



Function and Pointers

Learn how to pass pointers to functions.

We'll cover the following



- Passing pointers to the function
 - Basic syntax
 - Example program
 - Explanation
 - passPointer function
 - main function
 - Illustration

Passing pointers to the function#

In a previous lesson

(<https://www.educative.io/collection/page/10370001/6619096843026432/5929064813559808>), we discussed the following two ways of passing actual parameters to the formal parameters in the function:

- Pass by value
- Pass by reference

However, there is another way to pass arguments to the function that is passed by reference with a pointer parameter.

The function pointer parameter receives the address of the parameter. Then, it uses the dereference operator to access the value of the variable.

It uses the dereference operator to access the value of the variable.



Basic syntax#

The general syntax for passing a pointer to the function parameter is given below:

```
#include <stdio.h>

return_type function_name ( *number ) ;

.....

int main ( )
{
    int num = 10
    function_name ( &num ) ;
    return 0;
}
```

formal parameter

number

10

num

actual parameter

When we want to pass the pointer, we will declare a function parameter as a pointer. To declare a function parameter as a pointer, use an asterisk * before the function parameter. Then, pass the address of the variable in the actual parameter using the address-of & operator.

Example program

```
1 #include <iostream>
2
3 using namespace std;
4 // function definition
```



```

5 void passPointer(int *number) {
6     // Multiply the number by 10
7     *number = *number * 10;
8     cout << "Value of number inside the function = " << *number << endl;
9 }
10
11 int main() {
12     // Initialize variable
13     int num = 10;
14     cout << "Value of number before function call = " << num << endl;
15     // Call function
16     passPointer(&num);
17     cout << "Value of number after function call = " << num << endl;
18
19     return 0;
20 }

```



×

Output

0.94s

```

Value of number before function call = 10
Value of number inside the function = 100
Value of number after function call = 100

```

Explanation#

In the code above, we have two functions:

- passPointer function
- main function



passPointer function#



Line No. 5: The `passPointer` function receives an address of the `int` value and stores it in the `number`. Since the function is of type `void`, no value is returned.

Line No. 7: Multiplies the value that the pointer `number` is pointing to by 10 and stores the result in the location pointed by the `number`.

Line No. 8: Prints the value pointed by the `number`.

main function#

Line No. 13: Initializes a variable `num`.

Line No. 14: Prints the value of the `num` before the function call.

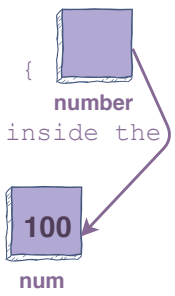
Line No. 16: Calls a function `passPointer` and passes the address of the `num` to function. The execution control is transferred to **Line No. 5**.

Line No. 17: Prints the value of the `num` after the function call.

Illustration#



```
void passValue (int *number) {  
    *number = *number *10;  
    cout << "Value of number inside the function = " << *number << endl;  
→ } Exit function passValue:  
  
int main ( ) {  
    int num =10;  
  
    cout << "Value of num before function call = " << num << endl;  
    passPointer(&num);  
  
    cout << "Value of num after function call = " << num << endl;  
  
    return0;  
}
```



Output:

Value of num before function call = 10
Value of number inside the function = 100

8 of 11

< > ▶ ↶ + []

i By default, pointers are passed by value. When we call the function, the value of the address is copied to the pointer variable. So if we change the value of the pointer inside the function, we cannot see that change outside the function body.

Quiz





Consider the function given below:



```
void passPointer(int *number) {  
    int value = 13;  
    number = &value;  
    *number = *number + 14;  
}
```

Suppose `num = 10` and we called the function `passPointer(&num)`, what would be the value of `num` after the function call?

Your Answer



A) 10

Explanation

The value of `num` will remain the same. In the `passPointer` body, we declared the new variable `value` and then updated the address stored in `number`. Now the `number` is pointing to `value`. `passPointer` is no more pointing to `num`.



B) 13



C) 24



D) 27

Submit Answer



Reset Quiz ↻

Let's get our hands dirty with a few challenges related to pointers.

← Back

Next →

References

Challenge 1: Calculate the Area of a R...



Mark as Completed



Report an Issue