5.3 Les opérateurs logiques

`&&` *AND* (et) `a && b`

`||` *OR* (ou) `a || b`

`!` *NOT* (ne pas) `!a`

`a && b` vaut 1 si $a \neq 0$ **et** $b \neq 0$
 0 si $a = 0$ ou $b = 0$

`a || b` vaut 1 si $a \neq 0$ **ou** $b \neq 0$
 0 si $a = 0$ et $b = 0$

`!a` vaut 1 si $a = 0$
 0 si $a \neq 0$

5.3 Les opérateurs logiques

`&&` *AND* (et) `a && b`

`||` *OR* (ou) `a || b`

`!` *NOT* (ne pas) `!a`

```
int n, m;
cin >> n >> m;
if ( ( n == 0 ) && ( m == 0 ) )
    cout << "l'operation n/m n'est pas definie" << endl;
else if ( ( n == 0 ) || ( m == 0 ) )
    cout << "n/m + m/n donne infini" << endl;
if ( !( m == 0 ) )
    cout << n / m << endl;
```

5.3 Les opérateurs logiques

`&&` *AND* (et) `a && b`

`||` *OR* (ou) `a || b`

`!` *NOT* (ne pas) `!a`

```
int n, m;
cin >> n >> m;
if ( ( n == 0 ) && ( m == 0 ) )
    cout << "l'operation n/m n'est pas definie" << endl;
else if ( ( n == 0 ) || ( m == 0 ) )
    cout << "n/m + m/n donne infini" << endl;
if ( !( m == 0 ) )
    cout << n / m << endl;

0
0
l'operation n/m n'est pas definie
```

5.3 Les opérateurs logiques

`&&` *AND* (et) `a && b`

`||` *OR* (ou) `a || b`

`!` *NOT* (ne pas) `!a`

```
int n, m;
cin >> n >> m;
if ( ( n == 0 ) && ( m == 0 ) )
    cout << "l'operation n/m n'est pas definie" << endl;
else if ( ( n == 0 ) || ( m == 0 ) )
    cout << "n/m + m/n donne infini" << endl;
if ( !( m == 0 ) )
    cout << n / m << endl;
```

0

2

n/m + m/n donne infini

0

5.3 Les opérateurs logiques

`&&` *AND* (et) `a && b`

`||` *OR* (ou) `a || b`

`!` *NOT* (ne pas) `!a`

```
int n, m;  
cin >> n >> m;  
if ( ( n == 0 ) && ( m == 0 ) )  
    cout << "l'operation n/m n'est pas definie" << endl;  
else if ( ( n == 0 ) || ( m == 0 ) )  
    cout << "n/m + m/n donne infini" << endl;  
if ( !( m == 0 ) )  
    cout << n / m << endl;
```

2

2

1

5.3 Les opérateurs logiques

`&&` *AND* (et) `a && b`

`||` *OR* (ou) `a || b`

`!` *NOT* (ne pas) `!a`

```
int n, m;
cin >> n >> m;
if ( ( !n ) && ( !m ) )
    cout << "l'operation n/m n'est pas definie" << endl;
else if ( ( !n ) || ( !m ) )
    cout << "n/m + m/n donne infini" << endl;
if ( m != 0 )
    cout << n / m << endl;
```

2

2

1

5.3 Les opérateurs logiques

`&&` *AND* (et) `a && b`

`||` *OR* (ou) `a || b`

`!` *NOT* (ne pas) `!a`

```
int n, m;
cin >> n >> m;
if ( !( n || m ) )
    cout << "l'operation n/m n'est pas definie" << endl;
else if ( !( n && m ) )
    cout << "n/m + m/n donne infini" << endl;
if ( m )
    cout << n / m << endl;
```

