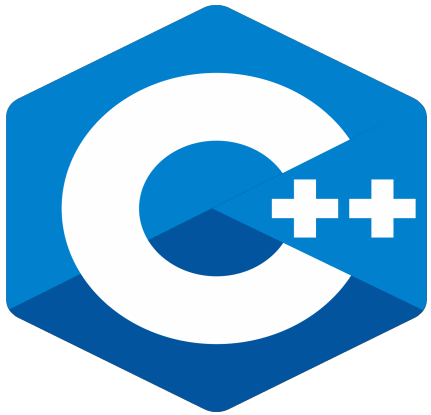
C++ : concepts de base

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com



Plan

Plan

Plan

C++

En C++

Deux types de commentaires

C++

Commentaire sur une seule ligne

```
// commentaire
```

C++

Commentaire sur une seule ligne

```
// commentaire
```


Raccourcis **VSC**

- Pour commenter : `Ctrl` + `k` ensuite `Ctrl` + `c`
- Pour décommenter : `Ctrl` + `k` ensuite `Ctrl` + `u`
- Ou aussi : `Ctrl` + `:` pour commenter et décommenter

C++

Commentaire sur plusieurs lignes

```
/* le commentaire
   la suite
   et encore la suite
*/
```

C++

Commentaire sur plusieurs lignes

```
/* le commentaire
   la suite
   et encore la suite
*/
```


Raccourci **VSC**

Pour commenter ou décommenter : `Alt` + `Shift` + `a`

C++

Variable ?

- Représentant d'un espace mémoire
- Permettant de stocker une ou plusieurs données
- Pouvant avoir plusieurs valeurs différentes dans un programme

C++

Variable ?

- Représentant d'un espace mémoire
- Permettant de stocker une ou plusieurs données
- Pouvant avoir plusieurs valeurs différentes dans un programme

C++ est un langage fortement typé

- Il faut préciser le type de chaque variable
- Une variable peut avoir de valeurs différentes mais ne peut changer de type.

C++

Déclarer une variable

```
type nomVariable;
```

C++

Principaux types de données numériques en **C++**

- `short int` (ou `short`) : entier sur deux octets dont la valeur est entre -32 768 et 32 767)
- `int` : entier dont la valeur est entre -2 147 483 648 et 2 147 483 647
- `long int` (ou `long`) : entier codé sur 8 octets
- `float` : nombre à virgule codé sur 4 octets
- `double` : nombre à virgule codé sur 8 octets
- ...

C++

Remarques

- À tous ces types, on peut ajouter le mot-clé `unsigned` pour avoir uniquement des valeurs positives.
- Le langage **C++** n'impose pas aux compilateurs un nombre exact d'octets pour chaque type mais impose un nombre minimal.
- Donc, le nombre d'octets réservé à un même type peut varier selon les compilateurs.

C++

Principaux types texte en **C++**

- `char` : caractère **UTF-8** entouré par deux `'` codé sur un octet
- `char16_t` [**C++11**] : caractère **UTF-16** situé entre deux `'` codé sur 2 ou 4 octets
- `char32_t` [**C++11**] : caractère **UTF-32** situé entre deux `'` codé sur 2 ou 4 octets
- `string` : une suite de caractères située entre deux `"`
- ...

C++

Principaux types texte en **C++**

- `char` : caractère **UTF-8** entouré par deux `'` codé sur un octet
- `char16_t` [**C++11**] : caractère **UTF-16** situé entre deux `'` codé sur 2 ou 4 octets
- `char32_t` [**C++11**] : caractère **UTF-32** situé entre deux `'` codé sur 2 ou 4 octets
- `string` : une suite de caractères située entre deux `"`
- ...

Type booléen en **C++**

`bool` : acceptant uniquement les valeurs `true` et `false`, codé sur 1 octet.

C++

Nom de variable en **C++**

- Peut contenir une lettre, un chiffre ou un _ (underscore)
- Doit commencer par une lettre ou un _ (underscore)
- Sensible à la casse
- Ne peut contenir le caractère espace ou les caractères spéciaux comme `#`, `!`...
- Ne peut être un mot réservé du langage comme `for`, `int`...

C++

Nom de variable en **C++**

- Peut contenir une lettre, un chiffre ou un _ (underscore)
- Doit commencer par une lettre ou un _ (underscore)
- Sensible à la casse
- Ne peut contenir le caractère espace ou les caractères spéciaux comme `#`, `!`...
- Ne peut être un mot réservé du langage comme `for`, `int`...

Remarque

Donner un nom significatif aux variables.

C++

Exemple

```
int x;
```

C++

Exemple

```
int x;
```

Déclaration + initialisation [copy initialization]

```
int x = 5;
```

C++

Exemple

```
int x;
```

Déclaration + initialisation [copy initialization]

```
int x = 5;
```

Ou aussi [direct initialization]

```
int y(3);
```

C++

Exemple

```
int x;
```

Déclaration + initialisation [copy initialization]

```
int x = 5;
```

Ou aussi [direct initialization]

```
int y(3);
```

Ou encore (valable depuis C++11) [value initialization]

```
int z{2};
```

C++

Il est possible de faire plusieurs déclarations sur la même ligne

```
int x = 5, y(3), z{2};
```

C++

Une variable non initialisé contient une valeur indéterminée

```
int x;
cout << x;
// affiche 6422352 (par exemple)
```

C++

Une variable non initialisé contient une valeur indéterminée

```
int x;
cout << x;
// affiche 6422352 (par exemple)
```

Attention ceci est considéré comme une fonction avec 1 comme valeur de retour

```
int y();
cout << y;
// affiche 1
```

C++

Une variable non initialisé contient une valeur indéterminée

```
int x;
cout << x;
// affiche 6422352 (par exemple)
```

Attention ceci est considéré comme une fonction avec 1 comme valeur de retour

```
int y();
cout << y;
// affiche 1
```

Une variable déclarée ainsi sans initialisation explicite est initialisée à 0

```
int z{};
cout << z;
// affiche 0
```

C++

Déclaration + initialisation d'un booléen

```
bool b = true;
```

C++

Déclaration + initialisation d'un booléen

```
bool b = true;
```

`true` ≡ **1**

```
cout << b << endl;
// affiche 1
```

C++

Déclaration + initialisation d'un booléen

```
bool b = true;
```

`true` ≡ **1**

```
cout << b << endl;
// affiche 1
```

Pour affiche `true` à la place de 1

```
cout << boolalpha << b << endl;
// affiche true
```

C++

`false` $\equiv$ **0**

```
b = false;
cout << b << endl;
// affiche 0
```

C++

`false` $\equiv$ **0**

```
b = false;
cout << b << endl;
// affiche 0
```

Pour affiche `false` à la place de 0

```
b = false;
cout << boolalpha << b << endl;
// affiche false
```

C++

Syntaxe (une variable typée dynamiquement doit être initialisée)

```
auto nomVariable = initialisation;
```

C++

Syntaxe (une variable typée dynamiquement doit être initialisée)

```
auto nomVariable = initialisation;
```

Exemple

```
auto str = "bonjour";
```


Une variable typée dynamiquement ne peut changer de type.

C++

Pour déclarer une variable ayant le même type qu'une autre, on utilise `decltype`

```
int x;
decltype(x) y;
```

C++

Pour déclarer une variable ayant le même type qu'une autre, on utilise `decltype`

```
int x;
decltype(x) y;
```

`x` et `y` sont toutes les deux de type `int`.

C++

Pour déclarer une variable ayant le même type qu'une autre, on utilise `decltype`

```
int x;
decltype(x) y;
```

`x` et `y` sont toutes les deux de type `int`.

On peut donc lui affecter un entier

```
y = 3;
```

C++

Pour déclarer une variable ayant le même type qu'une autre, on utilise `decltype`

```
int x;
decltype(x) y;
```

`x` et `y` sont toutes les deux de type `int`.

