Bài 4: Pointer

Nguyễn Hoàng Anh

Pointer

Trong ngôn ngữ lập trình C, con trỏ (pointer) là một biến chứa địa chỉ bộ nhớ của một đối tượng khác (biến, mảng, hàm). 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:

Pointer

```
int test;
```

Address: 0x01
Value: 0

```
int *ptr = &test;
ptr; // 0x01
*ptr; // 0
```

Address: 0xf1
Value: 0x01

Pointer

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 và truy cập giá trị

```
int x = 10;
int *ptr_x = &x;  // ptr_x giờ đây chứa địa chỉ của x
int y = *ptr_x;   // y sẽ bằng giá trị của x (dereference)
```

Pointer

Kích thước của con trỏ phụ thuộc vào kiến trúc máy tính và trình biên dịch hoặc kiến trúc vi xử lý.

```c
#include <stdio.h>

int main() {
    int *ptr;
    printf("Size of pointer: %d bytes\n", sizeof(ptr));
    return 0;
}
```

Pointer

Ứng dụng:

```c
#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;
}
```

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.

Cú pháp:

```
void *ptr_void;
```

Void Pointer

```c
#include <stdio.h>
#include <stdlib.h>

int sum(int a, int b)
{
    return a+b;
}

int main() {

    char array[] = "Hello";
    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));
```

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.

```
<return_type> (* func_pointer)(<data_type_1>, <data_type_2>);
```

Function Pointer

```c
#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;
```

Function Pointer

```c
#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)
```

Function Pointer

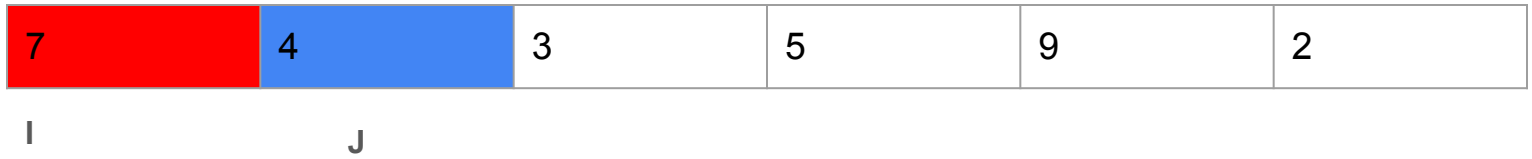
```c
#include <stdio.h>
#include <string.h>

void bubbleSort(int arr[], int n) {
    int i, j, temp;
    for (i = 0; i < n-1; i++)
        for (j = i+1; j < n; j++)
            if (arr[i] > arr[j]) {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
```