2

2

1

5.3 Les opérateurs logiques

`&&` *AND* (et) `a && b`

`||` *OR* (ou) `a || b`

`!` *NOT* (ne pas) `!a`

Combinaison de plusieurs opérations logiques:

```
int a, b;  
cin >> a >> b;  
if ( ( ( a == 3 ) && ( b == 2 ) ) ||  
      ( ( a == 2 ) && ( b == 3 ) ) )  
    cout << "bingo" << endl;  
else  
    cout << "non" << endl;
```


5.3 Les opérateurs logiques

`&&` *AND* (et) `a && b`

`||` *OR* (ou) `a || b`

`!` *NOT* (ne pas) `!a`

Combinaison de plusieurs opérations logiques:

```
int a, b;  
cin >> a >> b;  
if ( ( ( a == 3 ) && ( b == 2 ) ) ||  
      ( ( a == 2 ) && ( b == 3 ) ) )  
    cout << "bingo" << endl;  
else  
    cout << "non" << endl;  
  
2  
3  
bingo
```

5.3 Les opérateurs logiques

`&&` *AND* (et) `a && b`

`||` *OR* (ou) `a || b`

`!` *NOT* (ne pas) `!a`

Combinaison de plusieurs opérations logiques:

```
int a, b;  
cin >> a >> b;  
if ( ( ( a == 3 ) && ( b == 2 ) ) ||  
      ( ( a == 2 ) && ( b == 3 ) ) )  
    cout << "bingo" << endl;  
else  
    cout << "non" << endl;  
  
3  
2  
bingo
```

5.3 Les opérateurs logiques

`&&` *AND* (et) `a && b`

`||` *OR* (ou) `a || b`

`!` *NOT* (ne pas) `!a`

Combinaison de plusieurs opérations logiques:

```
int a, b;  
cin >> a >> b;  
if ( ( ( a == 3 ) && ( b == 2 ) ) ||  
      ( ( a == 2 ) && ( b == 3 ) ) )  
    cout << "bingo" << endl;  
else  
    cout << "non" << endl;  
  
2  
2  
non
```

5.4 Les blocs

Souvent on veut effectuer plusieurs opérations à la suite d'un branchement `if`

```
int n = 12, m;  
cout << "entrez un nombre pour une division:";  
cin >> m;  
if ( m == 0 )  
    cout << "attention: division par zero!" << endl;  
if ( m == 0 )  
    cout << "entrez un autre nombre:";  
if ( m == 0 )  
    cin >> m;  
if ( m != 0 )  
    cout << n / m << endl;
```

→ utilisation des accolades { ... }

5.4 Les blocs

```
int n = 12, m;  
cout << "entrez un nombre pour une division:";  
cin >> m;  
if ( m == 0 )  
{  
    cout << "attention: division par zero!" << endl;  
    cout << "entrez un autre nombre:";  
    cin >> m;  
    if ( m != 0 )  
        cout << "bien! " << n / m << endl;  
    else  
        cout << "division par zero! " << endl;  
}  
else  
    cout << n / m << endl;
```

→ tout ce qui est contenu dans les accolades { ... }
est effectué après `if` si `m` vaut zéro

5.4 Les blocs

```
int n = 12, m;  
cout << "entrez un nombre pour une division:";  
cin >> m;  
if ( m == 0 )  
{  
    cout << "attention: division par zero!" << endl;  
    cout << "entrez un autre nombre:";  
    cin >> m;  
    if ( m != 0 )  
        cout << "bien! " << n / m << endl;  
    else  
        cout << "division par zero! " << endl;  
}  
else  
    cout << n / m << endl;
```

Exécution:

entrez un nombre pour une division:

5.4 Les blocs

```
int n = 12, m;  
cout << "entrez un nombre pour une division:";  
cin >> m;  
if ( m == 0 )  
{  
    cout << "attention: division par zero!" << endl;  
    cout << "entrez un autre nombre:";  
    cin >> m;  
    if ( m != 0 )  
        cout << "bien! " << n / m << endl;  
    else  
        cout << "division par zero! " << endl;  
}  
else  
    cout << n / m << endl;
```

