

7

Boucle *for*

Les instructions en C/C++ (ou tout autre langage) peuvent être simples ou composées.

- Instructions simples
 - Affectation (`=`, `+=`, `-=`, ...) : changer la valeur d'une variable.
 - Lecture/écriture (`cin`, `cout`) : lire au clavier et afficher à l'écran.
- Instructions composées
 - Instructions conditionnelles (`if-else`) : faire un certain traitement ou l'ignorer.
 - Instructions répétitives (itératives) : répéter des traitements (instructions).

Dans cette section, nous allons aborder les instructions répétitives.

Exemple : Écrire un programme C++ permettant d'afficher le carré des 7 premiers entiers (0, 1, 2, ..., 6) de la manière suivante :

```
Le carré de 0 est 0
Le carré de 1 est 1
Le carré de 2 est 4
Le carré de 3 est 9
Le carré de 4 est 16
Le carré de 5 est 25
Le carré de 6 est 36
```

Solution

```
int main(){
    cout << "Le carre de " << 0 << " est " << 0 * 0 << endl;
    cout << "Le carre de " << 1 << " est " << 1 * 1 << endl;
    cout << "Le carre de " << 2 << " est " << 2 * 2 << endl;
    cout << "Le carre de " << 3 << " est " << 3 * 3 << endl;
    cout << "Le carre de " << 4 << " est " << 4 * 4 << endl;
    cout << "Le carre de " << 5 << " est " << 5 * 5 << endl;
    cout << "Le carre de " << 6 << " est " << 6 * 6 << endl;
}
```

Maintenant, comment vous allez faire, si on vous demande d'afficher le carré des 100 premiers entiers ou voire même le carré des 1000 premiers entiers?

```
cout << "Le carre de " << 0 << " est " << 0 * 0 << endl;
cout << "Le carre de " << 1 << " est " << 1 * 1 << endl;
cout << "Le carre de " << 2 << " est " << 2 * 2 << endl;
cout << "Le carre de " << 3 << " est " << 3 * 3 << endl;
cout << "Le carre de " << 4 << " est " << 4 * 4 << endl;
cout << "Le carre de " << 5 << " est " << 5 * 5 << endl;
cout << "Le carre de " << 6 << " est " << 6 * 6 << endl;
cout << "Le carre de " << 7 << " est " << 7 * 7 << endl;
cout << "Le carre de " << 8 << " est " << 8 * 8 << endl;
cout << "Le carre de " << 9 << " est " << 9 * 9 << endl;
.
.
```

Au lieu d'écrire toutes ces instructions, on peut utiliser une boucle *for*.

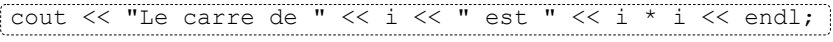
```
int main(){
    for (int i = 0; i < 7; i++){
        cout << "Le carre de " << i << " est " << i * i << endl;
    }
}
```

La boucle *for* permet de répéter des instructions un nombre donné de fois.

⇒ On appelle les instructions se trouvant entre accolades (i.e., les instructions à répéter) le *corps de la boucle*.

```
for (int i = 0; i < 7; i++){
    cout << "Le carre de " << i << " est " << i * i << endl;
}
```

Corps de la boucle



Et on appelle la variable *i* le *compteur de boucle*.

La boucle *for* est composée de trois parties : `for (int i = 0; i < 7; i++)`

- 1) `int i = 0` : déclaration et initialisation
- 2) `i < 7` : condition
- 3) `i++` : incrémentation.

Ces trois instructions sont séparées par des points-virgules et entourées de parenthèses.

On peut résumer le fonctionnement de la boucle For comme suit :

- `int i = 0;` (déclaration et initialisation) : n'est exécutée qu'une seule fois, avant d'entrer dans la boucle.
- `i < 7;` (condition) : testée avant l'exécution de chaque tour de boucle. Si la condition est fausse, on sort de la boucle. Si la condition est vraie, on exécute les instructions se trouvant entre accolades.
- `i++;` (incrément) : exécutée à la fin de chaque tour de boucle.