On peut donc lui affecter un entier

```
y = 3;
```

Main on ne peut lui affecter une valeur d'un autre type

```
y = "bonjour";
```

C++

Pour connaître le nombre d'octets réservés à une variable (qui varie selon le type))

```
int x = 5;
cout << sizeof(x);
// affiche 4
```

C++

Pour connaître le nombre d'octets réservés à une variable (qui varie selon le type))

```
int x = 5;
cout << sizeof(x);
// affiche 4
```

`sizeof` **fonctionne aussi avec les types**

```
cout << "Taille de char : " << sizeof(char) << endl;
// affiche Taille de char : 1

cout << "Taille de int : " << sizeof(int) << endl;
// affiche Taille de int : 4

cout << "Taille de float : " << sizeof(float) << endl;
// affiche Taille de float : 4

cout << "Taille de double : " << sizeof(double) << endl;
// affiche Taille de double : 8
```

C++

Remarque

Le type de la valeur de retour de `sizeof` est `size_t` : `unsigned int` codé sur 64 bits.

C++

Portée (scope) de variable

- locale : déclarée dans une fonction (`main` par exemple)
- globale : déclarée en dehors de toute fonction

C++

Dans ce programme, `l` est une variable locale et `g` est globale

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

int g = 10;

int main()
{
    int l = 2;
    return 0;
}
```

C++

Dans un programme, on peut déclarer une variable locale portant le nom d'une variable globale

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

int g = 10;

int main()
{
    int g = 2;
    cout << g;
    // affiche 2
    return 0;
}
```

C++

Pour accéder au contenu de la variable globale si on a une variable locale avec le même nom

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

int g = 10;

int main()
{
    int g = 2;
    cout << ::g;
    // affiche 10
    return 0;
}
```

C++

Remarques

- Une variable locale non-initialisée peut contenir soit 0, soit 1, soit une valeur aléatoire.
- Une variable globale non-initialisée contient
  - 0 si la variable est numérique (`int`, `float`, `double`...)
  - `NULL` si la variable est un pointeur
  - caractère vide pour `char` et `string`

C++

C++ nous autorise à créer des alias sur les types

```
typedef string str;
str s = "bonjour";
cout << s << endl;
// affiche bonjour
```

C++

Opérations d'entrée/sortie

- Pour afficher dans la console, on utilise
  - `cout` : sortie standard
  - `cerr` : sortie d'erreur
  - `clog` : sortie technique (journalisation)
- Pour lire une donnée saisie dans la console, on utilise `cin`.

C++

Pour afficher le contenu d'une variable

```
int x = 5;
cout << x;
// affiche 5
```

C++

Pour afficher le contenu d'une variable

```
int x = 5;
cout << x;
// affiche 5
```

Pour afficher le contenu de plusieurs variables

```
int x = 5, y(3);
cout << x << y;
// affiche 53
```

C++

Pour afficher le contenu d'une variable

```
int x = 5;
cout << x;
// affiche 5
```

Pour afficher le contenu de plusieurs variables

```
int x = 5, y(3);
cout << x << y;
// affiche 53
```

On peut aussi ajouter un espace pour séparer les variables

```
int x = 5, y(3);
cout << x << " " << y;
// affiche 5 3
```

Pour ajouter un retour à la ligne

```
int x = 5, y(3);
cout << x << endl;
cout << y << endl;
/* affiche
5
3
*/
```

Pour ajouter un retour à la ligne

```
int x = 5, y(3);
cout << x << endl;
cout << y << endl;
/* affiche
5
3
*/
```

Ou en plus simple

```
int x = 5, y(3);
cout << x << endl << y << endl;
/* affiche
5
3
*/
```

Pour ajouter un retour à la ligne

```
int x = 5, y(3);
cout << x << endl;
cout << y << endl;
/* affiche
5
3
*/
```

Ou en plus simple

```
int x = 5, y(3);
cout << x << endl << y << endl;
/* affiche
5
3
*/
```

Le compilateur **Visual C++** ne supporte pas la dernière écriture.

C++

Pour lire un entier

```
int x;
cin >> x;
```

C++

Pour lire un entier

```
int x;
cin >> x;
```

Problème de `cin` avec **Code Runner** de **VSC**

- Par défaut, **Code Runner** utilise la console **Output** (en écriture seule).
- Pour les opérations de lecture, il faut utiliser **Terminal**.
- Pour cela, il faut :
  - aller dans `File > Preferences > Settings`.
  - saisir `Code Runner` dans la zone de recherche
  - chercher `Code-runner: Run In Terminal` puis cocher la case correspondante

C++

`cin` est précédé souvent d'un `cout` explicatif

```
int x;
cout << "saisir un int ";
cin >> x;
cout << "La valeur saisie est " << x << endl;
```

C++

`cin` est précédé souvent d'un `cout` explicatif

```
int x;
cout << "saisir un int ";
cin >> x;
cout << "La valeur saisie est " << x << endl;
```

Pour lire un booléen

```
bool b;
cout << "saisir un bool ";
cin >> b;
cout << "La valeur saisie es t" << b << endl;
```

C++

Pour lire un caractère

```
char c;
cout << "saisir un char ";
cin >> c;
cout << "La valeur saisie est " << c << endl;
```

C++

Pour lire un caractère

```
char c;
cout << "saisir un char ";
cin >> c;
cout << "La valeur saisie est " << c << endl;
```

Attention au cas où l'on saisit plus d'un caractère pour le stocker dans une variable de type `char`

```
char c1;
cout << "saisir un char ";
// si on saisit aaaa
cin >> c1; // le premier a sera affecté à c1
char c2;
cin >> c2; // le deuxième a sera affecté à c2
string str;
cin >> str; // le reste sera affecté à str
cout << c1 << " " << c2 << " " << str;
// affiche a a aa
```

C++

Comment copier le contenu d'une variable dans une autre ?

```
int x (5);
int y = x;

cout << "le contenu de la variable x est "  << x <<  endl;
// affiche le contenu de la variable x est 5

cout << "le contenu de la variable y est "  << y <<  endl;
// affiche le contenu de la variable y est 5
```

C++

Comment copier le contenu d'une variable dans une autre ?

```
int x (5);
int y = x;

cout << "le contenu de la variable x est "  << x <<  endl;
// affiche le contenu de la variable x est 5

cout << "le contenu de la variable y est "  << y <<  endl;
// affiche le contenu de la variable y est 5
```

Modifier l'un $\not\Rightarrow$ modifier l'autre

```
y = 2;

cout << "le contenu de la variable x est "  << x <<  endl;
// affiche le contenu de la variable x est 5

cout << "le contenu de la variable y est "  << y <<  endl;
// affiche le contenu de la variable y est 2
```

C++

Remarque

Pour avoir deux variables qui pointent sur le même espace mémoire, il faut utiliser les références.

C++

Référence ?

- Étiquette (alias) d'une variable existante.
- Il est possible de créer une deuxième voire nième étiquette pour une même variable.
- Une étiquette $\equiv$ une référence vers une zone mémoire.

C++

Pour créer une référence, on utilise l'opérateur &

```
int &z = x;
```

C++

Pour créer une référence, on utilise l'opérateur &

```
int &z = x;
```

Ou aussi

```
int & z = x;
```

C++

Pour créer une référence, on utilise l'opérateur &

```
int &z = x;
```

Ou aussi

```
int & z = x;
```

Ou encore

```
int& z = x;
```

C++

Pour créer une référence, on utilise l'opérateur &

```
int &z = x;
```

Ou aussi

```
int & z = x;
```

Ou encore

```
int& z = x;
```

Les trois écritures sont équivalentes.

C++

L'opérateur & permet de récupérer l'adresse mémoire de la variable (en hexadécimal)

```
cout << &x << endl;
// affiche 0x61ff04
```

C++

Les deux variables `x` et `ref` ont le même contenu et la même référence mémoire

```
int x = 5;
int &ref = x;

cout << x << endl;
// affiche 5

cout << &x << endl;
// affiche 0x61ff04

cout << ref << endl;
// affiche 5

cout << &ref << endl;
// affiche 0x61ff04
```

C++

Modifier l'un $\Rightarrow$ modifier l'autre (la référence reste inchangée)

```
z = 2;

cout << x << endl;
// affiche 2

cout << &x << endl;
// affiche 0x61ff04

cout << ref << endl;
// affiche 2

cout << &ref << endl;
// affiche 0x61ff04
```

C++

On peut changer la valeur d'une référence mais pas l'adresse (Le code suivant génère une erreur)

```
int t = 10;
&ref = t;
```

C++

Exercice 1 : sans tester, qu'affiche le programme suivant ?