Exécution:

entrez un nombre pour une division: 3
4

5.4 Les blocs

```
int n = 12, m;  
cout << "entrez un nombre pour une division:";  
cin >> m;  
if ( m == 0 )  
{  
    cout << "attention: division par zero!" << endl;  
    cout << "entrez un autre nombre:";  
    cin >> m;  
    if ( m != 0 )  
        cout << "bien! " << n / m << endl;  
    else  
        cout << "division par zero! " << endl;  
}  
else  
    cout << n / m << endl;
```

Exécution:

```
entrez un nombre pour une division: 0  
attention: division par zero!  
entrez un autre nombre:
```


5.4 Les blocs

```
int n = 12, m;  
cout << "entrez un nombre pour une division:";  
cin >> m;  
if ( m == 0 )  
{  
    cout << "attention: division par zero!" << endl;  
    cout << "entrez un autre nombre:";  
    cin >> m;  
    if ( m != 0 )  
        cout << "bien! " << n / m << endl;  
    else  
        cout << "division par zero! " << endl;  
}  
else  
    cout << n / m << endl;
```

Exécution:

```
entrez un nombre pour une division: 0  
attention: division par zero!  
entrez un autre nombre: 2  
bien! 6
```

5.4 Les blocs

```
int n = 12, m;  
cout << "entrez un nombre pour une division:";  
cin >> m;  
if ( m == 0 )  
{  
    cout << "attention: division par zero!" << endl;  
    cout << "entrez un autre nombre:";  
    cin >> m;  
    if ( m != 0 )  
        cout << "bien! " << n / m << endl;  
    else  
        cout << "division par zero! " << endl;  
}  
else  
    cout << n / m << endl;
```

Exécution:

entrez un nombre pour une division:

5.4 Les blocs

```
int n = 12, m;  
cout << "entrez un nombre pour une division:";  
cin >> m;  
if ( m == 0 )  
{  
    cout << "attention: division par zero!" << endl;  
    cout << "entrez un autre nombre:";  
    cin >> m;  
    if ( m != 0 )  
        cout << "bien! " << n / m << endl;  
    else  
        cout << "division par zero! " << endl;  
}  
else  
    cout << n / m << endl;
```

Exécution:

```
entrez un nombre pour une division: 0  
attention: division par zero!  
entrez un autre nombre:
```

5.4 Les blocs

```
int n = 12, m;  
cout << "entrez un nombre pour une division:";  
cin >> m;  
if ( m == 0 )  
{  
    cout << "attention: division par zero!" << endl;  
    cout << "entrez un autre nombre:";  
    cin >> m;  
    if ( m != 0 )  
        cout << "bien! " << n / m << endl;  
    else  
        cout << "division par zero! " << endl;  
}  
else  
    cout << n / m << endl;
```

Exécution:

entrez un nombre pour une division: 0

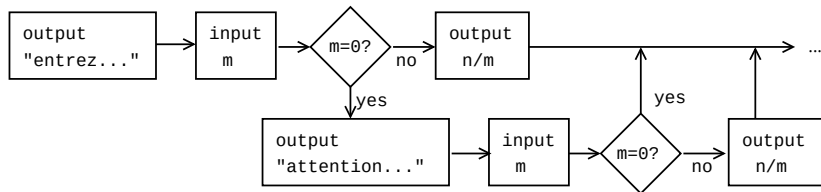
attention: division par zero!

entrez un autre nombre: 0

division par zero!

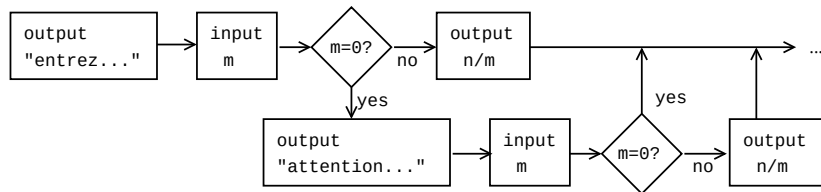
5.4 Les blocs

```
int n = 12, m;  
cout << "entrez un nombre pour une division:";  
cin >> m;  
if ( m == 0 )  
{  
    cout << "attention: division par zero!" << endl;  
    cout << "entrez un autre nombre:";  
    cin >> m;  
    if ( m != 0 )  
        cout << "bien! " << n / m << endl;  
    else  
        cout << "division par zero! " << endl;  
}  
else  
    cout << n / m << endl;
```