Exemple : Pour mieux expliquer les choses, exécutons la boucle *for* suivante pas à pas.

```
for (int i = 0; i < 3; i++){cout << "Le carre de " << i << " est " << i * i << endl;}
```

L'exécution de cette boucle *for* commence par l'instruction : `int i = 0;`

...
0
...

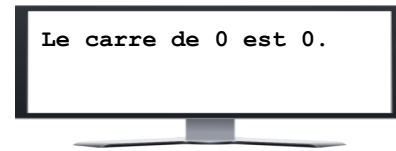
i

Cette instruction n'est exécutée *qu'une seule fois* au début i.e., avant d'entrer dans la boucle (c.-à-d., avant d'exécuter les instructions du corps de la boucle).

Après avoir déclaré et initialisé la variable *i*, on passe à l'instruction : `i < 3`; On vérifie la condition. Si elle est vraie, on exécute le corps de la boucle. Sinon, c'est-à-dire, la condition est fausse, on sort de la boucle For. Dans notre cas la variable *i* est initialisée à 0, ce qui veut dire que *i* est inférieure à 3. Condition vérifiée, donc, on entre et on exécute le corps de la boucle (avec `i == 0`) :

```
cout << "Le carre de " << i << " est " << i * i << endl;
```

Ce qui affiche à l'écran :



Après exécution de l'instruction `cout`, on exécute l'instruction : `i++`;

Rappel

L'instruction `i++` est équivalente à l'instruction `i = i + 1`.

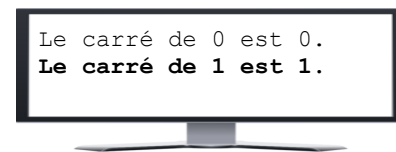
Elle permet de rajouter 1 à la valeur de la variable `i`.

i	...
	<u>0 1</u>
	...

Après avoir incrémenté la variable `i`, on refait le test (`i < 3`) avant d'entrer de nouveau dans le corps de la boucle.

On a `i == 1` alors la condition `i < 3` est vérifiée (vraie). Condition vérifiée, alors on ré-exécute le corps de la boucle (avec `i == 1`) :

```
cout << "Le carre de " << i << " est " << i * i << endl;
```

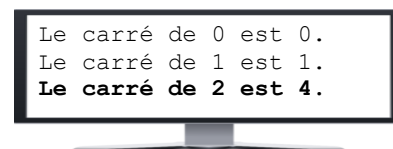


Après exécution de l'instruction `cout`, on exécute de nouveau l'instruction : `i++`;

i	...
	<u>1 2</u>
	...

Après incrémentation de la variable `i`, on refait le test (`i < 3`), toujours, avant d'entrer dans la boucle. On a `i == 2`, alors la condition `i < 3` est vérifiée. Condition vérifiée, alors on ré-exécute le corps de la boucle (avec `i == 2`) :

```
cout << "Le carre de " << i << " est " << i * i << endl;
```



Après exécution de l'instruction `cout`, on exécute de nouveau l'instruction : `i++`;

i	...
	<u>2 3</u>
	...

Après incrémentation de la variable `i`, on refait le test (`i < 3`) avant d'entrer dans la boucle. On a `i == 3`, alors la condition `i < 3` n'est pas vérifiée. Condition non vérifiée, alors on sort de la boucle "for".

