ALX LESSON

0x0F C - Function pointers

C - Programming

TABLE OF CONTENTS

01

OVERVIEW topics

Topics

C Programming

Topics

What are function pointers and how to use them

What does a function pointer exactly hold

Where does a function pointer point to in the virtual memory

Slides On Telegram

https://t.me/alx_2023

C Programming

Topics

02

Learning Objectives

What are function pointers and how to use them

In C, like normal data pointers (int *, char *, etc), we can have pointers to functions.

Function pointers are variables that store the memory address of a function. They allow you to indirectly call a function or pass a function as an argument to another function.

```
/* function returning pointer to int */
int *func(int a, float b);          or          int* func(int a, float b);

/* pointer to function returning int */
int (*func)(int a, float b);
```

What does a function pointer exactly hold

A function pointer holds the memory address of a function, which allows you to indirectly call the function. The memory address is essentially a pointer to the beginning of the function's machine code in memory. When you call a function through a function pointer, the program looks up the memory address stored in the pointer variable, and then jumps to that address to execute the code in the function.

Where does a function pointer point to in the virtual memory

In terms of virtual memory, the function pointer points to the location in the program's address space where the function's machine code is loaded. This location can vary depending on how the program was compiled and loaded into memory.

When a C program is compiled, the compiler generates machine code for each function, and the linker puts all the machine code together into an executable file. When the program is loaded into memory, the operating system maps the machine code of each function into the program's address space at a specific virtual memory address.

Where does a function pointer point to in the virtual memory

Function returning pointer to int

```c
#include <stdio.h>

// Function returning pointer to int
int *sum(int a, int b) {
    static int result; // Static variable to hold the result

    result = a + b; // Perform the addition and store the result in the variable

    return &result; // Return a pointer to the result variable
}

int main() {
    int a = 5;
    int b = 10;

    int *result = sum(a, b); // Call the function and get a pointer to the result

    printf("Result: %d\n", *result); // Print the result

    return 0;
}
```

e1

```c
#include <stdio.h>
// A normal function with an int parameter
// and void return type
void fun(int a)
{
    printf("Value of a is %d\n", a);
}

int main()
{
    // fun_ptr is a pointer to function fun()
    void (*fun_ptr)(int) = &fun;

    /* The above line is equivalent of following two
       void (*fun_ptr)(int);
       fun_ptr = &fun;
    */

    // Invoking fun() using fun_ptr
    (*fun_ptr)(10);

    return 0;
}
```

e2

```c
#include <stdio.h>
// A normal function with an int parameter
// and void return type
void fun(int a)
{
    printf("Value of a is %d\n", a);
}

int main()
{
    void (*fun_ptr)(int) = fun;  // & removed

    fun_ptr(10);  // * removed

    return 0;
}
```

Pointer to function returning int

```c
#include <stdio.h>

// Pointer to function returning int
int (*sum_ptr)(int a, int b);

// Function that can be pointed to by sum_ptr
int add(int a, int b) {
    return a + b;
}

int main() {
    int a = 5;
    int b = 10;

    sum_ptr = &add; // Set sum_ptr to point to the add function

    int result = (*sum_ptr)(a, b); // Call the function pointed to by sum_ptr

    printf("Result: %d\n", result); // Print the result

    return 0;
}
```

e3

```c
#include <stdio.h>
void add(int a, int b)
{
    printf("Addition is %d\n", a+b);
}
void subtract(int a, int b)
{
    printf("Subtraction is %d\n", a-b);
}
void multiply(int a, int b)
{
    printf("Multiplication is %d\n", a*b);
}

int main()
{
    // fun_ptr_arr is an array of function pointers
    void (*fun_ptr_arr[])(int, int) = {add, subtract, multiply};
    unsigned int ch, a = 15, b = 10;

    printf("Enter Choice: 0 for add, 1 for subtract and 2 "
           "for multiply\n");
    scanf("%d", &ch);

    if (ch > 2) return 0;

    (*fun_ptr_arr[ch])(a, b);

    return 0;
}
```

e4

```c
// A simple C program to show function pointers as parameter
#include <stdio.h>

// Two simple functions
void fun1() { printf("Fun1\n"); }
void fun2() { printf("Fun2\n"); }

// A function that receives a simple function
// as parameter and calls the function
void wrapper(void (*fun)())
{
    fun();
}

int main()
{
    wrapper(fun1);
    wrapper(fun2);
    return 0;
}
```

e5

```c
#include <stdio.h>

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

int subtract(int a, int b) {
    return a - b;
}

// A function that takes two integers and a pointer to a function,
// and applies the function to the integers
void apply(int a, int b, int (*operation)(int, int)) {
    int result = operation(a, b);
    printf("Result: %d\n", result);
}

int main() {
    int a = 5;
    int b = 3;

    int (*add_ptr)(int, int); // Declare a pointer to the add function
    add_ptr = &add; // Set the pointer to point to the add function

    int (*subtract_ptr)(int, int); // Declare a pointer to the subtract function
    subtract_ptr = &subtract; // Set the pointer to point to the subtract function

    apply(a, b, add_ptr); // Call apply function with add_ptr
    apply(a, b, subtract_ptr); // Call apply function with subtract_ptr

    return 0;
}
```

04

Hands on lab Practice


Have a Question
Leave a Comment!

Share

To let the others know more

Subscribe

To stay updated with latest videos

Thanks