5.4 Les blocs

```
int n = 12, m;  
cout << "entrez un nombre pour une division:";  
cin >> m;  
if ( m != 0 )  
    cout << n / m << endl;  
else  
{  
    cout << "attention: division par zero!" << endl;  
    cout << "entrez un autre nombre:";  
    cin >> m;  
    if ( m != 0 )  
        cout << "bien! " << n / m << endl;  
    else  
        cout << "division par zero! " << endl;  
}
```



5.4 Les blocs

→ chaque instruction peut être remplacé
par un bloc d'instructions

```
if ( <expression> )  
    <instruction1>;  
else  
    <instruction2>;
```

5.4 Les blocs

→ chaque instruction peut être remplacé
par un bloc d'instructions

```
if ( <expression> )  
{  
    <instruction1a>;  
    <instruction1b>;  
    ...  
}  
else  
    <instruction2>;
```


5.4 Les blocs

→ chaque instruction peut être remplacé
par un bloc d'instructions

```
if ( <expression> )  
{  
    <instruction1a>;  
    <instruction1b>;  
    ...  
}  
else  
{  
    <instruction2a>;  
    <instruction2b>;  
    ...  
}
```

5.4 Les blocs

→ chaque instruction peut être remplacé
par un bloc d'instructions

```
if ( <expression> )  
{  
    <instruction1a>;  
    <instruction1b>;  
    {  
        <instruction1c1>;  
        <instruction1c2>;  
        ...  
    }  
    ...  
}  
else  
    <instruction2>;
```

5.4 Les blocs

→ chaque instruction peut être remplacé
par un bloc d'instructions

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

```
int main()  
{  
    <instruction1>;  
    <instruction2>;  
    ...  
}
```

5.4 Les blocs

→ chaque instruction peut être remplacé
par un bloc d'instructions

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

```
int main()  
{  
    <instruction1>;  
    <instruction2>;  
    {  
        <instruction3>;  
        <instruction4>;  
        ...  
    }  
    ...  
}
```

5.4 Les blocs

- chaque instruction peut être remplacé par un bloc d'instructions
- vous pouvez définir des nouvelles variables dans des blocs

```
int n = 12, m;  
cout << "entrez un nombre pour une division:";  
cin >> m;  
if ( m == 0 )  
{  
    cout << "attention: division par zero!" << endl;  
    cout << "entrez un autre nombre:";  
    int k;  
    cin >> k;  
    if ( k != 0 )  
        cout << "bien! " << n / k << endl;  
    else  
        cout << "division par zero! " << endl;  
}  
else  
    cout << n / m << endl;
```

5.4 Les blocs

- chaque instruction peut être remplacé par un bloc d'instructions
- vous pouvez définir des nouvelles variables dans des blocs ces dernières ne sont disponible que dans le cadre du bloc

```
int n = 12, m;  
cout << "entrez un nombre pour une division:";  
cin >> m;  
if ( m == 0 )  
{  
    cout << "attention: division par zero!" << endl;  
    cout << "entrez un autre nombre:";  
    int k;  
    cin >> k;  
    if ( k != 0 )  
        cout << "bien! " << n / k << endl;  
    else  
        cout << "division par zero! " << endl;  
}  
else  
    cout << n / m << endl;  
cout << k << endl;
```

- erreur: 'k' was not declared in this scope

5.4 Les blocs

- chaque instruction peut être remplacé par un bloc d'instructions
- vous pouvez définir des nouvelles variables dans des blocs ces dernières ne sont disponible que dans le cadre du bloc

```
int main()
{
    int n = 1;
    {
        int k;
        cin >> k;
        n += k;
    }
    int k = 2;
    cout << n << " " << k << endl;
}
```