Syntaxe de la boucle For

```
for (Déclaration_et_Initialisation; Condition; Incrémentation){  
    Instructions à répéter  
}
```

Exercice : Qu'affiche le programme C++ suivant?

```
int main(){  
    for (int i = 0; i < 10; i++){ cout << "Hello!" << endl; }  
}
```

Solution La variable `i` prend les valeurs : 0, 1, 2, 3, 4, 5, 6, 7, 8 et 9. En d'autres termes, l'instruction `cout << "Hello!" << endl;` sera exécutée 10 fois (pour `i = 0`, `i = 1`, ... , `i = 9`).

```
Hello!  
Hello!  
Hello!  
Hello!  
Hello!  
Hello!  
Hello!  
Hello!  
Hello!  
Hello!
```

Exercice: Qu'affiche le programme suivant ?

```
int main(){  
    for (int i = 0; i > 10; i++){ cout << "Hello!" << endl; }  
}
```

Solution Ce programme n'affiche rien. Si la condition n'est pas vraie dès le début alors la boucle ne sera pas exécutée.

Exercice: Qu'affiche le programme suivant?

```
int main(){  
    for (int i = 0; i < 10; i++){  
        cout << "Hello!" << endl;  
    }  
  
    cout << "i = " << i << endl;  
}
```

Solution Ce programme n'affiche rien. En fait, ce programme est incorrect (ne sera pas exécuté) parce qu'il contient une erreur de compilation!

La variable `i` disparaît après exécution de la boucle For. La variable `i` n'est déclarée (connue) qu'à l'intérieur de la boucle, i.e., elle ne peut pas être utilisée à l'extérieur de la boucle. L'espace mémoire réservée à cette variable est libérée après exécution de la boucle.

Exercice Écrire un programme C++ qui permet d'afficher la table de multiplication par 5.

```
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

Si on n'utilise pas de boucle for, on serait obligé de répéter 10 fois la même instruction comme suit :

```
int main(){
    cout << "5 x 1 = " << 5 * 1 << endl;
    cout << "5 x 2 = " << 5 * 2 << endl;
    cout << "5 x 3 = " << 5 * 3 << endl;
    cout << "5 x 4 = " << 5 * 4 << endl;
    cout << "5 x 5 = " << 5 * 5 << endl;
    cout << "5 x 6 = " << 5 * 6 << endl;
    cout << "5 x 7 = " << 5 * 7 << endl;
    cout << "5 x 8 = " << 5 * 8 << endl;
    cout << "5 x 9 = " << 5 * 9 << endl;
    cout << "5 x 10 = " << 5 * 10 << endl;
}
```

Pour éviter cette écriture (répétition), on peut utiliser une boucle for :

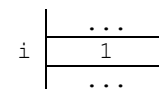
```
int main(){
    for (int i = 1; i <= 10; i++){
        cout << "5 x " << i << " = " << 5 * i << endl;
    }
}
```

La variable *i* prend, ici, les valeurs de 1 à 10.

Remarque : Le signe inférieur ou égal s'écrit en utilisant le signe '<' suivi du signe '=' et non pas de cette façon \leq .

Exécution pas à pas du programme précédent

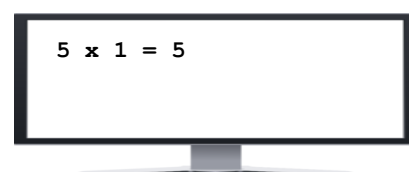
Au début, on déclare et initialise la variable *i*



Ensuite, on teste si *i* <= 10 (si vraie, on exécute l'instruction `cout`, sinon on sort de la boucle). On a *i* == 1, alors *i* <= 10. Condition vraie, alors exécuter le corps de la boucle (avec *i* == 1) :

