

Function pointer as argument in C

Till now, we have seen that in C programming, we can pass the variables as an argument to a function. We cannot pass the function as an argument to another function. But we can pass the reference of a function as a parameter by using a function pointer. This process is known as call by reference as the function parameter is passed as a pointer that holds the address of arguments. If any change made by the function using pointers, then it will also reflect the changes at the address of the passed variable.

Therefore, C programming allows you to create a pointer pointing to the function, which can be further passed as an argument to the function. We can create a function pointer as follows:

1. `(type) (*pointer_name)(parameter);`

In the above syntax, the **type** is the variable type which is returned by the function, ***pointer_name** is the function pointer, and the **parameter** is the list of the argument passed to the function.

Let's consider an example:

1. `float (*add()); // this is a legal declaration for the function pointer`
2. `float *add(); // this is an illegal declaration for the function pointer`

A function pointer can also point to another function, or we can say that it holds the address of another function.

1. `float add (int a, int b); // function declaration`
2. `float (*a)(int, int); // declaration of a pointer to a function`
3. `a=add; // assigning address of add() to 'a' pointer`

In the above case, we have declared a function named as '**add**'. We have also declared the function pointer (***a**) which returns the floating-type value, and contains two parameters of integer type. Now, we can assign the address of **add()** function to the '**a**' pointer as both are having the same return type(float), and the same type of arguments.

Now, '**a**' is a pointer pointing to the **add()** function. We can call the **add()** function by using the pointer, i.e., '**a**'. Let's see how we can do that:

1. `a(2, 3);`

The above statement calls the `add()` function by using pointer 'a', and two parameters are passed in 'a', i.e., 2 and 3.

Let's see a simple example of how we can pass the function pointer as a parameter.

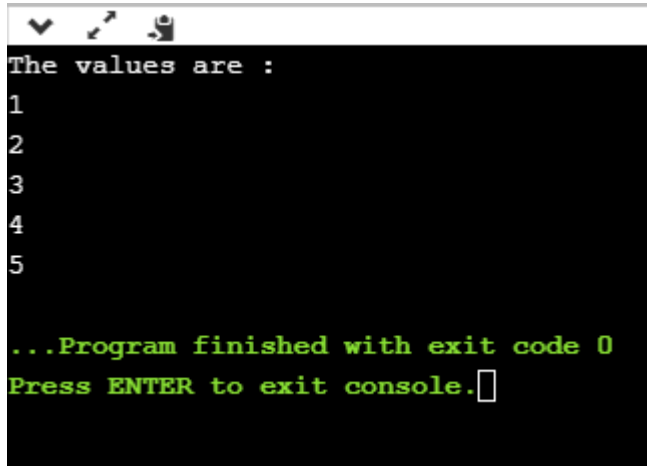
1. `void display(void (*p)())`
2. `{`
3. `for(int i=1;i<=5;i++)`
4. `{`
5. `p(i);`
6. `}`
7. `}`
8. `void print_numbers(int num)`
9. `{`
10. `cout<<num;`
11. `}`
12. `int main()`
13. `{`
14. `void (*p)(int); // void function pointer declaration`
15. `printf("The values are :");`
16. `display(print_numbers);`
17. `return 0;`
18. `}`

In the above code,

- We have defined two functions named 'display()' and `print_numbers()`.
- Inside the `main()` method, we have declared a function pointer named as `(*p)`, and we call the `display()` function in which we pass the `print_numbers()` function.
- When the control goes to the `display()` function, then pointer `*p` contains the address of `print_numbers()` function. It means that we can call the `print_numbers()` function using function pointer `*p`.
- In the definition of `display()` function, we have defined a 'for' loop, and inside the for loop, we call the `print_numbers()` function using statement `p(i)`. Here,

p(i) means that print_numbers() function will be called on each iteration of i, and the value of 'i' gets printed.

Output



```
The values are :
1
2
3
4
5
...Program finished with exit code 0
Press ENTER to exit console.█
```

Now, we will pass the function pointer as a argument in Quicksort function "qsort". It uses an algorithm that sorts an array.

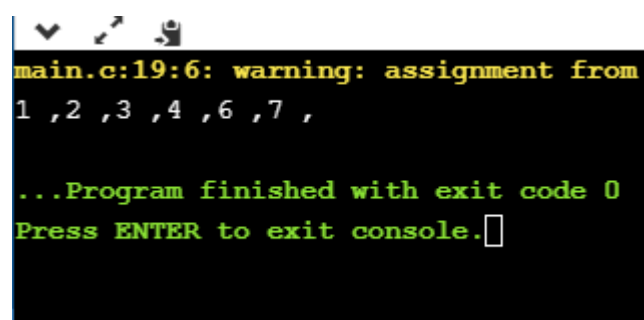
1. `#include <stdio.h>`
2. `#include <stdlib.h>`
- 3.
4. `#include<string.h>`
5. `int compare(const int *p, const int *q);`
6. `int (*f)(const void *a, const void *b);`
7. `int main()`
8. `{`
9. `int a[]={4,7,6,1,3,2};`
10. `int num=sizeof(a)/sizeof(int);`
11. `f=&compare;`
12. `qsort(a, num, sizeof(int), (*f));`
13. `for(int i=0;i<num;i++)`
14. `{`
15. `printf("%d ",a[i]);`
16. `}`
- 17.
18. `}`
- 19.
20. `int compare(const int *p, const int *q)`

```
21. {
22.     if (*p == *q)
23.         return 0;
24.     else if (*p < *q)
25.         return -1;
26.     else
27.         return 1;
28. }
```

In the above code,

- We have defined an array of integer type. After creating an array, we have calculated the size of an array by using the sizeof() operator, and stores the size in the **num**
- We define a compare() function, which compares all the elements in an array and arranges them in ascending order.
- We also have declared the function pointer, i.e., (*f), and stores the address of compare() function in (*f) by using the statement f=&compare.
- We call qsort() function in which we pass the array, size of the array, size of the element, and the comparison function. The comparison function, i.e., compare() will compare the array elements until the elements in an array get sorted in ascending order.

Output



```
main.c:19:6: warning: assignment from
1 ,2 ,3 ,4 ,6 ,7 ,
...Program finished with exit code 0
Press ENTER to exit console.□
```