Bài 3: Pointer

# Khái niệm và các loại Pointer

Trong ngôn ngữ lập trình C, con trỏ (pointer) là một biến chứa địa chỉ bộ nhớ của một biến khác. Việc sử dụng con trỏ giúp chúng ta thực hiện các thao tác trên bộ nhớ một cách linh hoạt hơn. Dưới đây là một số khái niệm cơ bản về con trỏ trong C:

Cách khai báo:

| **int** \*ptr; // con trỏ đến kiểu int **char** \*ptr\_char; // con trỏ đến kiểu char **float** \*ptr\_float; // con trỏ đến kiểu float |
| --- |

Lấy địa chỉ của một biến:

| **int** x = 10; **int** \*ptr\_x = &x; // ptr\_x giờ đây chứa địa chỉ của x |
| --- |

Sử dụng con trỏ để truy cập giá trị:

| **int** y = \*ptr\_x; // y sẽ bằng giá trị của x |
| --- |

Kích thước của con trỏ sẽ phụ thuộc kiến trúc máy tính và trình biên dịch. Ta có thể dùng sizeof() để kiểm tra kích thước của con trỏ:

| #**include** <stdio.h>  **int** **main**() {  **int** \*ptr;  printf("Size of pointer: %lu bytes\n", **sizeof**(ptr));  **return** 0; } |
| --- |

Ví dụ:

| #**include** <stdio.h> **void** **swap**(**int** \*a, **int** \*b) {  **int** tmp = \*a;  \*a = \*b;  \*b = tmp;  }  **int** **main**() {  **int** a = 10, b = 20;  swap(&a, &b);   printf("value a is: %d\n", a);  printf("value b is: %d\n", b);   **return** 0; } |
| --- |

Các loại pointer:

Void pointer:

* Void pointer thường dùng để trỏ để tới bất kỳ địa chỉ nào mà không cần biết tới kiểu dữ liệu của giá trị tại địa chỉ đó.

| **void** \*ptr\_void; |
| --- |

Ví dụ:

| #**include** <stdio.h> #**include** <stdlib.h>  **int** **main**() {    **int** value = 5;  **double** test = 15.7;  **char** letter = 'A';    **void** \*ptr = &value;  printf("value is: %d\n", \*(**int**\*)(ptr));   ptr = &test;  printf("value is: %f\n", \*(**double**\*)(ptr));   ptr = &letter;  printf("value is: %c\n", \*(**char**\*)(ptr));    **return** 0; } |
| --- |

Pointer to Constant:

Định nghĩa một con trỏ không thể thay đổi giá trị tại địa chỉ mà nó trỏ đến thông qua dereference nhưng giá trị tại địa chỉ đó có thể thay đổi.

Ví dụ:

| #**include** <stdio.h> #**include** <stdlib.h>  **int** **main**() {    **int** value = 5;  **int** **const** \*ptr\_const = &value;   //\*ptr\_const = 7; // wrong  //ptr\_const++; // right    printf("value: %d\n", \*ptr\_const);   value = 9;  printf("value: %d\n", \*ptr\_const);   **return** 0; } |
| --- |

Constant Pointer:

Định nghĩa một con trỏ mà giá trị nó trỏ đến (địa chỉ ) không thể thay đổi. Tức là khi con trỏ này được khởi tạo thì nó sẽ không thể trỏ tới địa chỉ khác.

| #**include** <stdio.h> #**include** <stdlib.h>   **int** **main**() {    **int** value = 5;  **int** test = 15;  **int** \***const** const\_ptr = &value;   printf("value: %d\n", \*const\_ptr);   \*const\_ptr = 7;  printf("value: %d\n", \*const\_ptr);   //const\_ptr = &test; // wrong    **return** 0; } |
| --- |

Function pointer:

Pointer to function (con trỏ hàm) là một biến mà giữ địa chỉ của một hàm. Có nghĩa là, nó trỏ đến vùng nhớ trong bộ nhớ chứa mã máy của hàm được định nghĩa trong chương trình.

Trong ngôn ngữ lập trình C, con trỏ hàm cho phép bạn truyền một hàm như là một đối số cho một hàm khác, lưu trữ địa chỉ của hàm trong một cấu trúc dữ liệu, hoặc thậm chí truyền hàm như một giá trị trả về từ một hàm khác.

Ví dụ:

| #**include** <stdio.h>  // Hàm mẫu 1 **void** **greetEnglish**() {  printf("Hello!\n"); }  // Hàm mẫu 2 **void** **greetFrench**() {  printf("Bonjour!\n"); }  **int** **main**() {  // Khai báo con trỏ hàm  **void** (\*ptrToGreet)();   // Gán địa chỉ của hàm greetEnglish cho con trỏ hàm  ptrToGreet = greetEnglish;   // Gọi hàm thông qua con trỏ hàm  (\*ptrToGreet)(); // In ra: Hello!   // Gán địa chỉ của hàm greetFrench cho con trỏ hàm  ptrToGreet = greetFrench;   // Gọi hàm thông qua con trỏ hàm  (\*ptrToGreet)(); // In ra: Bonjour!   **return** 0; } |
| --- |

Trong ví dụ này, ptrToGreet là một con trỏ hàm có thể trỏ đến các hàm greetEnglish và greetFrench. Việc này giúp linh hoạt trong việc chọn và sử dụng hàm tương ứng tại thời điểm chạy.

