Les boucles conditionnelles

Il y a 3 structures de contrôle:

- les branchements conditionnels,
- les itérations, et
- les **boucles conditionnelles**.

Les itérations, ou boucles **`for`**, permettent de répéter une partie du programme.

Elles sont utilisées quand le nombre de répétitions est connu *avant* d'entrer dans la boucle.

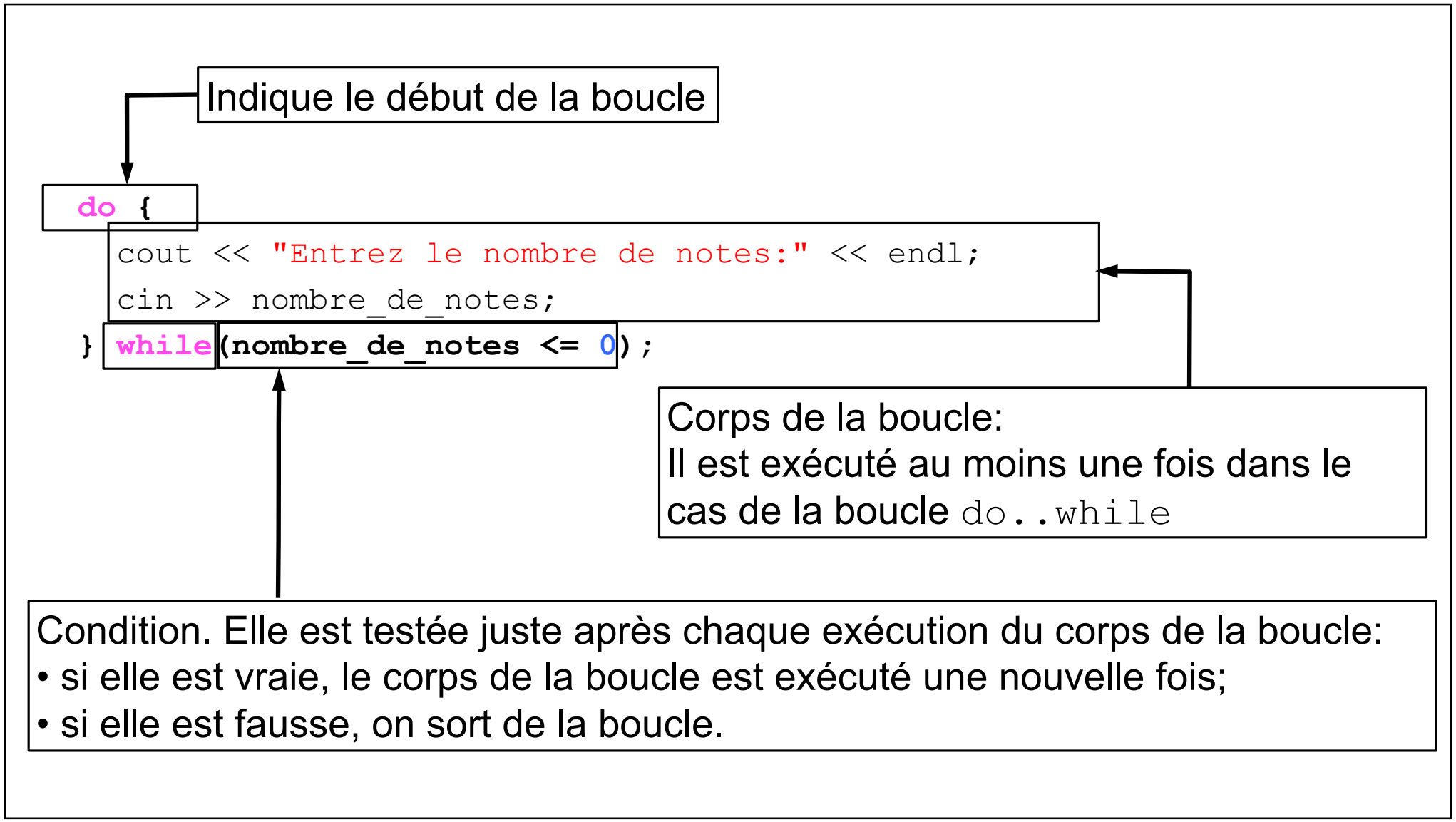
Selon le problème à résoudre, il arrive qu'on ne connaîsse pas combien de fois la boucle devra être exécutée.