```
int a(2), b(5);
int &c = a;
c = b;
cout << c << endl
     << b << endl
     << a;
```

C++

Exercice 1 : sans tester, qu'affiche le programme suivant ?

```
int a(2), b(5);
int &c = a;
c = b;
cout << c << endl
     << b << endl
     << a;
```


Réponse

```
5
5
5
```

C++

Exercice 2 : sans tester, le programme suivant affiche t-il trois adresses identiques ?

```
int a(2), b(5);
int &c = a;
c = b;
cout << &c << endl
     << &b << endl
     << &a;
```

C++

Exercice 2 : sans tester, le programme suivant affiche t-il trois adresses identiques ?

```
int a(2), b(5);
int &c = a;
c = b;
cout << &c << endl
     << &b << endl
     << &a;
```


Réponse

```
0x61ff08
0x61ff04
0x61ff08
```

C++

Pointeur ?

- Une variable contenant l'adresse d'une autre variable de même type.
- On le déclare avec l'opérateur `*` et s'utilise conjointement avec l'opérateur de référence `&`.
- Possibilité de récupérer la valeur avec l'opérateur `*` (on appelle ça **dé-référencement**).

C++

Pour créer un pointeur, on utilise l'opérateur $\star$

```
int x = 5;
int* ptr = &x;
```

C++

Pour créer un pointeur, on utilise l'opérateur $\star$

```
int x = 5;
int* ptr = &x;
```

La déclaration suivante est aussi correcte

```
int * ptr = &x;
```

C++

Pour créer un pointeur, on utilise l'opérateur $\star$

```
int x = 5;
int* ptr = &x;
```

La déclaration suivante est aussi correcte

```
int * ptr = &x;
```

Ou encore

```
int *ptr = &x;
```

C++

Pour créer un pointeur, on utilise l'opérateur $\star$

```
int x = 5;
int* ptr = &x;
```

La déclaration suivante est aussi correcte

```
int * ptr = &x;
```

Ou encore

```
int *ptr = &x;
```

La première est la préférée.

Remarques

- `ptr` contient l'adresse de `x`.
- `*ptr` contient la valeur de `x`.
- `&ptr` contient l'adresse de `x`.

```
cout << x << endl;
// affiche 5

cout << &x << endl;
// affiche 0x61ff04

cout << ptr << endl;
// affiche 0x61ff04

cout << &ptr << endl;
// affiche 0x61ff00

cout << *ptr << endl;
// affiche 5
```

C++

Remarque

L'opérateur * est utilisé à la fois pour

- déclarer un pointeur.
- accéder à la valeur de la variable pointée par le pointeur.

C++

Et si on modifie la valeur du pointeur

```
*ptr = 2;
cout << *ptr << " " << x << endl;
// affiche  2 2
```

C++

Et si on modifie la valeur du pointeur

```
*ptr = 2;
cout << *ptr << " " << x << endl;
// affiche  2 2
```


Cependant, l'adresse ne change pas.

Dans un programme, un pointeur peut pointer sur plusieurs variables

```
int t = 10;
ptr = &t;

cout << x << endl;
// affiche 5

cout << &x << endl;
// affiche 0x61ff04

cout << t << endl;
// affiche 10

cout << &t << endl;
// affiche 0x61fefc

cout << ptr << endl;
// affiche 0x61fefc

cout << *ptr << endl;
// affiche 10

cout << &ptr << endl;
// affiche 0x61ff00
```

C++

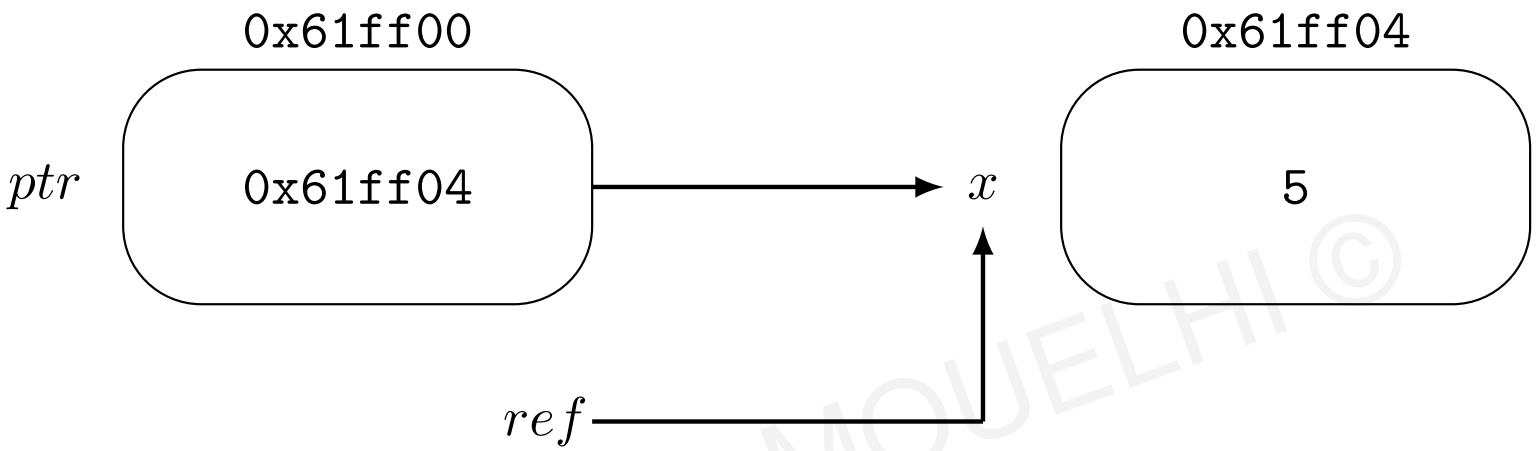
Remarques

- `ptr = &t` ⇔ `ptr` pointe désormais sur `t`.
- `*ptr = t` ⇔ La valeur de la variable pointée par `ptr` sera remplacée par celle de `t`.

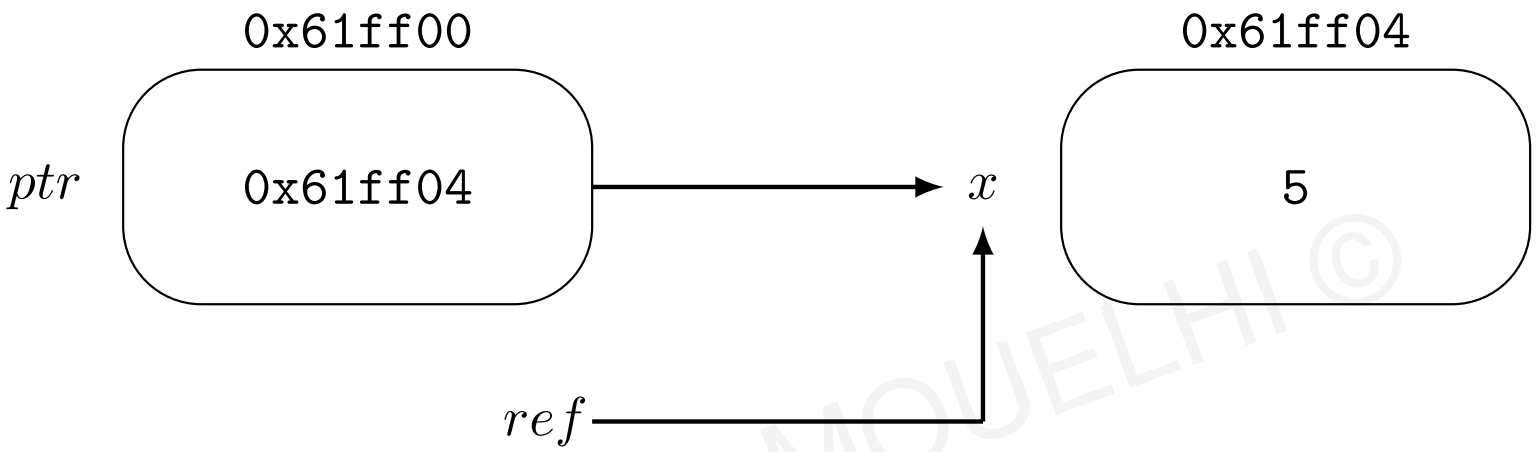
On peut changer la valeur de la variable pointée par le pointeur en utilisant ce dernier

