VÕ TIẾN
Thảo luận kiến thức CNTT trường BK về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

Kỹ Thuật Lập Trình (Cơ bản và nâng cao C++)

KTLT1 - HK242

TASK 4 POINT

Thảo luận kiến thức CNTT trường BK
về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

TP. HỒ CHÍ MINH, THÁNG 2/2025

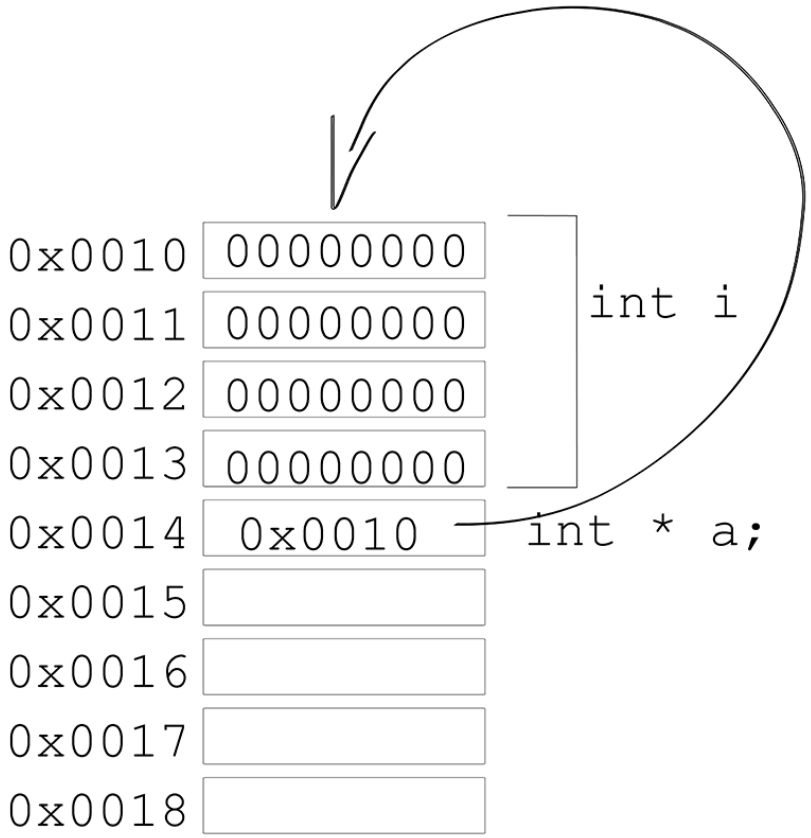
Mục lục

1 Định nghĩa con trỏ

Con trỏ (pointer) là một biến đặc biệt trong C++ dùng để lưu trữ địa chỉ của một biến khác. Thay vì lưu trữ giá trị thông thường, con trỏ lưu địa chỉ ô nhớ của một biến trong bộ nhớ.

Pointers and references

```
int i = 0;
int * p = &i;
```



Phân tích bộ nhớ

1. **Biến i được khai báo và khởi tạo bằng 0**
   - **i** là một biến kiểu **int**, được lưu trữ trong vùng nhớ Stack.
   - Địa chỉ giả định của **i** là **0x0010**.
   - Nội dung tại địa chỉ **0x0010** là **00000000** (giá trị của **i** là 0).
2. **Con trỏ p trỏ đến i**
   - **p** là một con trỏ kiểu **int***, nó lưu địa chỉ của **i**.
   - Biến **p** được lưu ở một địa chỉ khác, giả sử là **0x0014**.
   - Nội dung tại **0x0014** là **0x0010**, nghĩa là **p** đang trỏ đến **i**.

2 Con trỏ cơ bản

1. **Khai báo con trỏ**

   Cú pháp khai báo một con trỏ trong C++:

   ```
   int *p;    // Con trỏ kiểu int
   double *q; // Con trỏ kiểu double
   char *c;   // Con trỏ kiểu char
   ```

   Dấu '*' cho biết 'p' là một con trỏ. 'p' có thể lưu địa chỉ của một biến kiểu 'int'.

2. **Gán địa chỉ cho con trỏ**

   ```
   int a = 10;
   int *p = &a;  // p lưu địa chỉ của biến a
   ```

   - '&a' lấy địa chỉ của 'a' và gán cho 'p'.
   - 'p' bây giờ trỏ đến 'a'.

3. **Truy cập giá trị thông qua con trỏ**

   Ta có thể dùng toán tử '*' (dereference) để lấy giá trị tại địa chỉ mà con trỏ đang trỏ tới.

   ```
   int a = 10;
   int *p = &a;

   cout << "Gia tri cua a: " << a << endl;       // 10
   cout << "Gia tri cua *p: " << *p << endl;     // 10
   cout << "Dia chi cua a: " << &a << endl;      // Địa chỉ của a
   cout << "Gia tri cua p (dia chi ma p tro den): " << p << endl;
   ```

4. **Con trỏ NULL**

   Nếu con trỏ không trỏ đến địa chỉ hợp lệ, ta có thể gán nó bằng 'nullptr' (trong C++11 trở đi) hoặc 'NULL':

   ```
   int *p = nullptr;  // Con trỏ chưa trỏ đến đâu cả
   ```

3 Cấp phát động

Cấp phát động (Dynamic Memory Allocation) là cơ chế cấp phát bộ nhớ trong thời gian chạy (runtime) thay vì khi biên dịch (compile-time). Bộ nhớ cấp phát động được lấy từ vùng nhớ heap và cần được giải phóng thủ công.

1. **Toán tử `new` và `delete`**

   - Dùng `new` để cấp phát động một biến hoặc mảng.
   - Dùng `delete` để giải phóng bộ nhớ đã cấp phát.

