**Tổng hợp kiến thức c++**

**1.Biến trong c++**

* Cú pháp khai báo biến:

type variable\_list;

* Quy tắc khai báo biến
* Một biến có thể có các chữ cái, chữ số và dấu gạch dưới.
* Tên biến chỉ có thể bắt đầu bằng bảng chữ cái và dấu gạch dưới. Nó không thể bắt đầu bằng chữ số.
* Không có khoảng trắng trong tên biến.
* Tên biến không phải là bất kỳ từ hoặc từ khóa dành riêng như int, float, vv.
* Các kiểu biến trong c++
* Biến local (địa phương): Một biến được khai báo bên trong hàm hoặc khối lệnh được gọi là biến địa phương
* Biến global (toàn cầu): Một biến được khai báo bên ngoài hàm hoặc khối lệnh được gọi là biến toàn cầu. Bất kỳ hàm nào cũng có thể thay đổi giá trị của biến toàn cầu. Nó có sẵn cho tất cả các chức năng.
* Biến static: Một biến được khai báo với từ khóa **static** được gọi là biến tĩnh. Nó giữ lại giá trị của nó sau nhiều lần gọi hàm.
* Biến automatic: Tất cả các biến trong C được khai báo trong khối lệnh, là các biến tự động theo mặc định. Bởi chúng ta có thể khai báo một cách rõ ràng biến tự động bằng cách sử dụng **từ khóa auto**.
* Biến external: Chúng ta có thể chia sẻ một biến trong nhiều tập tin mã nguồn C bằng cách sử dụng biến external. Để khai báo biến bên ngoài, bạn cần sử dụng từ khóa extern

**2.Các kiểu dữ liệu trong c++**

* **Kiểu dữ liệu trong C++** xác định loại dữ liệu mà một biến có thể lưu trữ như số nguyên, số thực, ký tự vv.

|  |  |  |
| --- | --- | --- |
| **Kiểu** | **Kích thước bộ nhớ** | **Vùng giá trị** |
| char | 1 byte | -127 tới 127 hoặc 0 tới 255 |
| unsigned char | 1 byte | 0 tới 255 |
| signed char | 1 byte | -127 tới 127 |
| int | 4 byte | -2147483648 tới 2147483647 |
| unsigned int | 4 byte | 0 tới 4294967295 |
| signed int | 4 byte | -2147483648 tới 2147483647 |
| short int | 2 byte | -32768 tới 32767 |
| unsigned short int | Range | 0 tới 65,535 |
| signed short int | Range | -32768 tới 32767 |
| long int | 4 byte | -2,147,483,647 tới 2,147,483,647 |
| signed long int | 4 byte | Tương tự như long int |
| unsigned long int | 4 byte | 0 tới 4,294,967,295 |
| float | 4 byte | +/- 3.4e +/- 38 (~7 chữ số) |
| double | 8 byte | +/- 1.7e +/- 308 (~15 chữ số) |
| long double | 8 byte | +/- 1.7e +/- 308 (~15 chữ số) |
| wchar\_t | 2 hoặc 4 byte | 1 wide character |

* Ví dụ :

#include <iostream>

using namespace std;

int main() {

   cout << "Kich co cua char la: " << sizeof(char) << endl;

   cout << "Kich co cua int la: " << sizeof(int) << endl;

   cout << "Kich co cua short int la: " << sizeof(short int) << endl;

   cout << "Kich co cua long int la: " << sizeof(long int) << endl;

   cout << "Kich co cua float la: " << sizeof(float) << endl;

   cout << "Kich co cua double la: " << sizeof(double) << endl;

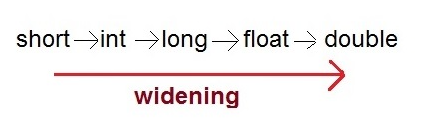
   cout << "Kich co cua wchar\_t la: " << sizeof(wchar\_t) << endl;

   return 0;

}

**3.Ép kiểu trong c++**

* **Ép kiểu trong C++** là việc gán giá trị của một biến có kiểu dữ liệu này tới biến khác có kiểu dữ liệu khác.Cú pháp : (type) value;
* **Phân loại ép kiểu trong Java:**
  + Nới rộng (widening): Là quá trình làm tròn số từ kiểu dữ liệu có kích thước nhỏ hơn sang kiểu có kích thước lớn hơn. Kiểu biến đổi này không làm mất thông tin.
  + Thu hẹp (narrowwing): Là quá trình làm tròn số từ kiểu dữ liệu có kích thước lớn hơn sang kiểu có kích thước nhỏ hơn. Kiểu biến đổi này có thể làm mất thông tin
* Nới rộng (widening)