```
cout << "5 x " << i << " = " << 5 * i << endl;
```

Ce qui affiche :

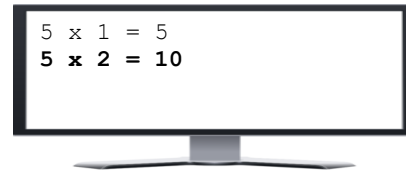


Exécuter l'instruction `i++`

i	...
	1 2
	...

Tester si `i <= 10`. On a `i == 2` alors `i <= 10`. Condition vraie, alors exécuter le corps de la boucle (avec `i == 2`): `cout << "5 x " << i << " = " << 5 * i << endl;`

Ce qui affiche :

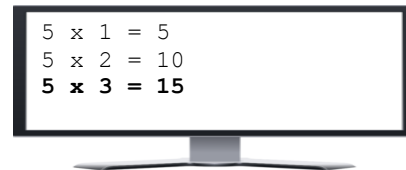


Exécuter l'instruction `i++`

i	...
	2 3
	...

Tester si `i <= 10`. On a `i == 3` alors `i <= 10`. Condition vraie, alors exécuter le corps de la boucle (avec `i == 3`): `cout << "5 x " << i << " = " << 5 * i << endl;`

Ce qui affiche :

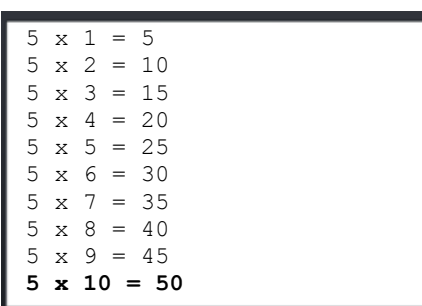


Et ainsi de suite, jusqu'à `i == 10`

i	...
	9 10
	...

Tester si `i <= 10`. On a `i == 10` alors `i <= 10`. Condition vraie, alors exécuter le corps de la boucle (avec `i == 10`).

Ce qui affiche :



Exécuter l'instruction `i++`

i	...
	10 11
	...

Tester si `i <= 10`. On a `i == 11` alors condition fausse; on sort de la boucle For (on n'exécute pas l'instruction `cout`).

Remarque : le bloc d'instructions répétées par une boucle `for` peut contenir n'importe quelles instructions (instructions simples ou composées). Par exemple, il peut s'agir de branchement conditionnel.

Exemple

```
int main(){
    for (int i = 0; i < 5; i++){
        cout << i;
        if (i % 2 == 0) {
            cout << "p";
        }
        cout << " ";
    }
    cout << endl;
}
```

Instruction simple

Instruction Composée

Instruction simple

La même règle s'applique à l'instruction `if`. Le bloc d'instructions exécutées par l'instruction `if` ou `else` peut contenir n'importe quelles instructions (instructions simples ou composées). Par exemple, il peut s'agir de branchement conditionnel ou de boucle `For`.

Exemple

```
int main(){
    int n;
    cout << "n = "; cin >> n;

    if (n > 0){
        for (int i = 0; i < n; i++){
            cout << i << endl;
        }
    }
}
```

Le programme ci-dessus est donné juste à titre d'exemple.

Exercice Qu'affiche le programme ci-dessous?