Ví dụ 2:

| #**include** <stdio.h>  **void** **sum**(**int** a, **int** b) {  printf("Sum of %d and %d is: %d\n",a,b, a+b); }  **void** **subtract**(**int** a, **int** b) {  printf("Subtract of %d by %d is: %d \n",a,b, a-b);   }  **void** **multiple**(**int** a, **int** b) {  printf("Multiple of %d and %d is: %d \n",a,b, a\*b );   }  **void** **divide**(**int** a, **int** b) {  **if** (b == 0)  {  printf("Mau so phai khac 0\n");  **return**;  }    printf("%d divided by %d is: %f \n",a,b, (**double**)a / (**double**)b);   }  **void** **calculator**(**void** (\*ptr)(**int** a, **int** b), **int** a, **int** b) {  printf("Program calculate: \n");  ptr(a,b); }  **int** **main**() {  **void** (\*ptrToFunc)(**int**,**int**);  ptrToFunc = &divide;   calculator(ptrToFunc,5,2);   **return** 0; } |
| --- |

Trong ví dụ này, ptrToFunc là một con trỏ hàm trỏ đến các hàm sum, subtract, multiple, divide. Hàm calculator với 3 tham số truyền vào là: con trỏ hàm, a, b, và sẽ call function mà con trỏ đang trỏ tới và truyền vào 2 tham số a và b.

Pointer to pointer:

Con trỏ đến con trỏ (Pointer to Pointer) là một kiểu dữ liệu trong ngôn ngữ lập trình cho phép bạn lưu trữ địa chỉ của một con trỏ. Con trỏ đến con trỏ cung cấp một cấp bậc trỏ mới, cho phép bạn thay đổi giá trị của con trỏ gốc. Cấp bậc này có thể hữu ích trong nhiều tình huống, đặc biệt là khi bạn làm việc với các hàm cần thay đổi giá trị của con trỏ.

Ví dụ:

| #**include** <stdio.h>  **int** **main**() {  **int** value = 42;  **int** \*ptr1 = &value; // Con trỏ thường trỏ đến một biến   **int** \*\*ptr2 = &ptr1; // Con trỏ đến con trỏ   printf("Value: %d\n", \*\*ptr2);   **return** 0; } |
| --- |

Trong ví dụ này:

* ptr1 là một con trỏ thường trỏ đến biến value.
* ptr2 là một con trỏ đến con trỏ, trỏ đến địa chỉ của ptr1.
* Khi sử dụng \*\*ptr2, chúng ta có thể truy cập giá trị của biến value.

Ứng dụng phổ biến trong cấp phát động mảng hai chiều:

| #**include** <stdio.h> #**include** <stdlib.h>  **void** **allocate2DArray**(**int** \*\*\*arr, **int** rows, **int** columns) {  // Cấp phát bộ nhớ cho mảng con trỏ  \*arr = (**int**\*\*)malloc(rows \* **sizeof**(**int**\*));   // Cấp phát bộ nhớ cho mỗi hàng  **for** (**int** i = 0; i < rows; ++i) {  (\*arr)[i] = (**int**\*)malloc(columns \* **sizeof**(**int**));  } }  **void** **initialize2DArray**(**int** \*\*\*arr, **int** rows, **int** columns) {  // Gán giá trị cho mảng 2D  **for** (**int** i = 0; i < rows; ++i) {  **for** (**int** j = 0; j < columns; ++j) {  (\*arr)[i][j] = i \* columns + j + 1;  }  } }  **void** **print2DArray**(**int** \*\*arr, **int** rows, **int** columns) {  // In ra giá trị của mảng 2D  **for** (**int** i = 0; i < rows; ++i) {  **for** (**int** j = 0; j < columns; ++j) {  printf("%4d ", arr[i][j]);  }  printf("\n");  } }  **void** **free2DArray**(**int** \*\*\*arr, **int** rows) {  // Giải phóng bộ nhớ của mảng 2D  **for** (**int** i = 0; i < rows; ++i) {  free((\*arr)[i]);  }  free(\*arr); }  **int** **main**() {  **int** rows = 3, columns = 4;  **int** \*\*matrix;   // Cấp phát bộ nhớ cho mảng 2D  allocate2DArray(&matrix, rows, columns);   // Khởi tạo giá trị cho mảng 2D  initialize2DArray(&matrix, rows, columns);   // In ra giá trị của mảng 2D  printf("2D Array:\n");  print2DArray(matrix, rows, columns);   // Giải phóng bộ nhớ của mảng 2D  free2DArray(&matrix, rows);   **return** 0; } |
| --- |

NULL pointer:

Null Pointer là một con trỏ không trỏ đến bất kỳ đối tượng hoặc vùng nhớ cụ thể nào. Trong ngôn ngữ lập trình C, một con trỏ có thể được gán giá trị NULL để biểu diễn trạng thái null.

Ví dụ:

| #**include** <stdio.h>  **int** **main**() {  **int** \*ptr = NULL; // Gán giá trị NULL cho con trỏ   **if** (ptr == NULL) {  printf("Pointer is NULL\n");  } **else** {  printf("Pointer is not NULL\n");  }   **return** 0; } |
| --- |

Trong ví dụ này:

* Con trỏ ptr được khai báo và được gán giá trị NULL.
* Một điều kiện kiểm tra xem con trỏ có trỏ đến một đối tượng nào đó hay không.
* Nếu con trỏ bằng NULL, chương trình in ra "Pointer is NULL", ngược lại nếu con trỏ không bằng NULL, chương trình in ra "Pointer is not NULL".

Sử dụng null pointer thường hữu ích để kiểm tra xem một con trỏ đã được khởi tạo và có trỏ đến một vùng nhớ hợp lệ chưa. Tránh dereferencing (sử dụng giá trị mà con trỏ trỏ đến) một null pointer là quan trọng để tránh lỗi chương trình.