Ví dụ : 

#include <iostream>

using namespace std;

int main() {

    int i = 100;

    long l = i;    // khong yeu cau chi dinh ep kieu

    float f = l;   // khong yeu cau chi dinh ep kieu

    cout << "Gia tri Int: " << i << endl;

    cout << "Gia tri Long: " << l << endl;

    cout << "Gia tri Float: " << f << endl;

    return 0;

}

* Thu hẹp (narrowwing)



Ví dụ:

#include <iostream>

int main() {

    double d = 100.04;

    long l = (long) d; // yeu cau chi dinh kieu du lieu (long)

    int i = (int) l; // yeu cau chi dinh kieu du lieu (int)

    cout << "Gia tri Double: " << d << endl;

    cout << "Gia tri Long: " << l << endl;

    cout << "Gia tri Int: " << i << endl;

    return 0;

}

**4.Mảng trong c++**

* Lợi thế của mảng trong c++:
* Tối ưu hóa code.
* Dễ dàng để duyệt các phần tử của mảng.
* Dễ dàng sắp xếp dữ liệu.
* Truy cập ngẫu nhiên.
* Bất lợi của mảng trong c++:Kích thước cố định: Kích thước của mảng trong C++ phải được định nghĩa tại thời điểm khai báo, chúng ta không thể lưu số phần tử vượt quá kích thước này.
* Khai báo mảng : data\_type array\_name[array\_size];
* Khởi tạo mảng trong c++:

marks[0] = 80; // khoi tao mang

marks[1] = 60;

marks[2] = 70;

marks[3] = 85;

marks[4] = 75;

ví dụ :

#include <iostream>

using namespace std;

 int main() {

    int i = 0;

    int marks[5];  // khai bao mang

    marks[0] = 80; // khoi tao mang

    marks[1] = 60;

    marks[2] = 70;

    marks[3] = 85;

    marks[4] = 75;

    //duyet mang

    for (i = 0; i < 5; i++) {

        cout << marks[i] << endl;

    }

    return 0;

}

* Khai báo khởi tạo với khởi tạo mảng nặc danh trong c++: int marks[5] = {20, 30, 40, 50, 60};

**5.Mảng hai chiều trong c++**

* Khai báo mảng 2 chiều:data\_type array\_name[size1][size2];
* Khởi tạo mảng 2 chiều:
* Khởi tạo mảng 2 chieu bằng mảng nặc danh:

#include <iostream>

using namespace std;

int main() {

    int i = 0;

    int j = 0;

    // khai bao mang 2 chieu bang mang nac danh

    int ma\_tran[4][3] = {{ 1, 2, 3 }, {2, 3, 4}, {3, 4, 5}, {4, 5, 6}};

    //duyet mang

    for (i = 0; i < 4; i++) {

     for (j = 0; j < 3; j++) {

            cout << ma\_tran[i][j] << " ";

        }

        cout << endl;

    }

    return 0;

}

* Khởi tạo mảng 2 chiều bằng cách nhập data từ bàn phím:

#include <iostream>

using namespace std;

int main() {

 int i, j;

    // khai bao mang 2 chieu bang mang nac danh

    int ma\_tran[4][3];

    //nhap mang

    cout << "Nhap mang: \n";

    for (int i = 0; i < 4; i++) {

     for (int j = 0; j < 3; j++) {

      cout << "Nhap a[" << i << "][" << j << "] = ";

      cin >> ma\_tran[i][j];

        }

        printf("\n");

    }

    //duyet mang

    cout << "Ket qua: \n";

    for (int i = 0; i < 4; i++) {

     for (int j = 0; j < 3; j++) {

            cout << ma\_tran[i][j] << " ";

        }

        cout << "\n";

    }

    return 0;

}

**5.Con trỏ trong c++**

* Một con trỏ là một biến mà trong đó giá trị của nó là địa chỉ của biến khác : kieu\_du\_lieu \*ten\_bien; Trong đó:
  + kiểu dữ liệu cơ bản con trỏ, nó là kiểu hợp lệ trong ngôn ngữ C++
  + **ten\_bien**là tên giá trị của con trỏ.
  + Phần ký tự \* sử dụng trong khai báo con trỏ giống như việc bạn sử dụng cho phép nhân.