```
int main(){
    for (int i = 0; i < 5; i++){
        cout << i;
        if (i % 2 == 0) {
            cout << "p";
        }
        cout << " ";
    }
    cout << endl;
}
```

Choisissez la bonne réponse :

- a) 0 1 2 3 4
- b) 0p 1p 2p 3p 4p
- c) 0p 1 2p 3 4p

Solution : Les valeurs prises par la variable `i` sont : 0, 1, 2, 3 et 4. En d'autres termes, la boucle sera exécutée pour `i = 0`, `i = 1`, `i = 2`, `i = 3` et `i = 4`.

Pour `i = 0` : `cout << i;`



Passons à l'instruction suivante dans la boucle For : `if (i % 2 == 0) { cout << "p"; }`

Rappel : l'opérateur modulo (%) renvoie le reste de la division entière. Par exemple :

`11 % 4 = 3` (le reste de la division de 11 sur 4 est 3).
`12 % 4 = 0` (le reste de la division de 12 sur 4 est 0).

Revenons à notre exemple : `if (i % 2 == 0) { cout << "p"; }`. On a `i == 0`. Le reste de la division de 0 sur 2 est 0, alors on affiche "p".



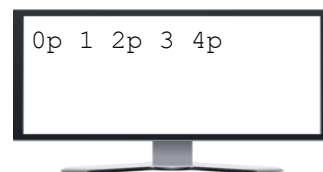
On passe à l'instruction suivante dans la boucle For, à savoir : `cout << " ";` ce qui affiche un espace après le "p".

On répète la même chose (les mêmes instructions) pour `i = 1`.



On commence par afficher 1. Le reste de la division de 1 sur 2 n'égale pas 0, alors on n'affiche pas "p". Ensuite, on affiche un espace après le 1.

On répète les mêmes instructions pour `i = 2, 3` et 4, on aura:



On remarque bien que le programme affiche la lettre "p" devant les nombres pairs.

Autres formes de la boucle For

1/ `for (int j = 0; j < 10; j += 3){ cout << j << endl; }`

Cette boucle sera exécutée pour `j = 0, 3, 6` et `9`.

Rappel : l'instruction `j += 3` est équivalente à l'instruction `j = j + 3`. On peut écrire :

`for (int j = 0; j < 10; j = j + 3){ cout << j << endl; }`

2/ `for (int k = 10; k > 0; k--){ cout << k << endl; }`

Cette boucle sera exécutée pour `k = 10, 9, 8, 7, 6, 5, 4, 3, 2` et `1`.

Rappel l'instruction `k--` est équivalente à l'instruction `k = k - 1`. Alors, on peut écrire :


```
for (int k = 10; k > 0; k = k - 1){ cout << k << endl; }
```

Maintenant, supposons qu'on s'est trompé sur la condition d'une boucle for :

```
for (int i = 1; i > 0; i++){ ... }
```

Ou, par exemple, on s'est trompé sur l'incrémentation :

```
int j = 0;
.
.
.
for (int i = 0; i < 10; j++){ ... }
```

Au lieu d'incrémenter *i*, on a incrémenté *j* (i.e., *i* va garder toujours la valeur 0). Que se passe-t-il dans ces cas? En fait, il s'agit de boucles infinies, et c'est ce que nous allons voir dans ce qui suit.

Boucles infinies

Dans une boucle for, que se passe-t-il si la condition ne devient jamais fausse? Par exemple :

```
for (int i = 0; i >=0; i++){ cout << i << endl; }
```

Cette boucle sera exécutée pour *i* = 0, 1, 2, 3, 4, 5, 6, ...

Si la condition est toujours vraie (ne devient jamais fausse), les instructions se trouvant dans la boucle seront répétées indéfiniment et la variable *i* prendra toutes les valeurs positives que le type '*int*' peut représenter.

Quelle est la plus grande valeur positive que le type '*int*' peut prendre (représenter)? Référez-vous à la section "*8. boucles infinies*" pour plus de détails concernant ce point.

Remarque : la variable compteur utilisée dans une boucle for peut être de n'importe quel type, pas forcément de type '*int*'.

Exemple

```
int main(){
    for (double i = 0; i < 10; i = i + 1.5){ cout << "i = " << i << endl; }
}
```

ce programme affiche:

```
i = 0
i = 1.5
i = 3
i = 4.5
i = 6
i = 7.5
i = 9
```

Utilisation des accolades

Exemple : Qu'affiche le programme suivant?

```
int main(){
    for (int i = 0; i < 10; i++){ cout << "Bonjour" << endl; }
}
```

Le programme affiche :

```
Bonjour
Bonjour
Bonjour
Bonjour
Bonjour
Bonjour
Bonjour
Bonjour
Bonjour
Bonjour
Bonjour
```

Remarque : comme pour l'instruction `if` ou `else`, les accolades ne sont obligatoires que si plusieurs instructions doivent être répétées . S'il n'y a qu'une seule instruction, on peut ne pas utiliser d'accolades.

