5 Les outils algorithmiques

Peter Schlagheck

Université de Liège

Avec tout ce que vous avez appris jusqu'à maintenant,
vous êtes bien capables d'utiliser le langage C/C++
comme une espèce de calculatrice avancée . . .

. . . mais vous n'avez pas encore exploré
toute la puissance de l'ordinateur,
s'exprimant d'abord par la possibilité d'effectuer
des branchements conditionnels et des boucles .

5 Les outils algorithmiques

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
```

$\longrightarrow$ 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

$\longrightarrow$ 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⟩`;`

$\longrightarrow$ ⟨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 ?

$\longrightarrow$ 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⟩`;`

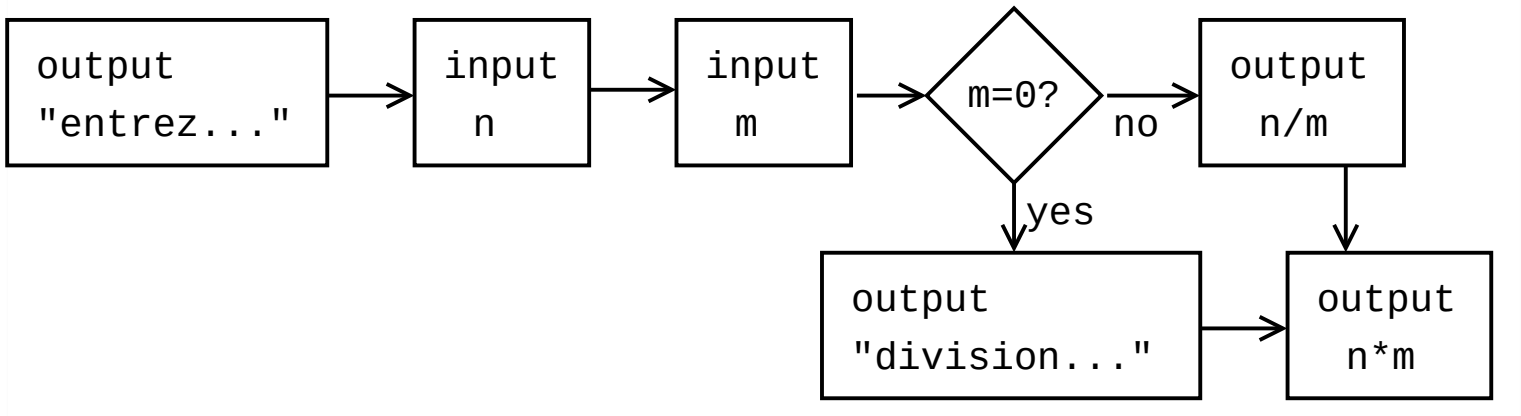
⟶ ⟨instruction⟩ sera effectuée seulement
si ⟨expression⟩ ne vaut pas zéro
⟶ dans l'autre cas, ⟨instruction2⟩ sera effectuée au lieu

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:



Le branchement conditionnel pour des expressions

```
int n, m;
cout << "entrez deux nombres entiers:";
cin >> n >> m;
if ( m )
  cout << n / m << endl;
else
  cout << 0 << endl;
```

Le branchement conditionnel pour des expressions

```
int n, m;
cout << "entrez deux nombres entiers:";
cin >> n >> m;
cout << ( m ? n / m : 0 ) << endl;
```

Le branchement conditionnel pour des expressions

```
int n, m;
cout << "entrez deux nombres entiers:";
cin >> n >> m;
cout << ( m ? n / m : 0 ) << endl;
```

Exécution:

```
entrez deux nombres entiers: 2
1
2
```

Le branchement conditionnel pour des expressions

```
int n, m;
cout << "entrez deux nombres entiers:";
cin >> n >> m;
cout << ( m ? n / m : 0 ) << endl;
```

Exécution:

```
entrez deux nombres entiers: 2
0
0
```

Le branchement conditionnel pour des expressions