Mặc dù vậy, trong khai báo này, ký tự \* được thiết kế để sử dụng các biến của con trỏ.

* Cách sử dụng con trỏ trong c++ :
* Dùng con trỏ để lưu địa chỉ của biến để thao tác: Bản thân con trỏ sẽ được trỏ vào địa chỉ của một biến có cùng kiểu dữ liệu với nó.Cú pháp:

<ten\_con\_tro> = &<ten\_bien>;

* Lấy giá trị của biến do con trỏ trỏ đến để thao tác: Phép lấy giá trị của biến do con trỏ trỏ đến được thực hiện bằng cách gọi tên:

\*<ten\_con\_tro>;

* Ví dụ :

#include <iostream>

using namespace std;

int main () {

   int  x = 20, y;    // khai bao bien binh thuong

   int  \*ip;          // bien con tro

   ip = &y;

   \*ip = x;           // dia chi luu tru cua bien con tro

   // in dia chi luu tru cua bien con tro ip

   cout << "Dia chi luu tru cua bien con tro ip: ";

   cout << ip << endl;

   // gia tri cua dia chi o nho cua con tro

   cout << "Gia tri cua bien \*ip: ";

   cout << \*ip << endl;

   // gia tri cua bien x

   cout << "Gia tri cua bien x: ";

   cout << x << endl;

   // gia tri cua bien y

   cout << "Gia tri cua bien y: ";

   cout << y << endl;

   return 0;

}

* Phép gán giữa các con trỏ : Các con trỏ cùng kiểu có thể gán cho nhau thông qua phép gán và lấy địa chỉ con trỏ.Cú pháp :

<ten\_con\_tro\_1> = <ten\_con\_tro\_2>

Trong phép gán giữa các con trỏ, bắt buộc phải dùng phép lấy **địa chỉ của biến** do con trỏ trỏ tới (không có dấu “\*” trong tên con trỏ) mà không được dùng phép lấy **giá trị của biến** do con trỏ trỏ tới.

Hai con trỏ phải cùng kiểu. Trong trường hợp hai con trỏ khác kiểu, phải sử dụng các phương thức ép kiểu tương tự như trong phép gán các biến thông thường có kiểu khác nhau.

* Chi tiết về con trỏ :

|  |  |
| --- | --- |
| **Khái niệm** | **Mô tả** |
| * [Con trỏ Null trong C++](https://viettuts.vn/lap-trinh-cpp/con-tro-null-trong-cpp) | * Ngôn ngữ lập trình C++ hỗ trợ con trỏ null, là một hằng số với giá trị là 0 được định nghĩa trong một số thư viện chuẩn. |
| * [Con trỏ số học trong C++](https://viettuts.vn/lap-trinh-cpp/con-tro-so-hoc-trong-cpp) | * Có 4 toán tử số học mà có thể được sử dụng trên các con trỏ là: ++, --, +, - |
| * [Con trỏ và mảng trong C++](https://viettuts.vn/lap-trinh-cpp/con-tro-va-mang-trong-cpp) | * Mối liên hệ giữa con trỏ và mảng trong C++. |
| * [Mảng con trỏ trong C++](https://viettuts.vn/lap-trinh-cpp/mang-con-tro-trong-cpp) | * Bạn có thể định nghĩa các mảng để lưu trữ các con trỏ. |
| * [Con trỏ tới con trỏ trong C++](https://viettuts.vn/lap-trinh-cpp/con-tro-tro-toi-con-tro-trong-cpp) | * Ngôn ngữ lập trình C++ cho phép bạn trỏ tới một con trỏ ... |
| * [Truyền con trỏ tới hàm trong C++](https://viettuts.vn/lap-trinh-cpp/tuyen-con-tro-toi-ham-trong-cpp) | * Truyền một tham số bởi tham chiếu hoặc địa chỉ: cả hai cho các tham số được truyền khả năng có thể được thay đổi trong hàm gọi bởi hàm được gọi. |
| * [Trả về con trỏ từ hàm trong C++](https://viettuts.vn/lap-trinh-cpp/tra-ve-con-tro-tu-ham-trong-cpp) | * Ngôn ngữ lập trình C++ cho phép một hàm trả về một con trỏ tới biến địa phương (local), biến tĩnh và cũng như bộ nhớ được cấp phát động. |

6. **Con trỏ NULL trong C++**