```
int main(){
    for (int i = 0; i < 10; i++) cout << "Bonjour" << endl;
}
```

Mais toujours comme pour le `if` ou `else`, il est très conseillé de garder les accolades :

```
int main(){
    for (int i = 0; i < 5; i++){ cout << "i = " << i << endl; }
}
```

Exemple

```
int main(){
    for (int i = 0; i < 10; i++)
        cout << "i = " << i << endl;
        cout << "Bonjour!" << endl;
        cout << "Hello!" << endl;
}
```

Que s'affiche-t-il si on exécute le programme ci-dessus?

Dans le cas où les accolades sont omises, l'ordinateur ne considère que la première instruction (il ne répète que la première instruction). Cela étant dit, le programme précédent affiche :

```
i = 0
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
Bonjour!
Hello!
```

L'ordinateur n'a répété que la première instruction parce qu'on a pas mis d'accolades. Le programme précédent peut être vu de cette façon :

```
int main(){
    for (int i = 0; i < 10; i++){
        cout << "i = " << i << endl;
    }
    cout << "Bonjour!" << endl;
    cout << "Hello!" << endl;
}
```

Si on veut que l'ordinateur répète plusieurs instructions, on doit les entourer d'accolades. Supposons qu'on veuille que le programme précédent répète toutes les instructions, alors on doit les entourer d'accolades, comme suit :

```
int main(){
    for (int i = 0; i < 4; i++){
        cout << "i = " << i << endl;
        cout << "Bonjour!" << endl;
        cout << "Hello!" << endl;
    }
}
```

Ce qui affiche :

```
i = 0
Bonjour!
Hello!
i = 1
Bonjour!
Hello!
i = 2
Bonjour!
Hello!
i = 3
Bonjour!
Hello!
```

Exemple : Qu'affiche ce programme?

```
int main(){
    for (int i = 0; i < 4; i++){
        cout << "i = " << i << endl;
        cout << "Bonjour!" << endl;
    }
    cout << "Hello!" << endl;
}
```

Dans cet exemple, l'ordinateur ne répète que les deux premières instructions. Ce qui affiche :

```
i = 0
Bonjour!
i = 1
Bonjour!
i = 2
Bonjour!
i = 3
Bonjour!
Hello!
```

Exemple : Qu'affiche ce programme?

```
int main(){
    for (int i = 0; i < 5; i++)
        cout << "i = " << i << endl;
        cout << "Bonjour!" << endl;
}
```

Le programme affiche :

```
i = 0
i = 1
i = 2
i = 3
i = 4
Bonjour!
```

Exercice : Qu'affiche ce programme C++?

```
int main(){
    for (int i = 0; i < 10; i++);
        cout << "Bonjour!" << endl;
}
```

Ce programme :

- a) affiche 10 fois le chaîne "Bonjour!".
- b) affiche une seule fois le chaîne "Bonjour!".
- c) N'affiche rien.

La bonne réponse est (b); ce programme affiche une seule fois la chaîne "Bonjour!". Voyons pourquoi. Le programme précédent est vu par l'ordinateur de la manière suivante :

```
int main(){
    for (int i = 0; i < 10; i++)
        ;
    cout << "Bonjour!" << endl;
}
```

Le point-virgule ";" seul est considéré comme une instruction vide (une instruction qui ne fait rien). Ce qui veut dire que l'ordinateur va répéter cette instruction "vide", 10 fois. Après, il sort de la boucle for et affiche "Bonjour!". Donc, ne mettez pas de point-virgule à la fin d'une instruction for.

Notez que les exemples que nous venons de voir (concernant la présence ou l'absence des accolades) sont donnés juste à titre explicatif. En réalité, il est fortement conseillé de toujours utiliser des accolades (dans tous les cas, même, si on n'a qu'une seule instruction à répéter).

Remarque : Même si le compilateur nous autorise à modifier la variable compteur à l'intérieur d'une boucle for, il est toutefois très déconseillé de le faire. Par exemple :

```
int main(){
    for (int i = 0; i < 10; i++){
        .
        .
        i--; // très déconseillé
        .
        .
    }
}
```

Exercice : Écrire un programme qui demande quatre notes à l'utilisateur, puis affiche la moyenne de ces quatre notes.

Solution Faisons simple et écrivons tout d'abord une première solution sans boucle for en utilisant 5 variables : `note1`, `note2`, `note3`, `note4` et `somme`.