```
int t = 10;
ptr = &t;

cout << x << endl;
// affiche 10

cout << &x << endl;
// affiche 0x61ff04

cout << t << endl;
// affiche 10

cout << &t << endl;
// affiche 0x61fefc

cout << ptr << endl;
// affiche 0x61ff04

cout << *ptr << endl;
// affiche 10

cout << &ptr << endl;
// affiche 0x61ff00
```

C++



C++



Référence

- Alias d'une variable déjà existante.
- Doit être initialisée à la déclaration.
- Peut référencer un seul objet.
- Ne peut contenir la valeur `NULL`.

Pointeur

- Variable contenant une adresse mémoire.
- Peut être initialisé à la déclaration ou après.
- Peut pointer sur plusieurs objets.
- Peut contenir la valeur `NULL`.

C++

Exercice 1 : sans tester, qu'affiche le programme suivant ?

```
int a(2), b(5);
int &c = a;
int *ptr = &c;
c = b;
cout << *ptr << endl
     << c << endl
     << b << endl
     << a;
```

C++

Exercice 1 : sans tester, qu'affiche le programme suivant ?

```
int a(2), b(5);
int &c = a;
int *ptr = &c;
c = b;
cout << *ptr << endl
     << c << endl
     << b << endl
     << a;
```


Réponse

```
5
5
5
5
```

C++

Exercice 2 : sans tester, le programme suivant affiche t-il quatre adresses identiques ?

```
int a(2), b(5);
int &c = a;
int *ptr = &c;
c = b;
cout << ptr << endl
     << &c << endl
     << &b << endl
     << &a;
```

C++

Exercice 2 : sans tester, le programme suivant affiche t-il quatre adresses identiques ?

```
int a(2), b(5);
int &c = a;
int *ptr = &c;
c = b;
cout << ptr << endl
     << &c << endl
     << &b << endl
     << &a;
```


Réponse

```
0x61ff04
0x61ff04
0x61ff00
0x61ff04
```

C++

Pour créer un pointeur de type entier (sans lui affecter la référence d'une deuxième variable)

```
int * p = new int;
```

C++

Pour créer un pointeur de type entier (sans lui affecter la référence d'une deuxième variable)

```
int * p = new int;
```

Pour initialiser la valeur du pointeur (dé-référencement)

```
*p = 10;
```

C++

Pour créer un pointeur de type entier (sans lui affecter la référence d'une deuxième variable)

```
int * p = new int;
```

Pour initialiser la valeur du pointeur (dé-référencement)

```
*p = 10;
```

Ou tout simplement

```
int * p = new int(10);
```

C++

Pour créer un pointeur de type entier (sans lui affecter la référence d'une deuxième variable)

```
int * p = new int;
```

Pour initialiser la valeur du pointeur (dé-référencement)

```
*p = 10;
```

Ou tout simplement

```
int * p = new int(10);
```

N'oublions pas de libérer l'espace mémoire dès qu'on finit d'utiliser le pointeur

```
delete p;
```

C++

On peut aussi créer un pointeur avec l'opérateur `new` et lui affecter une référence plus tard

```
int x = 5;
int *ptr = new int;
ptr = &x;
```

C++

Remarque

- Tout pointeur créé avec l'opérateur `new` doit être supprimé avec l'opérateur `delete`, après utilisation.
- Autrement, on aura une fuite de mémoire (**Memory leak**).

C++

Remarque

- Tout pointeur créé avec l'opérateur `new` doit être supprimé avec l'opérateur `delete`, après utilisation.
- Autrement, on aura une fuite de mémoire (**Memory leak**).

Memory leak (ou fuite de mémoire)

Un espace mémoire réservé mais plus jamais utilisé.

C++

Supprimer un pointeur $\nRightarrow$ supprimer la valeur de la variable pointée

```
int x = 5;
int *ptr = new int;
ptr = &x;

// suppression du pointeur
delete ptr;

cout << x << endl;
// affiche 5
```

C++

Après la suppression du pointeur, pensez à affecter la valeur `NULL` à ce dernier

```
int x = 5;
int *ptr = new int;
ptr = &x;

// suppression du pointeur
delete ptr;

// affectation de la valeur NULL
ptr = NULL;
```

C++

Un pointeur peut être copié

```
int *ptr1 = new int(2);
int *ptr2 = ptr1;

cout << "ptr1 : " << ptr1 << ' ' << &ptr1 << ' ' << *ptr1 << endl;
// affiche ptr1 : 0x1923c746a60 0x33e71ff830 2

cout << "ptr2 : " << ptr2 << ' ' << &ptr2 << ' ' << *ptr2 << endl;
// affiche ptr2 : 0x1923c746a60 0x33e71ff828 2
```

C++

Supprimer une copie ⇒ supprimer tous les autres

```
int *ptr1 = new int(2);
int *ptr2 = ptr1;

cout << "ptr1 : " << ptr1 << ' ' << &ptr1 << ' ' << *ptr1 << endl;
// affiche ptr1 : 0x1923c746a60 0x33e71ff830 2

cout << "ptr2 : " << ptr2 << ' ' << &ptr2 << ' ' << *ptr2 << endl;
// affiche ptr2 : 0x1923c746a60 0x33e71ff828 2

// suppression de la copie
delete ptr2;
ptr2 = NULL;

cout << "ptr1 : " << ptr1 << ' ' << &ptr1 << ' ' << *ptr1 << endl;
// affiche ptr1 : 0 0x33e71ff830 -1453319488

cout << "ptr2 : " << ptr2 << ' ' << &ptr2 << ' ' << *ptr2 << endl;
// affiche ptr2 : 0x1923c746a60 0x33e71ff828
```

C++

Deux problèmes avec les pointeurs

- Risque de fuite de mémoire
- Suppression de copie

C++

Solution avec les pointeurs intelligents [**C++11**]

- Classes définies dans le fichier `memory`
- Pas besoin d'appeler `delete` après utilisation : **Garbage collector** s'en occupera.
- Trois types de pointeur intelligent
  - `unique_ptr` : permet de définir un seul pointeur sur un espace mémoire (pas de copie)
  - `shared_ptr` : permet de définir plusieurs pointeurs sur un espace mémoire
  - `weak_ptr` : cas particulier de `shared_ptr`

Commençons par inclure `memory`

```
#include <memory>
```

C++

Commençons par inclure `memory`

```
#include <memory>
```

Pour créer un pointeur `unique_ptr`

```
unique_ptr<int> ptr1(new int(2));
```

C++

Pour afficher l'adresse de ce pointeur