```
int n, m;
cout << "entrez deux nombres entiers:";
cin >> n >> m;
cout << ( m ? n / m : 0 ) << endl;
```

Structure générale:

⟨comparaison⟩ `?` ⟨expression1⟩ `:` ⟨expression2⟩

$\longrightarrow$ ça donne ⟨expression1⟩ si ⟨comparaison⟩ ne vaut pas zéro et ⟨expression2⟩ si ⟨comparaison⟩ vaut zéro

$\longrightarrow$ ⟨expression1⟩ et ⟨expression2⟩ doivent être du même type

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 ?

$\longrightarrow$ 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;
```

$\longrightarrow$ 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`		`!=`	ne pas égal
`>`	plus grand:	`a > b`		`<=`	plus petit ou égal
`<`	plus petit:	`a < b`		`>=`	plus grand ou égal

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;
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

`==`	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;
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

`==`	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;
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

`==`	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;
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

`==`	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;
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

`==`	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;
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

`==`	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;
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

`==`	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;
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;
```

$\longrightarrow$ 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;
```

$\longrightarrow$ 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;
```

$\longrightarrow$ utilisation des accolades $\{ \ldots \}$

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;
```

$\longrightarrow$ 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

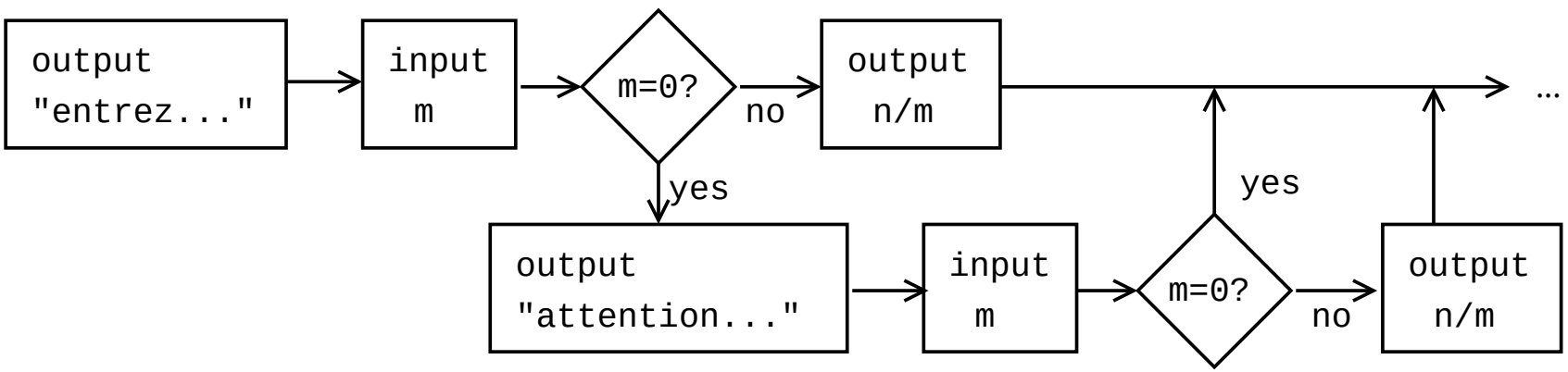
```
int n = 12, m;
cout << "entrez un nombre pour une division:";
cin >> m;
if ( m == 0 )
{
  cout << "attention: division par zero!" << endl;
  cout << "entrez un autre nombre:";
  cin >> m;
  if ( m != 0 )
    cout << "bien! " << n / m << endl;
  else
    cout << "division par zero! " << endl;
}
else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

Exécution:

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

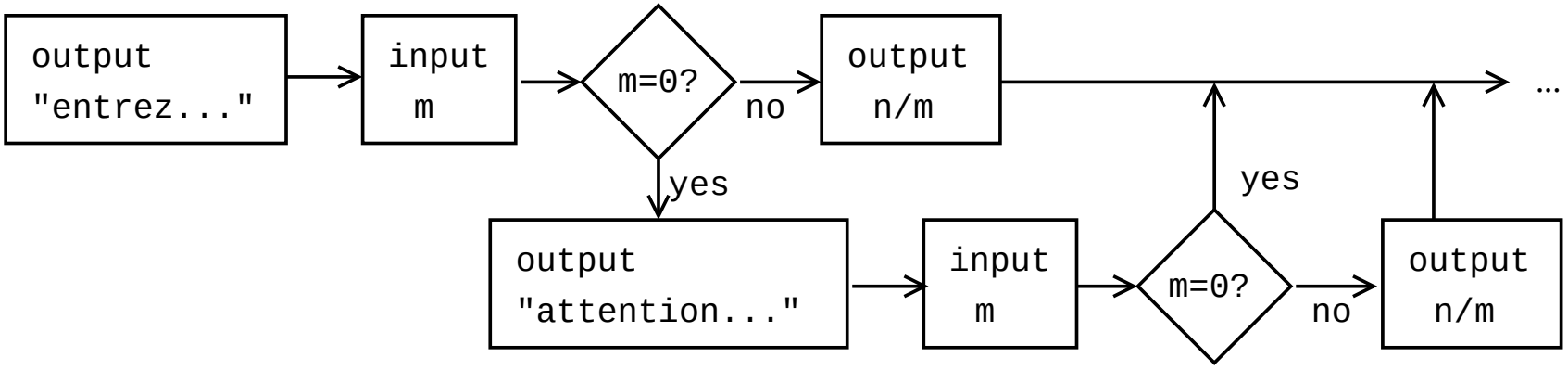
5.4 Les blocs