On utilise alors une boucle conditionnelle, ou boucle **`while`**.

```
double note, somme(0.0);
int nombre_de_notes;

cout << "Entrez le nombre de notes:" << endl;
cin >> nombre_de_notes;

if (nombre_de_notes > 0) {
  for(int i(1); i <= nombre_de_notes; ++i) {
    cout << "Entrez la note numero " << i << endl;
    cin >> note;
    somme = somme + note;
  }

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

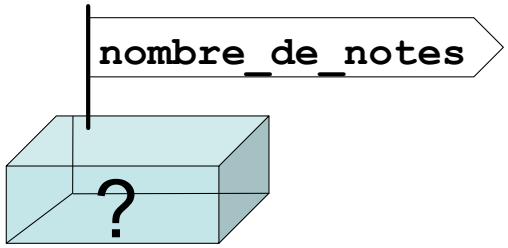
```
double note, somme(0.0);
int nombre_de_notes;

cout << "Entrez le nombre de notes:" << endl;
cin >> nombre_de_notes;

for(int i(1); i <= nombre_de_notes; ++i) {
  cout << "Entrez la note numero " << i << endl;
  cin >> note;
  somme = somme + note;
}

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

Comment forcer l'utilisateur à entrer une note supérieure à 0 ?


Indique le début de la boucle
do {
cout << "Entrez le nombre de notes:" << endl;
cin >> nombre_de_notes;
} while(nombre_de_notes <= 0);
Corps de la boucle:
Il est exécuté au moins une fois dans le cas de la boucle do..while
Condition. Elle est testée juste après chaque exécution du corps de la boucle:
• si elle est vraie, le corps de la boucle est exécuté une nouvelle fois;
• si elle est fausse, on sort de la boucle.

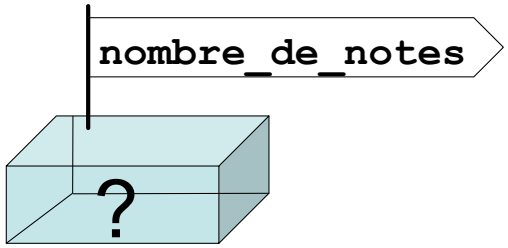
Pas-à-pas

```
do {
   cout << "Entrez le nombre de notes:" << endl;
   cin >> nombre_de_notes;
} while(nombre_de_notes <= 0);
```



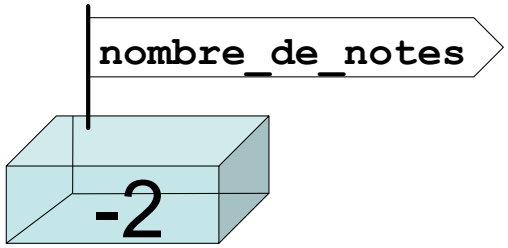
Pas-à-pas

```
do {
cout << "Entrez le nombre de notes:" << endl;
  cin >> nombre_de_notes;
} while(nombre_de_notes <= 0);
```



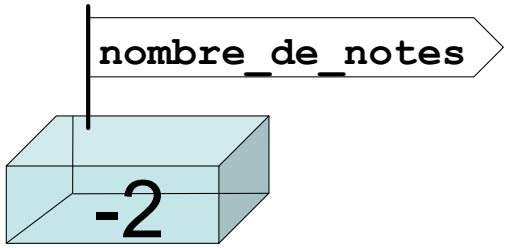
Pas-à-pas

```
do {
  cout << "Entrez le nombre de notes:" << endl;
  cin >> nombre_de_notes;
} while(nombre_de_notes <= 0);
```



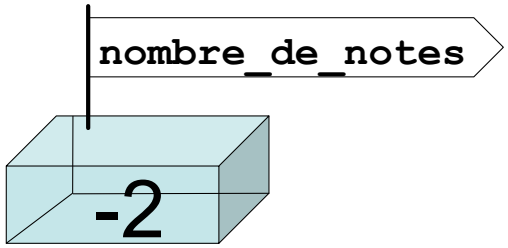
Pas-à-pas

```
do {
  cout << "Entrez le nombre de notes:" << endl;
  cin >> nombre_de_notes;
} while(nombre_de_notes <= 0);
```



