BIN222 - C++

IPSA / N.DECKNER 2024

Cours 1 - Types et opérateurs

Notions : Type, constante, adresse, pointeur, référence, structure, énumération, commentaire.

Introduction

C++ est un langage de programmation haut niveau orienté objet créé par Bjarne Stroustrup dans les années 1980. Il supporte la programmation bas niveau et la programmation système.

C++ est un langage compilé. Pour exécuter un programme, le compilateur C++ doit produire un fichier objet par fichier de code source. Les fichiers objets sont ensuite combinés par l'éditeur de liens pour générer un programme exécutable.

Il y a eu six versions du langage C++: C++98, C++03, C++11, C++14, C++17 et C++20.

Types

Voici les types les plus utilisés avec un exemple d'initialisation:

```
bool x = true;                                  // Booléen (8 bits)
char x = "c";                                   // Caractère (8 bits)
int x = 1;                                      // Nombre entier (32 bits)
double x = 1.1;                                 // Nombre décimal (64 bits)
auto x = 1;                                     // déduit à la compilation

string x = "abc";                               // Chaîne de caractères
vector<int> x = {1, 2, 3}                       // tableau dynamique
map<string, int> x = {{"a", 1}, {"b", 2}, {"c", 3}};   // conteneur de clés/valeurs
```

Opérateurs

Opérateurs arithmétiques

`+, -, *, /, //, %`

Opérateurs de comparaisons

`<, >, <=, >=, !=, ==`

Opérateurs logiques

`&&, ||, !`

Opérateurs d'incrémentation

`++, --`

opérateurs d'assignation

`+=, -=, *=, /=`

Constantes

Les constantes sont des variables qui ne peuvent pas être modifiées.

```
const int x = 1
constexpr int x = 1 + 1        // expression évaluée à la compilation
```

Pointeurs et références

Les pointeurs et les références servent à manipuler les données et à gérer la mémoire.

L'opérateur &

`&x` retourne l'adresse mémoire de la variable `x`.

Pointeurs

Un pointeur est une variable qui stocke l'adresse mémoire d'une autre variable. Il permet d'accéder à la mémoire directement, ce qui peut être utile pour la gestion de la mémoire et pour accéder à des données complexes.

Les pointeurs sont déclarés en utilisant l'opérateur `*`. `int* ptr` déclare un pointeur vers un entier.

Les pointeurs peuvent être utilisés pour accéder à la valeur à laquelle ils pointent en utilisant l'opérateur de déréférencement `*`. `*ptr` accède à la valeur pointée par `ptr`.

Les pointeurs peuvent être réassignés pour pointer vers d'autres emplacements mémoire.

Un pointeur peut valoir `nullptr`. On utilise cette valeur pour représenter un pointeur qui ne pointe sur rien. Par exemple: `int* ptr = nullptr;`

Exemple d'utilisation de pointeur

```
int x = 1;
int* ptr = &x;          // ptr pointe vers l'adresse mémoire de x
int valeur = *ptr;      // Accès à la valeur de x via le pointeur
```

Références

Une référence est un alias (un autre nom) pour une variable existante. Elle permet d'accéder à la valeur d'une variable sans utiliser de pointeur ni de copie de données.

Les références sont déclarées en utilisant l'opérateur `&`. `int& ref = x` déclare une référence à un entier.

Les références sont liées à la variable qu'elles référencent et ne peuvent pas être réassignées pour référencer une autre variable.

Les références garantissent l'absence de valeur nulle, car elles doivent être initialisées lors de leur déclaration.

Exemple d'utilisation d'une référence

```
int x = 1;
int& ref = x;           // Référence à la variable x
int valeur = ref;       // Accède à la valeur de x via la référence
```

Les pointeurs offrent plus de flexibilité pour gérer la mémoire et l'accès direct aux données, tandis que les références fournissent une syntaxe plus simple et plus sûre pour accéder aux données existantes sans copie.

Enumérations

```
enum class BikeType {
    vtt,
    vtc,
    route,
    ville
}

BikeType bike_type = BikeType::vtt
```

Structures

```
struct Point {
    int x;
    int y;
}

Point p;
p.x = 1;
p.y = 1
```

```
void init_point(Point & p) {
    p.x = 0;
    p.y = 0;
}

Point p;
init_point(p);
```

cin, cout et cerr

cin, cout et cerr sont des flux de la bibliothèque standard d'entrée/sortie `<iostream>`.

- cin est le flux d'entrée standard
- cout est le flux de sortie standard

- cerr est flux d'erreur standard

```cpp
#include <iostream>

int main() {
    int number;

    // Sortie
    std::cout << "Saisir un nombre: ";

    // Entrée
    std::cin >> number;


    // Erreur
    if (number < 0) {
        std::cerr << "Erreur" << std::endl;
    }

    return 0;
}
```

Commentaires

```cpp
// Commentaire sur une ligne

/* Commentaire
   sur plusieurs lignes */

int x = /* Commentaire dans une ligne */ 10;
```