```
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

$\longrightarrow$ chaque instruction peut être remplacé par un bloc d'instructions

```
if ( ⟨expression⟩ )
   ⟨instruction1⟩;
else
   ⟨instruction2⟩;
```

5.4 Les blocs

$\longrightarrow$ chaque instruction peut être remplacé par un bloc d'instructions

```
if ( ⟨expression⟩ )
{
   ⟨instruction1a⟩;
   ⟨instruction1b⟩;
   ...
}
else
   ⟨instruction2⟩;
```

5.4 Les blocs

$\longrightarrow$ chaque instruction peut être remplacé par un bloc d'instructions

```
if ( ⟨expression⟩ )
{
   ⟨instruction1a⟩;
   ⟨instruction1b⟩;
   ...
}
else
{
   ⟨instruction2a⟩;
   ⟨instruction2b⟩;
   ...
}
```

5.4 Les blocs

$\longrightarrow$ chaque instruction peut être remplacé par un bloc d'instructions

```
if ( ⟨expression⟩ )
{
   ⟨instruction1a⟩;
   ⟨instruction1b⟩;
   {
      ⟨instruction1c1⟩;
      ⟨instruction1c2⟩;
      ...
   }
   ...
}
else
   ⟨instruction2⟩;
```

5.4 Les blocs

$\longrightarrow$ chaque instruction peut être remplacé par un bloc d'instructions

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

int main()
{
   ⟨instruction1⟩;
   ⟨instruction2⟩;
   ...
}
```

5.4 Les blocs

$\longrightarrow$ 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

$\longrightarrow$ chaque instruction peut être remplacé
par un bloc d'instructions

$\longrightarrow$ 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

$\longrightarrow$ chaque instruction peut être remplacé
par un bloc d'instructions

$\longrightarrow$ 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

$\longrightarrow$ chaque instruction peut être remplacé
par un bloc d'instructions

$\longrightarrow$ 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`

$\longrightarrow$ 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`

$\longrightarrow$ 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`

$\longrightarrow$ 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`

$\longrightarrow$ 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`

$\longrightarrow$ 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`

$\longrightarrow$ 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;
```

