

# Function Pointer

## Pointers

Pointers are the data types that can be used to store the address of some data stored in a computer's memory. Pointers are mostly used as a data type that would store the address of other variables.

Pointers can point to data/functions where data could be stored as a constant or a variable. We can also use pointers to dereference and get the value at whatever address the pointer is pointing at.

```
// <variable_type> *<name>
// example:
int data; int *pointer_to_integer = &data;
```

---

## Function Pointers

Function pointers are used to store the address of functions. We need function pointers to make "callbacks", but let us understand the basic syntax first.

### Function Pointer Syntax

1. If the function return type is void

```
void (*func_pointer)(void);
```

Let us re-read the syntax, `*func_pointer` is the pointer to a function. `void` is the return type of that function, and finally `void` is the argument type of that function. The parenthesis around the function pointer is a must otherwise it will change the meaning of the function pointer declarations.

2. If a function returns an `int` and has a `char*` as an input parameter, then the code looks like this:

```
int (*func_pointer)(char *)
```

In this example:

1. `*func_pointer` is the function pointer

2. `int` is the return type of that function
  3. `char*` is the type of argument.
- 

## Examples

### Code Example 1: Function pointers with an `int` as an argument

```
#include <stdio.h>

void function(int arg) {
    printf("Function being called and arg is: %d\n", arg);
}

int main(void) {
    void (*func_pointer)(int);

    // assign function to the function pointer
    func_pointer = &function;

    // call the function pointer
    (*func_pointer)(6);

    // Or call it like this:
    func_pointer(6);}
```

### Code Example 2: Function pointer returns and taking argument as void data type.

```
// Let us "typedef" the function pointer: void void_function(void);
typedef void (*void_function_t)(void);

void foo(void) {
    puts("Hello");
}

int main(void) {
    // assign function to the function pointer
    void_function_t func_pointer = foo;

    // call the function pointer
    func_pointer();}
```

**Code Example 3:** How to use an array of functions using function pointers.

```

/* Example 1 */
void foo(void) { puts("foo"); }
void bar(void) { puts("bar"); }
// Typedef a function with void argument, returning nothing (void)
typedef void (*void_function_t)(void);
int main(void) {
    // assign array of functions to the function pointer
    void_function_t func_pointers[] = {foo, bar};

    // call the function pointers
    func_pointers[0]();
    func_pointers[1]();
}

/* Example 2 */
/* For simplicity considering number_one > number_two */
int add(int number_one, int number_two) { return number_one+number_two; }
int sub(int number_one, int number_two) { return number_one-number_two; }
int multiply(int number_one, int number_two) { return number_one*number_two; }
int divide(int number_one, int number_two) { if(number_two !=0) return (number_one/number_two); else r
int main(void) {
    int x = 10, y = 2;
    int choice,result;

    // assign array of functions to the function pointer
    int (*function_pointer[4])(int,int) = {add, sub, multiply, divide};

    printf("Enter 0: For Addition, 1 for subtraction, 2 for multiplication, and 3 for division: ");
    scanf("%d", &choice);

    // call the required function pointer
    result = function_pointer[choice](x, y);

    printf("Result: %d\r\n", result);
    return 0;}

```

---

Revision #22

Created 4 years ago by [vidushi](#)

Updated 3 years ago by [vidushi](#)