```
int main(){
    double note1, note2, note3, note4;

    cout << "Note 1: "; cin >> note1;
    cout << "Note 2: "; cin >> note2;
    cout << "Note 3: "; cin >> note3;
    cout << "Note 4: "; cin >> note4;

    double somme = note1 + note2 + note3 + note4;
    cout << "Moyenne = " << somme / 4 << endl;
}
```

Comme vous avez remarqué, l'instruction `cout` se répète :

```
cout << "Note 1: "; cin >> note1;
cout << "Note 2: "; cin >> note2;
cout << "Note 3: "; cin >> note3;
cout << "Note 4: "; cin >> note4;
```

On peut éviter cette répétition en utilisant une boucle for, comme suit :

```
int main(){
    double note1, note2, note3, note4;

    for(int i = 1; i <= 4; i++){
        cout << "Note" << i << ": "; cin >> ...?
    }
}
```

Le problème, que nous rencontrons, est comment écrire l'instruction `cin`. C'est-à-dire, quelle variable nous devons utiliser : `note1`, `note2`, `note3` ou `note4`? En fait, on ne peut utiliser qu'une seule variable à l'intérieur de la boucle. Pour vous faciliter les choses, réécrivons le programme précédent sans boucle for en n'utilisant que 2 variables : `note` et `somme`.

```
int main(){
    double note, somme = 0;

    cout << "Note 1: "; cin >> note;
    somme = somme + note;

    cout << "Note 2: "; cin >> note;
    somme = somme + note;

    cout << "Note 3: "; cin >> note;
    somme = somme + note;

    cout << "Note 4: "; cin >> note;
    somme = somme + note;

    cout << "Moyenne = " << somme / 4 << endl;
}
```

Remarquez bien les instructions qui se répètent. On peut remplacer ces instructions répétées par une boucle for :

```
int main(){
    double note, somme = 0;

    for (int i = 1; i <= 4; i++){
        cout << "Note " << i << ": "; cin >> note;
        somme = somme + note;
    }

    cout << "Moyenne = " << somme / 4 << endl;
}
```

Le programme que nous venons d'écrire n'est pas intéressant car si nous avons quatre modules (notes). Supposons maintenant qu'on veuille laisser choisir l'utilisateur le nombre de notes. Comment doit-on modifier le programme?

On remplace le 4 par le nombre de notes entré par l'utilisateur :

```
int main(){
    double note, somme = 0;

    int nombreNotes;
    cout << "Entrez le nombre de notes: "; cin >> nombreNotes;

    for (int i = 1; i <= nombreNotes; i++){
        cout << "Note " << i << ": "; cin >> note;
        somme = somme + note;
    }

    cout << "Moyenne = " << somme / nombreNotes << endl;
}
```

Que se passe-t-il si l'utilisateur entre 0 comme nombre de notes?

Erreur à l'exécution (Run-time Error): Division par 0. Cette erreur est causée par l'instruction :

```
cout << "Moyenne = " << somme / nombreNotes << endl;
```

Comment doit-on procéder pour régler ce problème?

```
int main(){
    double note, somme = 0;

    int nombreNotes;
    cout << "Entrez le nombre de notes: "; cin >> nombreNotes;

    for (int i = 1; i <= nombreNotes; i++){
        cout << "Note " << i << ": "; cin >> note;
        somme = somme + note;
    }

    if (nombreNotes > 0){
        cout << "Moyenne = " << somme / nombreNotes << endl;
    }
    else {
        cout << "Error: nombre de notes incorrect!";
    }
}
```

Boucles imbriquées

Reprenons l'exemple de la table de multiplication par 5.