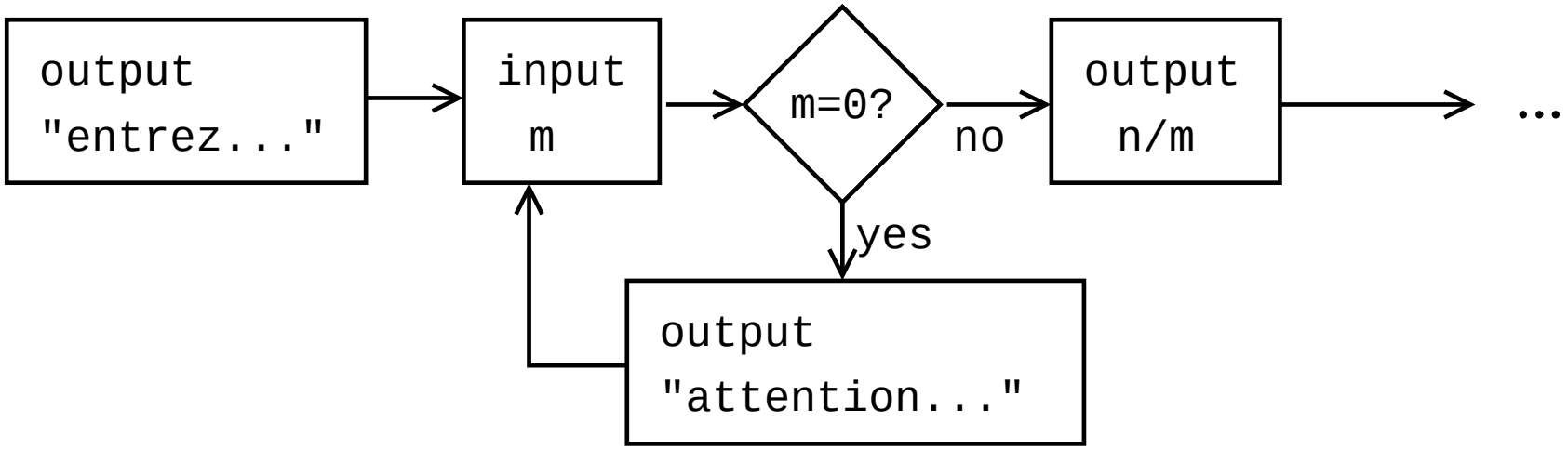
```
entrez un nombre pour une division: 0
attention: division par zero!
entrez un autre nombre: 0
attention: division par zero!
entrez un autre nombre: 2
6
```

5.5 Les boucles: `while`

$\longrightarrow$ 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⟩;
    ...
}
```

$\longrightarrow$ ⟨instruction1⟩, ⟨instruction2⟩, ... seront effectuées
tant que ⟨expression⟩ soit “vraie” (c-à-d ne vaille pas zéro)

5.5 Les boucles: `while`

Structure générale:

```
while ( ⟨expression⟩ )
    ⟨instruction⟩;
```

Variante:

```
do
    ⟨instruction⟩;
while ( ⟨expression⟩ );
```

$\longrightarrow$ ⟨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⟩;
    ...
}
```

Variante:

```
do
{
    ⟨instruction1⟩;
    ⟨instruction2⟩;
    ...
}
while ( ⟨expression⟩ );
```

$\longrightarrow$ ⟨instruction1⟩, ⟨instruction2⟩, ... seront effectuées
tant que ⟨expression⟩ soit “vraie” (c-à-d ne vaille pas zéro)

5.5 Les boucles: `while`

Structure générale:

`while` (⟨expression⟩)
    ⟨instruction⟩;

Variante:

`do`
    ⟨instruction⟩;
`while` (⟨expression⟩);

La différence entre `while` et `do`:

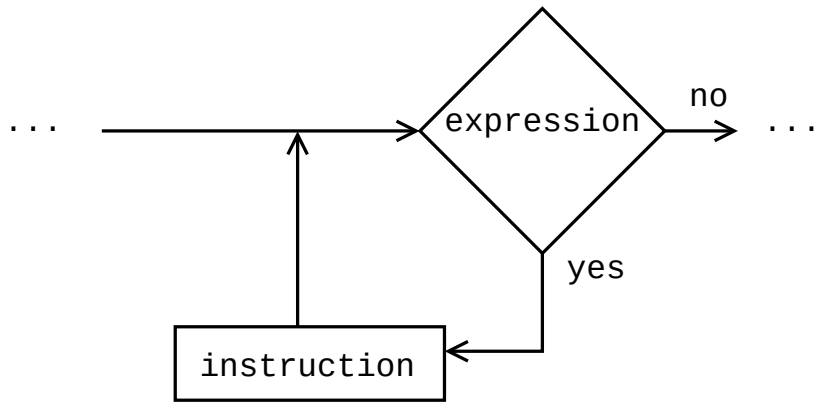
⟶ `while` verifie d'abord ⟨expression⟩
avant de décider s'il faut effectuer ⟨instruction⟩