```
cout << &ptr1 << endl;
```

C++

Pour afficher l'adresse de ce pointeur

```
cout << &ptr1 << endl;
```

Pour afficher la valeur de ce pointeur

```
cout << *ptr1 << endl;
```

C++

Pour afficher l'adresse de ce pointeur

```
cout << &ptr1 << endl;
```

Pour afficher la valeur de ce pointeur

```
cout << *ptr1 << endl;
```

Pour afficher l'adresse de la variable pointée

```
cout << ptr1.get() << endl;
```

C++

Pour afficher l'adresse de ce pointeur

```
cout << &ptr1 << endl;
```

Pour afficher la valeur de ce pointeur

```
cout << *ptr1 << endl;
```

Pour afficher l'adresse de la variable pointée

```
cout << ptr1.get() << endl;
```

Le code suivant ne sera pas accepté par le compilateur car on ne peut copier un `unique_ptr`

```
unique_ptr<int> ptr2(ptr1);
```

C++

Un `unique_ptr` peut transférer sa propriété vers un autre `unique_ptr` en appelant la méthode `move`

```
unique_ptr<int> ptr1(new int(2));

cout << "ptr1: " << ptr1.get() << ' ' << &ptr1 << ' ' << *ptr1 << endl;
// ptr1: 0x1d361e91750 0x1001ffd70 2

unique_ptr<int> ptr2 = move(ptr1);

cout << "ptr1: " << ptr1.get() << ' ' << &ptr1 <<  endl;
// ptr1: 0x1d361e91750 0x1001ffd70

cout << "ptr2: " << ptr2.get() << ' ' << &ptr2 << ' ' << *ptr2 << endl;
// ptr2: 0x1d361e91750 0x1001ffd60 2
```

C++

Un `shared_ptr` peut être copié plusieurs fois

```
shared_ptr<int> ptr1(new int(2));
shared_ptr<int> ptr2(ptr1);
shared_ptr<int> ptr3(ptr1);

cout << "ptr1: " << ptr1.get() << ' ' << &ptr1 << ' ' << *ptr1 << endl;
cout << "ptr2: " << ptr2.get() << ' ' << &ptr2 << ' ' << *ptr2 << endl;
cout << "ptr3: " << ptr3.get() << ' ' << &ptr3 << ' ' << *ptr3 << endl;
```

C++

Un `shared_ptr` peut être copié plusieurs fois

```
shared_ptr<int> ptr1(new int(2));
shared_ptr<int> ptr2(ptr1);
shared_ptr<int> ptr3(ptr1);

cout << "ptr1: " << ptr1.get() << ' ' << &ptr1 << ' ' << *ptr1 << endl;
cout << "ptr2: " << ptr2.get() << ' ' << &ptr2 << ' ' << *ptr2 << endl;
cout << "ptr3: " << ptr3.get() << ' ' << &ptr3 << ' ' << *ptr3 << endl;
```


Le résultat

```
ptr1: 0x27864ef1750 0xb103fffc20 2
ptr2: 0x27864ef1750 0xb103fffc10 2
ptr3: 0x27864ef1750 0xb103fffc00 2
```

Pour supprimer le pointeur et lui affecter `NULL`

```
ptr1.reset();
```

C++

Pour supprimer le pointeur et lui affecter `NULL`

```
ptr1.reset();
```

Remarque

En supprimant un `shared_ptr`, les autres ne seront pas affectés et ils auront toujours accès à la ressource.

C++

Pour supprimer le pointeur et lui affecter `NULL`

```
ptr1.reset();
```

Remarque

En supprimant un `shared_ptr`, les autres ne seront pas affectés et ils auront toujours accès à la ressource.

Pour compter le nombre de pointeurs pointant la même ressource

```
ptr2.use_count()
```

C++

weak_ptr

- Une version faible ou légère de `shared_ptr`
- Ne sera pas compté avec la méthode `use_count()`
- Lorsque le dernier `shared_ptr` qui gère une ressource est supprimé, la ressource sera libérée, même s'il existe `weak_ptr` pointant vers cette même ressource.

C++

Exemple

```
shared_ptr<int> ptr1(new int(2));
shared_ptr<int> ptr2(ptr1);
weak_ptr<int> ptr3(ptr1);

cout << "ptr1 : " << ptr1.use_count() << endl;
cout << "ptr2 : " << ptr2.use_count() << endl;
cout << "ptr3 : " << ptr3.use_count() << endl;
```

C++

Exemple

```
shared_ptr<int> ptr1(new int(2));
shared_ptr<int> ptr2(ptr1);
weak_ptr<int> ptr3(ptr1);

cout << "ptr1 : " << ptr1.use_count() << endl;
cout << "ptr2 : " << ptr2.use_count() << endl;
cout << "ptr3 : " << ptr3.use_count() << endl;
```


Le résultat

```
ptr1: 2
ptr2: 2
ptr3: 2
```

C++

La méthode `expired` permet de vérifier si tous les `shared_ptr` associés à la ressource pointée par le `weak_ptr` ont été supprimés

```
shared_ptr<int> ptr1(new int(2));
shared_ptr<int> ptr2(ptr1);
weak_ptr<int> ptr3(ptr1);

cout << boolalpha << ptr3.expired() << endl;
// false

ptr1.reset();

cout << boolalpha << ptr3.expired() << endl;
// false

ptr2.reset();

cout << boolalpha << ptr3.expired() << endl;
// true
```

C++

`weak_ptr` **est un pointeur léger, il ne peut utiliser l'opérateur** `*` **mais peut appeler la méthode** `lock`

```
shared_ptr<int> ptr1(new int(2));
shared_ptr<int> ptr2(ptr1);
weak_ptr<int> ptr3(ptr1);
cout << "ptr1 : " << ptr1.get() << ' ' << &ptr1 << ' ' << *ptr1 << endl;
cout << "ptr2 : " << ptr2.get() << ' ' << &ptr2 << ' ' << *ptr2 << endl;
cout << "ptr3 : " << ptr3.lock() << ' ' << &ptr3 << ' ' << *ptr3.lock() << endl;
```

C++

`weak_ptr` **est un pointeur léger, il ne peut utiliser l'opérateur** `*` **mais peut appeler la méthode** `lock`

```
shared_ptr<int> ptr1(new int(2));
shared_ptr<int> ptr2(ptr1);
weak_ptr<int> ptr3(ptr1);
cout << "ptr1 : " << ptr1.get() << ' ' << &ptr1 << ' ' << *ptr1 << endl;
cout << "ptr2 : " << ptr2.get() << ' ' << &ptr2 << ' ' << *ptr2 << endl;
cout << "ptr3 : " << ptr3.lock() << ' ' << &ptr3 << ' ' << *ptr3.lock() << endl;
```

Le résultat

```
ptr1 : 0x249972d1750 0x96f29ff660 2
ptr2 : 0x249972d1750 0x96f29ff650 2
ptr3 : 0x249972d1750 0x96f29ff640 2
```

C++

Pour les variables numériques (`int`, `float`...)

- `=` : affectation
- `+` : addition
- `-` : soustraction
- `*` : multiplication
- `/` : division
- `%` : reste de la division

C++

Exercice

Écrire un code **C++** qui

- demande à l'utilisateur de saisir deux entiers `a` et `b`
- affiche le résultat de l'équation $\texttt{a} * x + \texttt{b} = 0$

C++

Quelques raccourcis

- `i = i + 1` ⇒ `i++;`
- `i = i - 1` ⇒ `i--;`
- `i = i + 2` ⇒ `i += 2;`
- `i = i - 2` ⇒ `i -= 2;`

C++

Exemple de post-incrémentation

```
int i = 2;
int j = i++;
cout  << i << endl; // affiche 3
cout  << j << endl; // affiche 2
```

C++

Exemple de post-incrémentation

```
int i = 2;
int j = i++;
cout  << i << endl; // affiche 3
cout  << j << endl; // affiche 2
```

Exemple de pre-incrémentation

```
int i = 2;
int j = ++i;
cout  << i << endl; // affiche 3
cout  << j << endl; // affiche 3
```

C++

L'opérateur + permet de concaténer deux chaînes de caractère