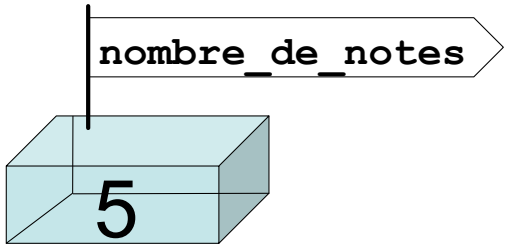
Pas-à-pas

```
do {
cout << "Entrez le nombre de notes:" << endl;
  cin >> nombre_de_notes;
} while(nombre_de_notes <= 0);
```



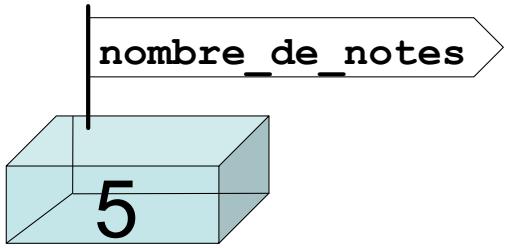
Pas-à-pas

```
do {
  cout << "Entrez le nombre de notes:" << endl;
  cin >> nombre_de_notes;
} while(nombre_de_notes <= 0);
```



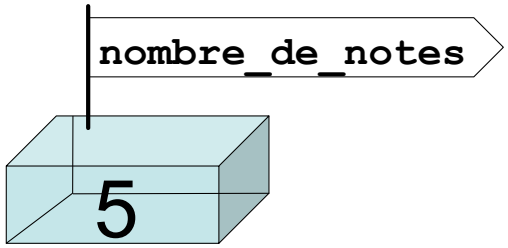
Pas-à-pas

```
do {
   cout << "Entrez le nombre de notes:" << endl;
   cin >> nombre_de_notes;
} while(nombre_de_notes <= 0);
```



Pas-à-pas

```
do {
  cout << "Entrez le nombre de notes:" << endl;
  cin >> nombre_de_notes;
} while(nombre_de_notes <= 0);
```



Syntaxe de l'instruction `do...while`

```
do {
  bloc
} while (condition);
```

- Comme pour l'instruction `if`:
  - La condition peut utiliser des opérateurs logiques.
  - Les parenthèses autour de la condition sont obligatoires.

- Les instructions à l'intérieur de la boucle `do…while` sont toujours exécutées au moins une fois.

- Si la condition ne devient jamais fausse, les instructions dans la boucle sont répétées indéfiniment !

L'instruction `while...`

Il existe également la forme suivante:

```
while (condition) {
 bloc
}
```

Le principe est similaire à celui de la boucle `do…while` que nous venons de voir.

La différence est que la condition est testée **avant** d'entrer dans la boucle. Si la condition est fausse, les instructions dans la boucle ne sont donc pas exécutées.

Exemple

```
int i(100);
do {
  cout << "bonjour" << endl;
} while (i < 10);
```

affichera une fois `bonjour`.

Dans les 2 cas,
la condition `i < 10` est fausse.

```
int i(100);
while (i < 10) {
  cout << "bonjour" << endl;
}
```

n'affichera rien.

Erreurs classiques

Il n'y a pas de `;` à la fin du `while...`:

```
while (i < 10); // !!
  ++i;
```

sera interprété comme

```
while(i < 10)
  ;
++i;
```

Le point-virgule est considéré comme le corps de la boucle, et l'instruction `++i` est **après la boucle**.

Si `i` est inférieur à 10, on entre dans la boucle pour ne jamais en ressortir puisque la valeur de `i` ne sera jamais modifiée.

En revanche, il y a un point-virgule à la fin du `do..while`:

```
do {
  ++i;
} while(i < 10);
```

Quand utiliser la boucle `while` ?
Quand utiliser la boucle `for` ?

Quand le nombre d'itérations (de répétitions) est connu avant d'entrer dans la boucle, utiliser `for`:

```
for(int i(0); i < nombre_d_iterations; ++i) {
```

Sinon, utiliser `while`:

– quand les instructions doivent être effectuées au moins une fois, utiliser `do…while`:

```
do {
  instructions;
} while (condition);
```

– Sinon, utiliser la forme `while…`

```
while (condition) {
   instructions;
}
```

```
do {
  cout << "Entrez le nombre de notes:" << endl;
  cin >> nombre_de_notes;
} while(nombre_de_notes <= 0);
```

```
Entrez le nombre de notes:
-2
il faut entrer un nombre supérieur à 0
Entrez le nombre de notes:
5
```

```
do {
  cout << "Entrez le nombre de notes:" << endl;
  cin >> nombre_de_notes;
  if (nombre_de_notes <= 0) {
    cout << "il faut entrer un nombre supérieur a 0" << endl;
  }
} while(nombre_de_notes <= 0);
```

```
Entrez le nombre de notes:
-2
il faut entrer un nombre superieur a 0
Entrez le nombre de notes:
5
```

Comment trouver la condition ?