```
int main(){
    for (int i = 1; i <= 10; i++){
        cout << "5 x " << i << " = " << 5 * i << endl;
    }
}
```

```
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

Supposons, maintenant, qu'on veuille afficher les tables de multiplication de 2 à 10.

```
int main(){
    for (int i = 1; i <= 10; i++){
        cout << "2 x " << i << " = " << 2 * i << endl;
    }

    for (int i = 1; i <= 10; i++){
        cout << "3 x " << i << " = " << 3 * i << endl;
    }

    for (int i = 1; i <= 10; i++){
        cout << "4 x " << i << " = " << 4 * i << endl;
    }
    .
    .
    .
}
```

Au lieu de répéter toutes ces boucles for, il suffit de mettre la boucle for précédente dans une autre boucle for, comme suit :

```
int main(){
    for (int j = 2; j <= 10; j++){
        for (int i = 1; i <= 10; i++){
            cout << "2 x " << i << " = " << 2 * i << endl;
        }

        cout << endl;
    }
}
```

Comme vous avez déjà remarqué, le programme ci-dessus, affiche 9 fois la table de multiplication par 2. Il faut le modifier de la manière suivante :

```
int main(){
    for (int j = 2; j <= 10; j++){
        for (int i = 1; i <= 10; i++){
            cout << j << " x " << i << " = " << j * i << endl;
        }

        cout << endl;
    }
}
```

Exercice : Que s'affiche-t-il quand on exécute le code suivant?

```
int main(){
    for (int i = 0; i < 3; i++){
        for (int j = 0; j < 4; j++){
            if (i == j){ cout << 0; }
            else { cout << 1; }
        }
        cout << endl;
    }
}
```

A:	B:	C:	D:
0 1 1 1	0 0 0 0	0 1 1 0	0 1 1 1
0 1 1 1	0 0 0 0	0 1 1 0	1 0 1 1
0 1 1 1	0 0 0 0	0 1 1 0	1 1 0 1

Exercice : Qu'affiche le programme suivant?

```
int main(){
    for (int i = 0; i < 3; i++){
        for (int j = 0; j < i; j++){
            cout << i + j;
        }
        cout << endl;
    }
}
```

A:	B:	C: Rien	D:
	0		0 1 2 3
1	0 1		0 1 2 3
2 3	0 1 2		0 1 2 3

Exercice écrire un programme C++ qui :

- lit deux nombres entiers positifs x et y .
- calcule et affiche le produit de ces deux nombres sans utiliser l'opérateur de multiplication '*'.

Astuce : le programme calcule la somme : $(x + x + \dots + x)$.

y fois

Par exemple, si $x = 3$ et $y = 4$ alors $x * y = 3 + 3 + 3 + 3$

4 fois

```
int main(){
    int x, y;
    cout << "x = "; cin >> x;
    cout << "y = "; cin >> y;

    int p = 0;
    for (int i = 1; i <= y; i++){
        p = p + x;
    }

    cout << x << " x " << y << " = " << p << endl;
}
```


Exercice écrire un programme C++ qui :

1. lit deux nombres entiers m et n .
2. Affiche la somme des entiers compris entre m et n (on supposera que $m < n$).

Exemple : si $m = 5$ et $n = 8$, alors la somme = $5 + 6 + 7 + 8 = 26$.

```
int main(){
    int m, n;
    cout << "m = "; cin >> m;
    cout << "n = "; cin >> n;

    int s = 0;
    for (int i = m; i <= n; i++){
        s = s + i;
    }

    cout << "Somme des entiers entre " << m << " et " << n << " = " << s << endl;
}
```