```
string str1 = "bon";
string str2 = "jour";
string concatResult = str1 + str2;
cout  << concatResult << endl;
// affiche bonjour
```

C++

Opérations sur les `string` (exemple avec une variable `str`)

- `str.length()` : retourne le nombre de caractère de la chaîne `str`.
- `str.find(str2)` : retourne l'indice de la première occurrence de la valeur de `str2` dans `str`, `string::pos` (constante) sinon.
- `str[i]` : retourne le caractère d'indice `i` dans `str`.
- `str.substr(i, j)` : permet d'extraire j caractères de `str` à partir de l'indice `i`
- `str.compare(str2)` : permet de comparer `str` à `str2` et retourne `0` en cas d'égalité, `1` si le code ASCII du premier caractère différent de la première chaîne est supérieur à celui de la deuxième chaîne, `-1` sinon.
- `str.replace(posI, posF, str2)` : permet de remplacer les caractères situés entre `posI` et `posF` dans `str` par `str2` et retourne la nouvelle chaîne
- ...

C++

Commençons par déclarer la chaîne de caractère suivante

```
string str = "bonjour";
```

C++

Commençons par déclarer la chaîne de caractère suivante

```
string str = "bonjour";
```

Pour afficher la longueur de la chaîne

```
cout << str.length() <<  endl;
// affiche 7
```

C++

Commençons par déclarer la chaîne de caractère suivante

```
string str = "bonjour";
```

Pour afficher la longueur de la chaîne

```
cout << str.length() <<  endl;
// affiche 7
```


Pour afficher le caractère d'indice 3 (le premier caractère est d'indice 0)

```
cout << str[3] <<  endl;
// affiche j
```

C++

Pour récupérer la position de la première occurrence d'une sous-chaîne dans une chaîne

```
cout << str.find("jour") <<  endl;
// affiche 3
```

C++

Pour récupérer la position de la première occurrence d'une sous-chaîne dans une chaîne

```
cout << str.find("jour") <<  endl;
// affiche 3
```

`find` **permet de chercher un caractère**

```
cout << str.find('o') << endl;
// affiche 1
```

C++

Pour récupérer la position de la première occurrence d'une sous-chaîne dans une chaîne

```
cout << str.find("jour") <<  endl;
// affiche 3
```

`find` **permet de chercher un caractère**

```
cout << str.find('o') << endl;
// affiche 1
```

Pour extraire une sous-chaîne de 4 caractères à partir de la position 3

```
cout << str.substr(3, 4) <<  endl;
// affiche jour
```

C++

Pour comparer deux chaînes de caractères

```
string str = "bonjour";
string str2 = "bonsoir";

cout << str.compare(str2) << endl;
// affiche -1 car 'j' < 's'

cout << str2.compare(str) << endl;
// affiche 1 car 's' > 'j'

cout << str.compare(str) <<  endl;
// affiche 0
```

C++

On peut aussi utiliser les opérateurs == et !=

```
cout << (str == str2) << endl;
// affiche 0

cout << (str2 == str2) << endl;
// affiche 1

cout << (str != str2) << endl;
// affiche 1

cout << (str2 != str2) << endl;
// affiche 0
```

Remarque

D'après la documentation officielle, `operator==` retourne `lhs.compare(rhs) == 0`.

C++

Pour remplacer une sous-chaîne dans une chaîne par une nouvelle sous-chaîne

```
cout << str.replace(3, 2, "soir") << endl;
// affiche bonsoirur
```

Explication

- on insère la sous-chaîne `soir` dans `str` à partir de la position 3
- les deux premiers caractères dans `str`, à partir de la position 3, seront supprimés
- le reste de la chaîne sera décalé

C++

Exercice : qu'affiche le code suivant ?

```
int main()
{
      string str = "bonjour";
      str[0] = 'B';
      cout << str << endl;
      return 0;
}
```

C++

Exercice : qu'affiche le code suivant ?

```
int main()
{
      string str = "bonjour";
      str[0] = 'B';
      cout << str << endl;
      return 0;
}
```


Question

Que peut t-on constater si on remplace `string` par `auto` ?

C++

Explication

- Un texte entouré par `" . . . "` déclaré avec le mot-clé `auto` est de type `char *` (pointeur sur le premier caractère d'une suite de caractères)
- `char *` : type hérité du langage C ($\neq$ `string`)

C++

Explication

- Un texte entouré par `"..."` déclaré avec le mot-clé `auto` est de type `char *` (pointeur sur le premier caractère d'une suite de caractères)
- `char *` : type hérité du langage C ($\neq$ `string`)

Remarque

- Pour convertir `char *` en `string`, utiliser `string(char *)`
- Pour convertir `string` en `char *`, utiliser `c_str(string)`

Conversion implicite (entre type compatible)

```
short x = 5;
int y = x;
cout << y << endl;
// affiche 5
```

Conversion implicite (entre type compatible)

```
short x = 5;
int y = x;
cout << y << endl;
// affiche 5
```

Attention au dépassement

```
int x = 50000;
short y = x;
cout << y << endl;
// affiche -15536
```

C++

On peut aussi spécifier le type explicitement

```
double pi = 3.14;
int x;
x = int (pi);
cout << x << endl;
// affiche 3
```

C++

On peut aussi spécifier le type explicitement

```
double pi = 3.14;
int x;
x = int (pi);
cout << x << endl;
// affiche 3
```

Ou aussi

```
double pi = 3.14;
int x;
x = (int) pi;
cout << x << endl;
// affiche 3
```

C++

Conversion de chaîne de caractère en entier (string to int : `stoi`)

```
string str = "100";
int x = stoi(str);
cout << (x * 20) << endl;
// affiche 2000
```

C++

Conversion de chaîne de caractère en entier (string to int : `stoi`)

```
string str = "100";
int x = stoi(str);
cout << (x * 20) << endl;
// affiche 2000
```


Si la chaîne contient un caractère non numérique, on aura l'erreur `std::invalid_argument`

```
string str = "a";
int x = stoi(str);
cout << x << endl;
```

C++

Si la chaîne commence par des caractères numériques et ensuite d'autres non numériques, alors pas d'erreur

```
string str = "100a";
int x = stoi(str);
cout << x << endl;
// affiche 100
```

C++

Si la chaîne commence par des caractères numériques et ensuite d'autres non numériques, alors pas d'erreur

```
string str = "100a";
int x = stoi(str);
cout << x << endl;
// affiche 100
```

Si le début est alphabétique et la suite est numérique, on aura aussi l'erreur `std::invalid_argument`

```
string str = "a100";
int x = stoi(str);
cout << x << endl;
```

Conversion d'un entier en chaîne de caractères [C++11]

```
int x = 100;
string s = to_string(x);
cout << s.length() << endl;
// affiche 3
```

Conversion d'un caractère en entier

```
char c = '7';
int x = (int)c - 48;
cout << x << endl;
// affiche 7
```

Conversion d'un caractère en entier

```
char c = '7';
int x = (int)c - 48;
cout << x << endl;
// affiche 7
```


Attention, le code suivant retourne le code ASCII

```
char c = '7';
int x = (int)c;
cout << x << endl;
// affiche 55
```

Conversion d'un entier en caractère

```
int x = 8;
string s = to_string(x);
char c = s[0];
cout << c <<  endl;
// affiche 7
```

C++

Librairie cmath

= { fonctions mathématiques prédéfinies }

C++

Librairie cmath

= { fonctions mathématiques prédéfinies }

Pour l'utiliser, il faut commencer par l'inclure

```
#include <cmath>
```

C++

Quelques fonctions de cmath

- `abs(x)` : retourne la valeur absolue de `x`
- `pow(x, y)` : retourne la `x` puissance `y`
- `max(x, y)` : retourne le max de `x` et `y`
- `min(x, y)` : retourne le min de `x` et `y`
- `sqrt(x)` : retourne la racine carré de `x`
- `floor(x)`, `ceil(x)` et `round(x)` : retournent l'arrondi de `x`
- ...

C++

Exemple avec sqrt