⟶ `do` effectue ⟨instruction⟩ au moins une fois
avant de verifier ⟨expression⟩

5.5 Les boucles: `while`

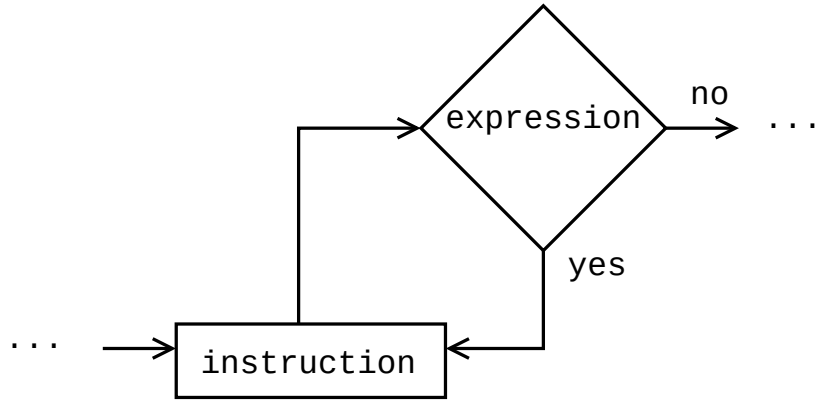
Structure générale:

```
while ( ⟨expression⟩ )
    ⟨instruction⟩;
```



Variante:

```
do
    ⟨instruction⟩;
while ( ⟨expression⟩ );
```



5.5 Les boucles: `while`

Structure générale:

`while` (⟨expression⟩)
    ⟨instruction⟩;

```
int x = 0;
cout << "2 + 3 = ";
cin >> x;
while ( x != 5 )
{
   cout << "faux!" << endl;
   cout << "2 + 3 = ";
   cin >> x;
}
cout << "bravo!" << endl;
```

Variante:

`do`
    ⟨instruction⟩;
`while` (⟨expression⟩);

```
int x = 0;
do
{
   cout << "2 + 3 = ";
   cin >> x;
   if ( x != 5 )
       cout << "faux!" << endl;
}
while ( x != 5 );
cout << "bravo!" << endl;
```

5.5 Les boucles: `while`

Structure générale:

`while` (⟨expression⟩)
    ⟨instruction⟩;

```
int x = 0;
cout << "2 + 3 = ";
cin >> x;
while ( x != 5 )
{
   cout << "faux!" << endl;
   cout << "2 + 3 = ";
}
cout << "bravo!" << endl;
```

Variante:

`do`
    ⟨instruction⟩;
`while` (⟨expression⟩);

```
int x = 0;
do
{
   cout << "2 + 3 = ";
   cin >> x;
   if ( x != 5 )
       cout << "faux!" << endl;
}
while ( x != 5 );
cout << "bravo!" << endl;
```

Attention !
Il est très facile d'écrire des boucles
qui ne se terminent jamais . . .

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;
```

$\longrightarrow$ 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⟩)
⟨instruction⟩;

...c'est équivalent à

```
{
   ⟨instruction_start⟩;
   while ( ⟨expression⟩ )
   {
      ⟨instruction⟩;
      ⟨instruction_it⟩;
   }
}
```

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
```

$\longrightarrow$ 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;
}
```

Exécution:

```
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|
```

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 << "|";
      break;
    }
    else
      cout << " ";
  }
  cout << endl;
}
```

Exécution:

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

⟶ `break` sort de la boucle actuelle (intérieure)

(ça marche également avec des boucles définies par `while`)

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

$\longrightarrow$ 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++

$\longrightarrow$ vous pouvez faire toutes combinaisons possibles
des boucles, des itérations, des branchements . . .

. . . c'est votre responsabilité de vérifier
que ça fonctionne bien . . .