Pointer to a Function in C

Pedram Pasandide

Pointer to a Function in C

In C, a function pointer is a variable that stores the address of a function. Function pointers allow functions to be passed as arguments, stored in arrays, or used for callbacks and dynamic dispatch.

To declare a pointer to a function, we specify the function's return type and parameter types.

```c
// Function prototype
int add(int a, int b);

// Function pointer declaration
int (*func_ptr)(int, int);
```

Here, `func_ptr` is a pointer to a function that takes two `int` arguments and returns an `int`. Take a look at the following example:

```c
#include <stdio.h>

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

int main() {
 int (*func_ptr)(int, int) = add;  // assign function address
 int result = func_ptr(3, 4);      // call through pointer
 printf("Result: %d\n", result); // Result: 7
 return 0;
}
```

`func_ptr(3, 4)` is equivalent to `(*func_ptr)(3, 4)` in C.

Function pointers can be included in `structs`, enabling behavior similar to **object-oriented programming**, where structs can have "member functions".

```c
#include <stdio.h>

typedef struct Calculator {
 int (*operation)(int, int);  // function pointer member
} Calculator;

int multiply(int a, int b) {
 return a * b;
}

int main() {
 Calculator calc;
 calc.operation = multiply;

 int result = calc.operation(5, 6);
 printf("Multiplication: %d\n", result);
 return 0;
}
```

Why Use Function Pointers?

- **Callbacks**: Passing a function to be called later (e.g., `qsort()`).
- **Plugins**: Load different functionality dynamically.
- **Abstraction**: Allow different behaviors using the same interface.
- **State Machines**: Jump to different logic based on states.

It is also useful for dispatch tables or menu-driven programs. We can have function pointer arrays:

```c
#include <stdio.h>

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

int main() {
 int (*ops[2])(int, int) = { add, sub };
 printf("Add: %d\n", ops[0](10, 5));   // Calls add
 printf("Sub: %d\n", ops[1](10, 5));   // Calls sub
```

```
  return 0;
}
```