```
int x = 9;
cout << sqrt(x) <<  endl;
// affiche 9
```

C++

Exemple avec `floor(x)`, `ceil(x)` **et** `round(x)`

```
cout << round(1.9) << endl; // affiche 2
cout << round(1.5) << endl; // affiche 2
cout << round(1.4) << endl; // affiche 1
cout << ceil(1.9) << endl; // affiche 2
cout << ceil(1.5) << endl; // affiche 2
cout << ceil(1.4) << endl; // affiche 2
cout << floor(1.9) << endl; // affiche 1
cout << floor(1.5) << endl; // affiche 1
cout << floor(1.4) << endl; // affiche 1
```

C++

La librairie typeinfo

elle permet de déterminer le type d'une valeur

C++

La librairie typeinfo

elle permet de déterminer le type d'une valeur

Il faut commencer par l'inclure

```
#include <typeinfo>
```

C++

Exemple avec types différents

```
int i = 7;
string str = "bonjour";

cout << (typeid(i).name() == typeid(int).name()) << endl;
// affiche 1

cout << (typeid(str).name() == typeid(string).name()) << endl;
// affiche 1

cout << (typeid(i).name() == typeid(string).name()) << endl;
// affiche 0

cout << (typeid(2).name() == typeid(int).name()) << endl;
// affiche 1

cout << (typeid(2).name() == typeid(float).name()) << endl;
// affiche 0
```

C++

Constante ?

- Élément qui ne peut changer de valeur
- Déclaré avec les mot-clé `const` ou `#define`

C++

Constante ?

- Élément qui ne peut changer de valeur
- Déclaré avec les mot-clé `const` ou `#define`

Déclaration d'une constante

```
const double PI = 3.1415;
```

C++

Constante ?

- Élément qui ne peut changer de valeur
- Déclaré avec les mot-clé `const` ou `#define`

Déclaration d'une constante

```
const double PI = 3.1415;
```

L'instruction déclenche l'erreur suivante : error : assignment of read-only variable 'pi'

```
PI = 5;
```

C++

Dans le contexte global, la déclaration pourrait être simplifiée ainsi

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

#define PI 3.1415

int main()
{

    cout << PI;
    // affiche 3.1415
    return 0;
}
```

C++

Exécuter si une condition est vraie

```
if (condition)
{
    ...
}
```

C++

Exemple

```
int x = 3;
if (x > 0)
{
   cout << x << " est strictement positif" <<  endl;
}
```

C++

Exemple

```
int x = 3;
if (x > 0)
{
   cout << x << " est strictement positif" <<  endl;
}
```


Pour les conditions, on utilise les fonctions ou les opérateurs de comparaison

C++

Opérateurs de comparaison

- == : pour tester l'égalité
- != : pour tester l'inégalité
- > : supérieur à
- < : inférieur à
- >= : supérieur ou égal à
- <= : inférieur ou égal à

C++

Opérateurs de comparaison

- == : pour tester l'égalité
- != : pour tester l'inégalité
- > : supérieur à
- < : inférieur à
- >= : supérieur ou égal à
- <= : inférieur ou égal à

En **C++**, on ne peut comparer deux valeurs de type incompatible.

C++

Exercice

Écrire un code **C++** qui demande à l'utilisateur de saisir un entier positif et qui affiche ensuite sa parité (sans `else`).

C++

Exécuter un premier bloc si une condition est vraie, un deuxième sinon (le bloc `else`)

```
if (condition)
{
   ...
}
else
{
   ...
}
```

C++

Exemple

```
int x = 3;

if (x > 0)
{
    cout << x << " est strictement positif" << endl;
}
else
{
    cout << x << " est strictement négatif ou nul" <<  endl;
}
```

C++

Exercice

Écrire un programme **C++** qui permet de déterminer si une chaîne de caractères saisie par l'utilisateur contient un nombre pair ou impair de caractères.

C++

Exécuter un premier bloc si une condition est vraie, un deuxième sinon (voire un troisième ... nième)

```
if (condition1)
{
    ...
}
else if (condition2)
{
    ...
}
...
else
{
    ...
}
```

C++

Exemple

```
int x = -3;

if (x > 0)
{
    cout << x << " est strictement positif" << endl;
}
else if (x < 0)
{
    cout << x << " est strictement négatif" << endl;
}
else
{
    cout << x << " est nul" <<  endl;
}
```

C++

Exemple

```
int x = -3;

if (x > 0)
{
    cout << x << " est strictement positif" << endl;
}
else if (x < 0)
{
    cout << x << " est strictement négatif" << endl;
}
else
{
    cout << x << " est nul" <<  endl;
}
```

On peut tester plusieurs conditions à la fois en utilisant les opérateurs logiques.

C++

Exercice

Écrire un code **C++** qui

- demande à l'utilisateur de saisir trois entiers `a`, `b` et `c`
- affiche le résultat de l'équation $\texttt{a}x^2 \texttt{ + b}x \texttt{ + c = 0}$.

C++

On peut tester plusieurs conditions à la fois en utilisant les opérateurs logiques.

C++

On peut tester plusieurs conditions à la fois en utilisant les opérateurs logiques.

Opérateurs logiques

- `&&` : et
- `||` : ou
- `!` : non
- `^` : ou exclusif

C++

Tester plusieurs conditions

```
if (condition1 && !condition2 || condition3)
{
    ...
}
[else ...]
```

C++

Exercice 1

Écrire un code **C++** qui

- demande à l'utilisateur de saisir une année (un entier),
- affiche si l'année saisie est bissextile (voir `https://fr.wikipedia.org/wiki/Ann%C3%A9e_bissextile`).

C++

Exercice 2

Écrire un code **C++** qui

- demande à l'utilisateur de saisir deux entiers `a` et `b` différents de zéro,
- affiche le signe du résultat de la multiplication sans calculer le produit.

C++

Exercice 3

Écrire un code **C++** qui

- demande à l'utilisateur de saisir deux entiers `a` et `b`,
- détermine et affiche si le résultat de l'addition (sans calculer la somme) est pair ou impair.

L'opérateur ternaire (Elvis Opertor) est supporté par C++

```
condition ? traitementsSiConditionVraie : traitementsSiConditionFausse;
```

C++

L'opérateur ternaire (Elvis Opertor) est supporté par C++

```
condition ? traitementsSiConditionVraie : traitementsSiConditionFausse;
```

Exemple

```
int z {2};
string b = z % 2 == 0 ? "pair" : "impair";
cout  << b << endl;
// affiche pair
```

C++

Structure conditionnelle `switch` : syntaxe

```
switch (nomVariable)
{
  case constante-1:
    groupe-instructions-1;
    break;
  case constante-2:
    groupe-instructions-2;
    break;
  ...
  case constante-N:
    groupe-instructions-N;
    break;
  default:
    groupe-instructions-par-défaut;
}
```

C++

Remarques

- Le `switch` permet **seulement** de tester l'égalité.
- Le `break` permet de quitter le `switch` une fois le bloc `case` est vérifié.
- Le `break` nous évite d'entourer le bloc d'instructions de `case` par des { } $\Rightarrow$ possibilité de regrouper plusieurs `case`.
- Le bloc `default` est facultatif, il sera exécuté si la valeur de la variable ne correspond à aucune constante de `case`.
- En **C++**, la variable de `switch` doit être de type entier.