Example

Input

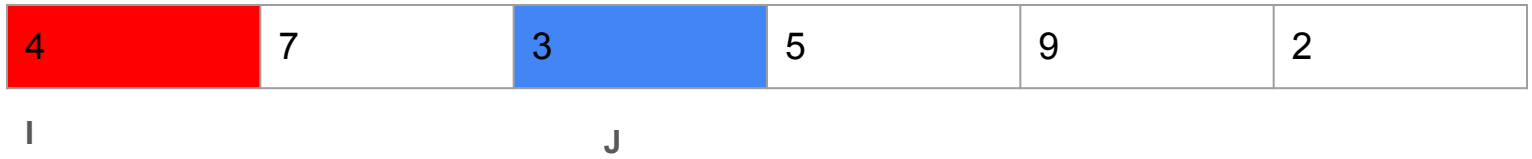


Output

4	7	3	5	9	2

Example

Input



Output

3	7	4	5	9	2

Example

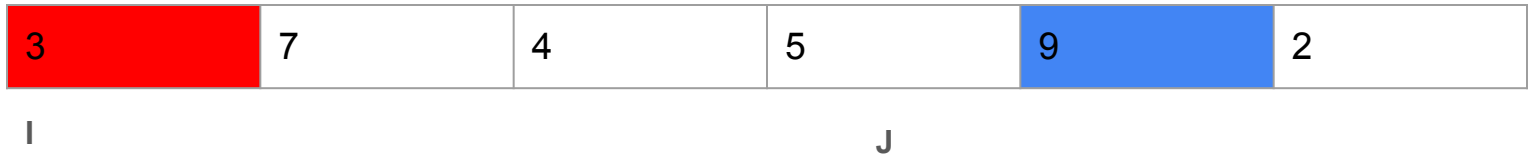
Input

Output

3	7	4	5	9	2

Example

Input

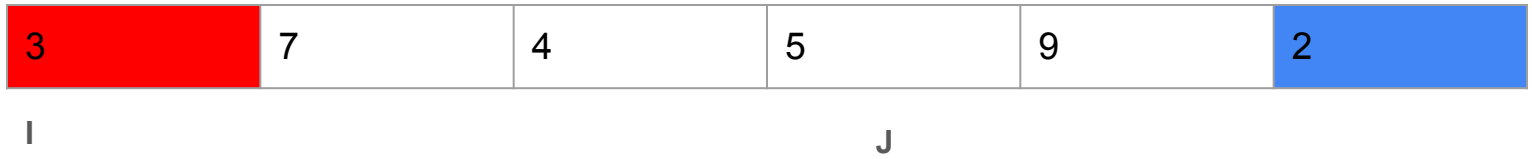


Output

3	7	4	5	9	2

Example

Input



Output

2	7	4	5	9	3

Example

Input

Output

2	4	7	5	9	3

Example

Input

Output

2	4	7	5	9	3

Function Pointer

```c
#include <stdio.h>
#include <string.h>

typedef struct {
   char ten[50];
   float diemTrungBinh;
   int id;
} SinhVien;

int stringCompare(const char *str1, const char *str2) {
   while (*str1 && (*str1 == *str2)) {
       str1++;
       str2++;
   }
   return *(const unsigned char*)str1 - *(const unsigned char*)str2;
}
```

Function Pointer

```c
#include <stdio.h>

typedef struct {
    void (*start)(int gpio);
    void (*stop)(int gpio);
    void (*changeSpeed)(int gpio, int speed);
} MotorController;

typedef int PIN;

// Các hàm chung
void startMotor(PIN pin) {
    printf("Start motor at PIN %d\n", pin);
}

void stopMotor(PIN pin) {
    printf("Stop motor at PIN %d\n", pin);
}
```

Pointer to Constant

Là cách 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.

Cú pháp:

```
int const *ptr_const;
const int *ptr_const;
```

Pointer to Constant

```c
#include <stdio.h>
#include <stdlib.h>

int main() {

    int value = 5;
    int test = 8;
    int const *ptr_const = &value;

    //*ptr_const = 7; // wrong
    //ptr_const = &test; // 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.

Cú pháp:

```
int *const const_ptr = &value;
```

Constant Pointer

```c
#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;
}
```

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.

Pointer

```c
int test = 5;
```

Address: 0x01
Value: 5

```c
int *ptr = &test;
```

Address: 0xf1
Value: 0x01

```c
int **ptp = &ptr;
```

Address: 0xef
Value: 0xf1

Pointer to Pointer

```c
#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

    /*
        **ptr2 = &ptr1
        ptr2 = &ptr1;
        *ptr2 = ptr1 = &value;
        **ptr2 = *ptr1 = value
    */
```

Pointer to Pointer

Ứng dụng:
kiểu dữ liệu json
Cấu trúc dữ liệu list

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, C++ (NULL, nullptr).

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.

NULL Pointer

```c
#include <stdio.h>

int main() {
    int *ptr = NULL;  // Gán giá trị NULL cho con trỏ 0x0000000

int a;

    if (ptr == NULL) {
        printf("Pointer is NULL\n");
    } else {
        printf("Pointer is not NULL\n");
    }

    int score_game = 5;
    if (ptr == NULL)
    {
        ptr = &score_game;
```