Erreurs de débutants

Erreurs classiques

Le test d'égalité s'écrit ==, et pas =

```
if (a = 1) // !!!
```

est accepté par le compilateur mais
- ne teste pas si `a` vaut 1, et
- affecte la valeur 1 à `a`.

Utilisé avec `-Wall`, `g++` affiche le *warning* suivant:

```
warning: suggest parentheses around assignment used as truth value
```

ou si vous avez une installation en Français:

```
attention : parenthèses suggérées autour de l'affectation utilisée comme valeur de vérité
```

Erreurs classiques

```
if (a == 1); // !!!
  cout << "a vaut 1" << endl;
```

`a vaut 1` est toujours affiché quelle que soit la valeur de `a`!

Le point-virgule est considéré comme une instruction, qui ne fait rien.
Le code précédent est compris par le compilateur comme:

```
if (a == 1)
  ;
cout << "a vaut 1" << endl;
```

le `cout` est donc situé après le `if`.
Aucun *warning* n'est affiché.

Si on utilise des accolades même quand il n'y a qu'une instruction dans le bloc, et qu'on écrive le test de la façon suivante:

```
if (a == 1) {
  cout << "a vaut 1" << endl;
}
```

l'erreur précédente a beaucoup moins de chance d'arriver.

Erreurs classiques

Ne pas oublier les accolades, l'indentation ne suffit pas:

```
if (n < p)
  cout << "n est plus petit que p" << endl;
  max = p;
else
  cout << "n est plus grand ou egal a p" << endl;
```

génère à la compilation l'erreur:

`syntax error before "else"`

Voici une meilleure présentation du code précédent:

```
if (n < p)
  cout << "n est plus petit que p" << endl;

max = p;
else
  cout << "n est plus grand ou egal a p" << endl;
```

L'instruction `max = p;` est déjà en dehors du `if` !

```
cout << "Entrez le premier nombre:" << endl;
cin >> n;
cout << "Entrez le deuxieme nombre:" << endl;
cin >> p;

if ((n < p) and (2 * n >= p)) {
  cout << "1";
}

if ((n < p) or (2 * n >= p)) {
  cout << "2";
}

if (n < p) {
  if (2 * n >= p) {
    cout << "3";
  } else {
    cout << "4";
  }
}
cout << endl;
```

Rappel:

- Pour le ET (`and`):
les deux conditions doivent être vraies;
- Pour le OU (`or`):
au moins l'une des conditions doit être vraie.

Qu'affiche ce programme quand l'utilisateur entre **1 et 2** ?

A: `2`
B: `24`
C: `123`
D: `1234`

?

```
cout << "Entrez le premier nombre:" << endl;
cin >> n;
cout << "Entrez le deuxieme nombre:" << endl;
cin >> p;

if ((n < p) and (2 * n >= p)) {
  cout << "1";
}

if ((n < p) or (2 * n >= p)) {
  cout << "2";
}

if (n < p) {
  if (2 * n >= p) {
    cout << "3";
  } else {
    cout << "4";
  }
}
cout << endl;
```

Rappel:

- Pour le ET (`and`):
les deux conditions doivent être vraies;
- Pour le OU (`or`):
au moins l'une des conditions doit être vraie.

Qu'affiche ce programme quand l'utilisateur entre **1 et 3** ?

A: `2`
B: `24`
C: `123`
D: `1234`

?

```
cout << "Entrez le premier nombre:" << endl;
cin >> n;
cout << "Entrez le deuxieme nombre:" << endl;
cin >> p;

if ((n < p) and (2 * n >= p)) {
  cout << "1";
}

if ((n < p) or (2 * n >= p)) {
  cout << "2";
}

if (n < p) {
  if (2 * n >= p) {
    cout << "3";
  } else {
    cout << "4";
  }
}
cout << endl;
```

Rappel:

- Pour le ET (`and`):
les deux conditions doivent être vraies;
- Pour le OU (`or`):
au moins l'une des conditions doit être vraie.

Qu'affiche ce programme quand l'utilisateur entre **2 et 1** ?

A: `2`
B: `24`
C: `123`
D: `1234`

?

Le type booléen (`bool`)

Le type `bool`

Le type `bool` (pour *boolean*, ou booléen) est le type des **conditions**.

Il permet de déclarer des **variables contenant la valeur d'une condition**.

Une variable de type booléen est souvent appelée simplement un **booléen**.

Comme les conditions, un booléen ne peut prendre que **deux valeurs possibles**: `true` ou `false`

Exemple:

```
int a(1);
int b(2);
bool test1(a == b);
bool test2(a < b);
```


Le type `bool`

Le type `bool` (pour *boolean*, ou booléen) est le type des **conditions**.

Il permet de déclarer des **variables contenant la valeur d'une condition**.

Une variable de type booléen est souvent appelée simplement un **booléen**.

Comme les conditions, un booléen ne peut prendre que **deux valeurs possibles**: `true` ou `false`

Exemple:

```
int a(1);
int b(2);
bool test1(a == b);
bool test2(a < b);
```

Le type `bool`

Le type `bool` (pour *boolean*, ou booléen) est le type des **conditions**.

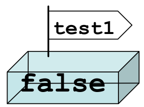
Il permet de déclarer des **variables contenant la valeur d'une condition**.

Une variable de type booléen est souvent appelée simplement un **booléen**.

Comme les conditions, un booléen ne peut prendre que **deux valeurs possibles**: `true` ou `false`

Exemple:

```
int a(1);
int b(2);
bool test1(a == b);
bool test2(a < b);
```

Le type `bool`

Le type `bool` (pour *boolean*, ou booléen) est le type des **conditions**.

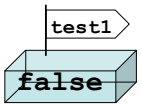
Il permet de déclarer des **variables contenant la valeur d'une condition**.