C++

Structure conditionnelle `switch` : exemple

```
int x = 2;
switch (x)
{
    case 1:
      cout << "un" << endl;
      break;
    case 2:
      cout << "deux" << endl;
      break;
    case 3:
      cout << "trois" << endl;
      break;
    default:
      cout << "autre" << endl;
}

// affiche deux
```

C++

En l'absence du `break`, le `case` suivant sera aussi exécuté

```
int x = 2;
switch (x)
{
    case 1:
      cout << "un" << endl;
      break;
    case 2:
      cout << "deux" << endl;
    case 3:
      cout << "trois" << endl;
      break;
    default:
      cout << "autre" << endl;
}

// affiche deux trois
```

C++

Exemple avec regroupement de `case`

```
int x = 5;
switch (x)
{
  case 1:
  case 2:
  case 3:
    cout << "un, deux ou trois" << endl;
    break;
  case 4:
    cout << "quatre" << endl;
    break;
  case 5:
  case 6:
    cout << "cinq ou six" << endl;
    break;
  default:
    cout << "autre" << endl;
}

// affiche cinq ou six
```

C++

Si la variable dans `switch` est un `char`, alors c'est le code ASCII qui sera testé

```
auto x = '2';
switch (x)
{
    case 1:
      cout << "un" << endl;
      break;
    case 50:
      cout << "deux" << endl;
      break;
    case 3:
      cout << "trois" << endl;
      break;
    default:
      cout << "autre" << endl;
}

// affiche deux car 50 est le code ASCII de 2
```

C++

Exercice

Écrire un programme qui demande à l'utilisateur de saisir l'indice d'un mois (entier compris entre 1 et 12) et qui retourne le nombre de jours de ce mois

- si l'entier est égal à 1, 3, 5, 7, 8, 10 ou 12 le programme affiche 31
- sinon si l'entier est égal à 4, 6, 9 ou 11 le programme affiche 30
- sinon si l'entier est égal à 2, le programme demande à l'utilisateur de saisir l'année et lui retourne 29 si l'année est bissextile, 28 sinon (voir `https://fr.wikipedia.org/wiki/Ann%C3%A9e_bissextile`)
- pour toute autre valeur, le programme affiche une erreur

C++

Boucle `while` : à chaque itération on teste si la condition est vraie avant d'accéder aux traitements

```
while (condition[s]) {
   ...
}
```

C++

Boucle `while` : à chaque itération on teste si la condition est vraie avant d'accéder aux traitements

```
while (condition[s]) {
   ...
}
```


Attention aux boucles infinies, vérifier que la condition d'arrêt sera bien atteinte après un certain nombre d'itérations.

C++

Exemple

```
auto i = 0;
while (i < 5) {
    cout << i << endl;
    i++;
}
```

C++

Exemple

```
auto i = 0;
while (i < 5) {
    cout << i << endl;
    i++;
}
```


Le résultat est

```
0
1
2
3
4
```

Étant donnée la chaîne de caractères suivante

```
string maChaine = "hello les holoulos";
```

C++

Étant donnée la chaîne de caractères suivante

```
string maChaine = "hello les holoulos";
```

Exercice

Écrire un code **C++** qui permet de remplacer chaque caractère d'indice impair dans `maChaine` par son équivalent en majuscule.

Solution

```
int i = 0;
while (i < maChaine.length())
{
    maChaine[i] = (maChaine[i] - 32);
    i += 2;
}
cout << maChaine << endl;
```

C++

La Boucle `do` ... `while` exécute le bloc au moins une fois ensuite elle vérifie la condition

```
do {
   ...
}
while (condition[s]);
```

C++

La Boucle `do` ... `while` exécute le bloc au moins une fois ensuite elle vérifie la condition

```
do {
   ...
}
while (condition[s]);
```


Attention aux boucles infinies, vérifier que la condition d'arrêt sera bien atteinte après un certain nombre d'itérations.

C++

Exemple

```
auto i = 0;
do {
    cout << i << endl;
    i++;
} while (i < 5);
```

C++

Exemple

```
auto i = 0;
do {
     cout << i << endl;
     i++;
} while (i < 5);
```


Le résultat est

```
0
1
2
3
4
```

C++

Boucle `for`

```
for (initialisation; condition[s]; incrémentation) {
   ...
}
```

C++

Boucle `for`

```
for (initialisation; condition[s]; incrémentation) {
   ...
}
```


Attention aux boucles infinies si vous modifiez la valeur du compteur à l'intérieur de la boucle.

C++

Exemple

```
for (auto i = 0; i < 5; i++) {
    cout << i << endl;
}
```

C++

Exemple

```
for (auto i = 0; i < 5; i++) {
    cout << i << endl;
}
```

Le résultat est

```
0
1
2
3
4
```

C++

Exercice

Écrire un code **C++** qui permet d'afficher les nombres pairs compris entre 0 et 10.

C++

Exercice

Écrire un code **C++** qui permet d'afficher les nombres pairs compris entre 0 et 10.

Première solution

```
for (auto i = 0; i < 10; i++) {
  if (i % 2 == 0) {
      cout << i << endl;
  }
}
```

C++

Exercice

Écrire un code **C++** qui permet d'afficher les nombres pairs compris entre 0 et 10.

Première solution

```
for (auto i = 0; i < 10; i++) {
  if (i % 2 == 0) {
      cout << i << endl;
  }
}
```

Deuxième solution

```
for (auto i = 0; i < 10; i += 2) {
    cout << i << endl;
}
```

C++

Étant donnée la chaîne de caractères suivante

```
string maChaine = "Une première phrase. Et voici une
    deuxième.";
```

C++

Étant donnée la chaîne de caractères suivante

```
string maChaine = "Une première phrase. Et voici une
    deuxième.";
```

Exercice

Écrire un code **C++** qui permet de compter le nombre de mots et de phrases dans `maChaine`.

Remarques

Dans ces structures itératives, on peut utiliser :

- `break` : pour quitter la boucle
- `continue` : pour ignorer l'itération courante
- `goto` : pour renvoyer vers un label.

C++

Exemple avec `break`

```
int j = 5;
do
{
    cout << j << endl;
    if (j == 3)
    {
        break;
    }
    j--;
} while (j > 0);
```

C++

Exemple avec `break`

```
int j = 5;
do
{
    cout << j << endl;
    if (j == 3)
    {
        break;
    }
    j--;
} while (j > 0);
```

Résultat

```
5
4
3
```

C++

Considérons le code suivant

```
for (int i = 0; i < 5; i++)
{
    for (int j = 0; j < i + 1; j++)
    {
        cout << "*";
    }
    cout << endl;
}
```

C++

Considérons le code suivant

```
for (int i = 0; i < 5; i++)
{
    for (int j = 0; j < i + 1; j++)
    {
        cout << "*";
    }
    cout << endl;
}
```


Résultat

```
*
**
***
****
*****
```

C++

Si nous voulions ajouter une condition avec un `break`, dans le deuxième `for`, qui permet de quitter les deux boucles

```
int limit = 6, compteur = 0;
for (int i = 0; i < 5; i++)
{
    for (int j = 0; j < i + 1; j++)
    {
        cout << "*";
        compteur++;
        if (compteur == limit)
        {
            break;
        }
    }
    cout << endl;
}
```

C++

Problème

Malheureusement, le résultat reste inchangé car le `break` permet de quitter seulement la deuxième boucle.

C++

Problème

Malheureusement, le résultat reste inchangé car le `break` permet de quitter seulement la deuxième boucle.

Solution

Définir un label et utiliser `goto`.

C++

Solution avec goto

```
int limit = 6, compteur = 0;
for (int i = 0; i < 5; i++)
{
    for (int j = 0; j < i + 1; j++)
    {
        cout << "*";
        compteur++;
        if (compteur == limit)
        {
            goto exit;
        }
    }
    cout << endl;
}
exit : return 0;
```

C++

Exemple avec `continue`

```
int j = 5;
while (j > 0)
{
    if (j == 3)
    {
        j--;
        continue;
    }
    j--;
    cout << j << endl;
}
```

C++

Exemple avec `continue`

```
int j = 5;
while (j > 0)
{
    if (j == 3)
    {
        j--;
        continue;
    }
    j--;
    cout << j << endl;
}
```


Résultat

```
4
3
1
0
```

Remarque

`break`, `goto` et `continue` rendent le code difficile à lire, à l'exception de l'utilisation de `break` dans `switch`, ils ne doivent pas être utilisés.