   ```
   // Cấp phát động một số nguyên
   int *p = new int(10);
   cout << "Gia tri cua p: " << *p << endl;  // In ra 10
   delete p; // Giải phóng bộ nhớ
   ```

2. **Cấp phát động mảng**

   Để cấp phát động một mảng, ta dùng `new[]` và giải phóng bằng `delete[]`:

   ```
   // Cấp phát động một mảng 5 phần tử
   int *arr = new int[5]{1, 2, 3, 4, 5};

   // Truy cập mảng
   for (int i = 0; i < 5; i++) {
   ```

```
    cout << arr[i] << " ";
}
cout << endl;

// Giải phóng bộ nhớ
delete[] arr;
```

3. **Cấp phát động với con trỏ đôi (Double Pointer)**

   Khi làm việc với mảng 2D động, ta cần sử dụng con trỏ đôi:

```
// Cấp phát động mảng 2D kích thước 3x3
int **matrix = new int*[3];
for (int i = 0; i < 3; i++) {
    matrix[i] = new int[3];
}

// Gán giá trị và hiển thị
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        matrix[i][j] = i * 3 + j;
        cout << matrix[i][j] << " ";
    }
    cout << endl;
}

// Giải phóng bộ nhớ
for (int i = 0; i < 3; i++) {
    delete[] matrix[i];
}
delete[] matrix;
```

4. **Cấp phát động struct**

   Chúng ta sử dụng toán tử new để cấp phát động một struct và delete để giải phóng bộ nhớ:

```
// Cấp phát động một struct
Student* p = new Student;

// Gán giá trị cho struct
p->name = "John";
p->age = 20;

// In thông tin
cout << "Name: " << p->name << ", Age: " << p->age << endl;

// Giải phóng bộ nhớ
delete p;
```

5. **Các lỗi thường gặp khi cấp phát động**

   - Rò rỉ bộ nhớ (Memory Leak): Không giải phóng bộ nhớ đã cấp phát động.

```
void memoryLeak() {
    int* p = new int(10); // Cấp phát nhưng không giải phóng
}
```

   - Sử dụng delete thay vì delete[]

```
int* arr = new int[5];
delete arr; // LỖI: Chỉ giải phóng phần tử đầu tiên, gây rò rỉ bộ nhớ
```

   - Truy cập vùng nhớ đã giải phóng (Dangling Pointer)

```
int* p = new int(10);
delete p;
cout << *p; // LỖI: Truy cập vùng nhớ đã giải phóng
```

- Truy cập vùng nhớ chưa cấp phát (Uninitialized Pointer)

```
int* p;
*p = 5; // LỖI: p trỏ đến vùng nhớ không xác định
```

- Lỗi truy cập ngoài phạm vi (Out-of-Bounds Access)

```
int* arr = new int[5];
arr[10] = 100; // LỖI: Truy cập ngoài phạm vi mảng
```

- Quên cấp phát vùng nhớ trước khi sử dụng (Null Pointer Dereference)

```
int* p = nullptr;
cout << *p; // LỖI: Dereference con trỏ null
```

- Truy cập vùng nhớ bị ghi đè (Double Delete)

```
int* p = new int(10);
delete p;
delete p; // LỖI: Giải phóng hai lần
```

4 Array và Con trỏ

Mối quan hệ giữa Array và Con trỏ

```
int arr[] = {10, 20, 30};
int *p = arr; // p trỏ đến phần tử đầu tiên của arr
```

- Một mảng trong C++ thực chất là một địa chỉ của phần tử đầu tiên.
- Khi khai báo một mảng, tên mảng chính là con trỏ trỏ đến phần tử đầu tiên.
- arr bản chất là một hằng con trỏ (const int *), nên không thể thay đổi địa chỉ của nó.

Truy cập phần tử mảng bằng con trỏ

```
int arr[] = {1, 2, 3, 4, 5};
int *p = arr;

for (int i = 0; i < 5; i++) {
    cout << *(p + i) << " "; // Truy cập phần tử bằng con trỏ
}
```

Toán tử ++ với con trỏ Toán tử ++ làm tăng địa chỉ mà con trỏ trỏ đến, tức là nó sẽ trỏ đến phần tử tiếp theo trong bộ nhớ.

```
int arr[] = {10, 20, 30};
int *p = arr; // p trỏ đến phần tử đầu tiên

cout << *p << endl; // In ra 10
p++; // Di chuyển con trỏ đến phần tử tiếp theo
cout << *p << endl; // In ra 20
```

5 Con trỏ đến con trỏ

Con trỏ đến con trỏ (pointer to pointer) là một biến lưu địa chỉ của một con trỏ khác. Nó cho phép thao tác gián tiếp lên con trỏ mà nó trỏ tới.

```
int a = 10;
int* p = &a;   // p trỏ đến a
int** pp = &p; // pp trỏ đến p
```

- p là con trỏ trỏ đến a, nghĩa là p lưu địa chỉ của a.
- pp là con trỏ đến con trỏ (pointer to pointer), nghĩa là pp lưu địa chỉ của p.

Biến	Giá trị	Địa chỉ (giả sử)
a	10	0x1000
p	0x1000	0x2000
pp	0x2000	0x3000

Bảng 1: *Bảng giá trị và địa chỉ của các biến*

Cách truy cập	Ý nghĩa	Giá trị
a	Giá trị gốc của biến a	10
*p	Giá trị mà p trỏ đến	10
*pp	Giá trị mà pp trỏ đến (p)	Địa chỉ a (0x1000)
**pp	Giá trị của a qua pp	10

Bảng 2: *Truy cập giá trị thông qua con trỏ*