Une variable de type booléen est souvent appelée simplement un **booléen**.

Comme les conditions, un booléen ne peut prendre que **deux valeurs possibles**: `true` ou `false`

Exemple:

```
int a(1);
int b(2);
bool test1(a == b);
bool test2(a < b);
```



Le type `bool`

Le type `bool` (pour *boolean*, ou booléen) est le type des **conditions**.

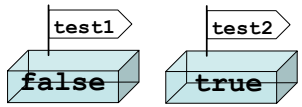
Il permet de déclarer des **variables contenant la valeur d'une condition**.

Une variable de type booléen est souvent appelée simplement un **booléen**.

Comme les conditions, un booléen ne peut prendre que **deux valeurs possibles**: `true` ou `false`

Exemple:

```
int a(1);
int b(2);
bool test1(a == b);
bool test2(a < b);
```



On peut initialiser des booléens à l'aide des constantes `false` et `true`.

On peut utiliser des booléens comme des conditions. Par exemple:
- on peut utiliser des opérateurs logiques (`and`, `or` et `not`) entre booléens;
- on peut utiliser un booléen comme condition dans un `if`.

```
int a(1);
int b(2);

bool c(true);
bool d(a == b);
bool e(d or (a < b));

if (e) {
  cout << "e vaut true" << endl;
}
```

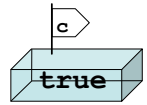
On peut initialiser des booléens à l'aide des constantes `false` et `true`.

On peut utiliser des booléens comme des conditions. Par exemple:
- on peut utiliser des opérateurs logiques (`and`, `or` et `not`) entre booléens;
- on peut utiliser un booléen comme condition dans un `if`.

```
int a(1);
int b(2);

bool c(true);
bool d(a == b);
bool e(d or (a < b));

if (e) {
  cout << "e vaut true" << endl;
}
```



On peut initialiser des booléens à l'aide des constantes `false` et `true`.

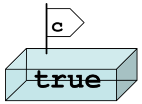
On peut utiliser des booléens comme des conditions. Par exemple:

- on peut utiliser des opérateurs logiques (`and`, `or` et `not`) entre booléens;
- on peut utiliser un booléen comme condition dans un `if`.

```
int a(1);
int b(2);

bool c(true);
bool d(a == b);
bool e(d or (a < b));

if (e) {
  cout << "e vaut true" << endl;
}
```



On peut initialiser des booléens à l'aide des constantes `false` et `true`.

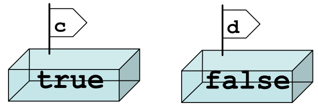
On peut utiliser des booléens comme des conditions. Par exemple:

- on peut utiliser des opérateurs logiques (`and`, `or` et `not`) entre booléens;
- on peut utiliser un booléen comme condition dans un `if`.

```
int a(1);
int b(2);

bool c(true);
bool d(a == b);
bool e(d or (a < b));

if (e) {
  cout << "e vaut true" << endl;
}
```



On peut initialiser des booléens à l'aide des constantes `false` et `true`.

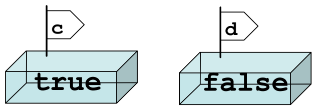
On peut utiliser des booléens comme des conditions. Par exemple:

- on peut utiliser des opérateurs logiques (`and`, `or` et `not`) entre booléens;
- on peut utiliser un booléen comme condition dans un `if`.

```
int a(1);
int b(2);

bool c(true);
bool d(a == b);
bool e(d or (a < b));

if (e) {
  cout << "e vaut true" << endl;
}
```



On peut initialiser des booléens à l'aide des constantes `false` et `true`.

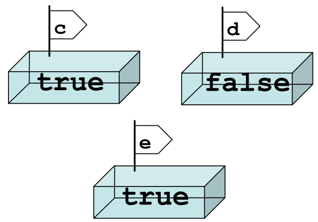
On peut utiliser des booléens comme des conditions. Par exemple:

- on peut utiliser des opérateurs logiques (`and`, `or` et `not`) entre booléens;
- on peut utiliser un booléen comme condition dans un `if`.

```
int a(1);
int b(2);

bool c(true);
bool d(a == b);
bool e(d or (a < b));

if (e) {
  cout << "e vaut true" << endl;
}
```



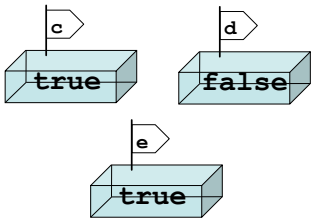
On peut initialiser des booléens à l'aide des constantes `false` et `true`.

On peut utiliser des booléens comme des conditions. Par exemple:
- on peut utiliser des opérateurs logiques (`and`, `or` et `not`) entre booléens;
- on peut utiliser un booléen comme condition dans un `if`.

```
int a(1);
int b(2);

bool c(true);
bool d(a == b);
bool e(d or (a < b));

if (e) {
  cout << "e vaut true" << endl;
}
```



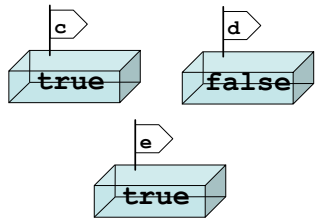
On peut initialiser des booléens à l'aide des constantes `false` et `true`.

On peut utiliser des booléens comme des conditions. Par exemple:
- on peut utiliser des opérateurs logiques (`and`, `or` et `not`) entre booléens;
- on peut utiliser un booléen comme condition dans un `if`.

```
int a(1);
int b(2);

bool c(true);
bool d(a == b);
bool e(d or (a < b));

if (e) {
  cout << "e vaut true" << endl;
}
```



Les booléens sont utiles pour de nombreux problèmes, nous rencontrerons des exemples concrets dans la suite du cours.