5.4 Les blocs

- chaque instruction peut être remplacé par un bloc d'instructions
- vous pouvez définir des nouvelles variables dans des blocs ces dernières ne sont disponible que dans le cadre du bloc

```
int main()
{
    int n = 1;
    {
        int k;
        cin >> k;
        n += k;
    }
    int k = 2;
    cout << n << " " << k << endl;
}
5
6 2
```


5.5 Les boucles: while

→ effectuer des instructions d'une manière répétitive
... jusqu'à ce qu'une certaine condition soit satisfaite

```
int n = 12, m;  
cout << "entrez un nombre pour une division:";  
cin >> m;  
if ( m == 0 )  
{  
    cout << "attention: division par zero!" << endl;  
    cout << "entrez un autre nombre:";  
    cin >> m;  
    if ( m != 0 )  
        cout << "bien! " << n / m << endl;  
    else  
        cout << "division par zero! " << endl;  
}  
else  
    cout << n / m << endl;
```

5.5 Les boucles: while

→ effectuer des instructions d'une manière répétitive
... jusqu'à ce qu'une certaine condition soit satisfaite

```
int n = 12, m;  
cout << "entrez un nombre pour une division:";  
cin >> m;  
if ( m == 0 )  
{  
    cout << "attention: division par zero!" << endl;  
    cout << "entrez un autre nombre:";  
    cin >> m;  
    if ( m != 0 )  
        cout << "bien! " << n / m << endl;  
    else  
        cout << "division par zero! " << endl;  
}  
else  
    cout << n / m << endl;
```

entrez un nombre pour une division: 0

attention: division par zero!

entrez un autre nombre: 0

division par zero!

→ encore un (3ème, 4ème ...) essai ??

5.5 Les boucles: `while`

→ effectuer des instructions d'une manière répétitive
...jusqu'à ce qu'une certaine condition soit satisfaite

```
int n = 12, m;  
cout << "entrez un nombre pour une division:";  
cin >> m;  
while ( m == 0 )  
{  
    cout << "attention: division par zero!" << endl;  
    cout << "entrez un autre nombre:";  
    cin >> m;  
}  
cout << n / m << endl;
```

5.5 Les boucles: `while`

—→ effectuer des instructions d'une manière répétitive
... jusqu'à ce qu'une certaine condition soit satisfaite

```
int n = 12, m;  
cout << "entrez un nombre pour une division:";  
cin >> m;  
while ( m == 0 )  
{  
    cout << "attention: division par zero!" << endl;  
    cout << "entrez un autre nombre:";  
    cin >> m;  
}  
cout << n / m << endl;
```

entrez un nombre pour une division:

5.5 Les boucles: while

→ effectuer des instructions d'une manière répétitive
... jusqu'à ce qu'une certaine condition soit satisfaite

```
int n = 12, m;  
cout << "entrez un nombre pour une division:";  
cin >> m;  
while ( m == 0 )  
{  
    cout << "attention: division par zero!" << endl;  
    cout << "entrez un autre nombre:";  
    cin >> m;  
}  
cout << n / m << endl;
```

```
entrez un nombre pour une division: 0  
attention: division par zero!  
entrez un autre nombre:
```

5.5 Les boucles: while

→ effectuer des instructions d'une manière répétitive
... jusqu'à ce qu'une certaine condition soit satisfaite

```
int n = 12, m;  
cout << "entrez un nombre pour une division:";  
cin >> m;  
while ( m == 0 )  
{  
    cout << "attention: division par zero!" << endl;  
    cout << "entrez un autre nombre:";  
    cin >> m;  
}  
cout << n / m << endl;
```

```
entrez un nombre pour une division: 0  
attention: division par zero!  
entrez un autre nombre: 0  
attention: division par zero!  
entrez un autre nombre:
```