On veut répéter la boucle **tant que** le nombre de notes est incorrect,

le nombre de notes est incorrect si il est inférieur ou égal à 0,

ce qui donne la condition précédente:

```
while (nombre_de_notes <= 0);
```

Comment trouver la condition ?

Supposons maintenant qu'on veuille limiter le nombre de notes à 10.

On veut toujours qu'il soit supérieur à 0.

Comment modifier la nouvelle condition ?

On veut répéter la boucle **tant que** le nombre de notes est incorrect,

le nombre de notes est incorrect si il est inférieur ou égal à 0 **ou** si il est supérieur à 10,

ce qui donne la nouvelle condition:

```
while (nombre_de_notes <= 0 or nombre_de_notes > 10);
```

Supposons qu'on veuille écrire un programme qui demande à l'utilisateur de deviner un nombre. Pour simplifier, nous supposerons que le nombre à deviner est toujours 5.

Le programme peut s'écrire ainsi:

```
int nombre_a_deviner(5);
int nombre_entre;

do {
  cout << "Entrez un nombre entre 1 et 10" << endl;
  cin >> nombre_entre;
} while( condition ? );

cout << "Trouve" << endl;
```

la boucle doit être répétée
tant que l'utilisateur n'a pas trouvé le nombre à deviner, c'est-à-dire
tant que `nombre_entre` est différent de `nombre_a_deviner`,
la condition est donc:

```
    cin >> nombre_entre;
} while(nombre_entre != nombre_a_deviner);

cout << "Trouve" << endl;
```

Supposons qu'on veuille en plus limiter le nombre d'essais à 3.
On peut ajouter une variable qui va compter le nombre d'essais utilisés:

```
int nombre_a_deviner(5);
int nombre_entre;
int nombre_essais(0);

do {
  cout << "Entrez un nombre entre 1 et 10" << endl;
  cin >> nombre_entre;
  ++nombre_essais;
} while( condition ? );
```

Comment modifier la condition pour que la boucle s'arrête quand le nombre d'essais dépasse 3 ?

la boucle doit être répétée
tant que l'utilisateur n'a pas trouvé le nombre à deviner **et** qu'il reste des essais,
c'est-à-dire
tant que `nombre_entre` est différent de `nombre_a_deviner` **et** que `nombre_essais` est inférieur à 3,
la condition est donc:

```
    } while(nombre_entre != nombre_a_deviner and nombre_essais < 3);
```

```
int nombre_a_deviner(5);
int nombre_entre;
int nombre_essais(0);

do {
  cout << "Entrez un nombre entre 1 et 10" << endl;
  cin >> nombre_entre;
  ++nombre_essais;
} while(nombre_entre != nombre_a_deviner and nombre_essais < 3);
```

Si on veut afficher un message pour indiquer à l'utilisateur s'il a trouvé le nombre ou si il a épuisé ses essais, on peut ajouter après la boucle:

```
if (nombre_entre == nombre_a_deviner) {
  cout << "Trouve" << endl;
} else {
  cout << "Perdu. Le nombre etait " << nombre_a_deviner << endl;
}
```

```
int nombre_a_deviner(5);
int nombre_entre;
int nombre_essais(0);

do {
  cout << "Entrez un nombre entre 1 et 10" << endl;
  cin >> nombre_entre;
  ++nombre_essais;
} while(nombre_entre != nombre_a_deviner and nombre_essais < 3);
```

Attention, si on avait utilisé `nombre_essais < 3` comme condition:

```
if (nombre_essais < 3) {
  cout << "Trouve" << endl;
} else {
  cout << "Perdu. Le nombre etait " << nombre_a_deviner << endl;
}
```

le programme afficherait `"Perdu. ..."` quand l'utilisateur trouve au troisième essai.