5.5 Les boucles: while

→ effectuer des instructions d'une manière répétitive
... jusqu'à ce qu'une certaine condition soit satisfaite

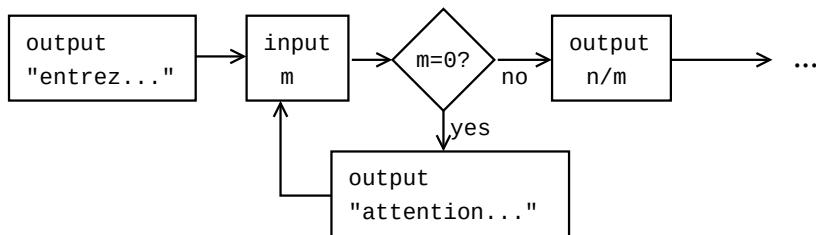
```
int n = 12, m;  
cout << "entrez un nombre pour une division:";  
cin >> m;  
while ( m == 0 )  
{  
    cout << "attention: division par zero!" << endl;  
    cout << "entrez un autre nombre:";  
    cin >> m;  
}  
cout << n / m << endl;
```

```
entrez un nombre pour une division: 0  
attention: division par zero!  
entrez un autre nombre: 0  
attention: division par zero!  
entrez un autre nombre: 2  
6
```

5.5 Les boucles: while

→ effectuer des instructions d'une manière répétitive
... jusqu'à ce qu'une certaine condition soit satisfaite

```
int n = 12, m;  
cout << "entrez un nombre pour une division:";  
cin >> m;  
while ( m == 0 )  
{  
    cout << "attention: division par zero!" << endl;  
    cout << "entrez un autre nombre:";  
    cin >> m;  
}  
cout << n / m << endl;
```



5.5 Les boucles: `while`

Structure générale:

```
while ( <expression> )  
    <instruction>;
```

→ `<instruction>` sera effectuée
tant que `<expression>` soit “vraie” (c-à-d ne vaille pas zéro)

5.5 Les boucles: `while`

Structure générale:

```
while ( <expression> )  
{  
    <instruction1>;  
    <instruction2>;  
    ...  
}
```

→ `<instruction1>`, `<instruction2>`, ... seront effectuées
tant que `<expression>` soit “vraie” (c-à-d ne vaille pas zéro)

5.6 Les itérations: `for`

Souvent on veut faire des boucles pour **itérer** une action:

```
int n;  
cout << "entrez un nombre:" << endl;  
cin >> n;  
int x = 0;  
int i = 1;  
while ( i <= n )  
{  
    x += i;  
    i++;  
}  
cout << "1 + 2 + ... + " << n << " = " << x << endl;
```

5.6 Les itérations: `for`

Souvent on veut faire des boucles pour **itérer** une action:

```
int n;  
cout << "entrez un nombre:" << endl;  
cin >> n;  
int x = 0;  
int i = 1;  
while ( i <= n )  
{  
    x = x + i;  
    i = i + 1;  
}  
cout << "1 + 2 + ... + " << n << " = " << x << endl;
```

5.6 Les itérations: `for`

Souvent on veut faire des boucles pour **itérer** une action:

```
int n;  
cout << "entrez un nombre:" << endl;  
cin >> n;  
int x = 0;  
int i = 1;  
while ( i <= n )  
{  
    x += i;  
    i++;  
}  
cout << "1 + 2 + ... + " << n << " = " << x << endl;
```

Exécution:

```
entrez un nombre: 6  
1 + 2 + ... + 6 = 21
```

5.6 Les itérations: `for`

Souvent on veut faire des boucles pour **itérer** une action:

```
int n;  
cout << "entrez un nombre:" << endl;  
cin >> n;  
int x = 1;  
int i = 1;  
while ( i <= n )  
{  
    x *= i;  
    i++;  
}  
cout << "1 * 2 * ... * " << n << " = " << x << endl;
```

Exécution:

```
entrez un nombre: 6  
1 * 2 * ... * 6 = 720
```

5.6 Les itérations: `for`

Souvent on veut faire des boucles pour **itérer** une action:

```
int n;  
cout << "entrez un nombre:" << endl;  
cin >> n;  
int x = 0;  
int i = 1;  
while ( i <= n )  
{  
    x += i;  
    i++;  
}  
cout << "1 + 2 + ... + " << n << " = " << x << endl;
```

→ utilisation de l'instruction `for`

5.6 Les itérations: `for`

Souvent on veut faire des boucles pour **itérer** une action:

```
int n;  
cout << "entrez un nombre:" << endl;  
cin >> n;  
int x = 0;  
int i;  
for ( i = 1; i <= n; i++ )  
    x += i;  
cout << "1 + 2 + ... + " << n << " = " << x << endl;
```


5.6 Les itérations: `for`

Souvent on veut faire des boucles pour **itérer** une action:

```
int n;  
cout << "entrez un nombre:" << endl;  
cin >> n;  
int x = 0;  
for ( int i = 1; i <= n; i++ )  
    x += i;  
cout << "1 + 2 + ... + " << n << " = " << x << endl;
```

5.6 Les itérations: `for`

Souvent on veut faire des boucles pour **itérer** une action:

```
int n;  
cout << "entrez un nombre:" << endl;  
cin >> n;  
int x = 1;  
for ( int i = 1; i <= n; i++ )  
    x *= i;  
cout << "1 * 2 * ... * " << n << " = " << x << endl;
```

5.6 Les itérations: `for`

```
for ( <instruction_start>; <expression>; <instruction_it> )  
{  
    <instruction1>;  
    <instruction2>;  
    ...  
}
```

5.6 Les itérations: `for`

```
for ( <instruction_start>; <expression>; <instruction_it> )  
{  
    <instruction1>;  
    <instruction2>;  
    ...  
}
```

...c'est équivalent à

```
{  
    <instruction_start>;  
    while ( <expression> )  
    {  
        <instruction1>; <instruction2>; ...  
        <instruction_it>;  
    }  
}
```

5.6 Les itérations: `for`

```
int n;  
cout << "entrez un nombre:" << endl;  
cin >> n;  
int x = 0;  
for ( int i = 1; i <= n; i++ )  
    x += i;  
cout << "1 + 2 + ... + " << n << " = " << x << endl;  
int i = -2;  
cout << i << endl;
```

Exécution:

entrez un nombre: 6

1 + 2 + ... + 6 = 21

-2

→ pas de conflit entre les deux déclarations `int i = ...;`

5.6 Les itérations: for

```
int n;  
cout << "entrez un nombre:" << endl;  
cin >> n;  
int x = 0;  
{  
    int i = 1;  
    while ( i <= n )  
    {  
        x += i;  
        i++;  
    }  
}  
cout << "1 + 2 + ... + " << n << " = " << x << endl;  
int i = -2;  
cout << i << endl;
```

... parce que `int i = 1;` est bien défini dans un bloc séparé

Des itérations multiples

```
int N = 5;
for ( int i = 0; i < N; i++ )
{
    for ( int j = 0; j < N; j++ )
    {
        cout << i + j;
        if ( i == j )
            cout << "|";
        else
            cout << " ";
    }
    cout << endl;
}
```

```
0|1 2 3 4
1 2|3 4 5
2 3 4|5 6
3 4 5 6|7
4 5 6 7 8|
```

Vos libertés en tant que programmeurs en C/C++

→ vous pouvez faire toutes combinaisons possibles
des boucles, des itérations, des branchements ...

```
if ( ... )  
{  
    for ( ... )  
        while ( ... )  
            {  
                ...  
            }  
}  
else  
{  
    while ( ... )  
        if ( ... )  
            {  
                ...  
            }  
}
```


Vos libertés en tant que programmeurs en C/C++

- vous pouvez faire toutes combinaisons possibles des boucles, des itérations, des branchements ...
- ... c'est votre responsabilité de vérifier que ça fonctionne bien ...