Tutorial Link https://codequotient.com/tutorials/Function Pointers or Pointers to Functions or Callbacks/5a008c9bcbb2fe34b7774fc9

TUTORIAL

Function Pointers or Pointers to Functions or Callbacks

Chapter

On many occasions we may want to write a program to which a user defined function can be passed. For example, as you will see later in this article, one can write general functions to sort the numbers in different ways. We would then like to have it in such a way that a user can write a function, which sort all the numbers. We will now see how this can be achieved by using a pointer. Similar to a pointer to variable, we can have a pointer to a function. Function pointers can be declared, assigned values and then used to access the functions they point to. Function pointers are declared as follows:

```
type (*pointer_name)(function_argument_list);
```

Here pointer_name will be a pointer to a function returning type, for example,

```
int (*fp)();
double (*dptr)();
```

Here, fp is declared as a pointer to a function that returns int type, and dptr is a pointer to a function that returns double.

The interpretation is as follows for the first declaration: the dereferenced value of fp, i.e. (*fp) followed by () indicates a function, which returns integer type. The parentheses are essential in the declarations. The declaration without the parentheses:

```
int *fp();
```

It declares a function fp that returns an integer pointer. We can assign values to function pointer variables by making use of the fact that, in C/C++, the name of a function, used in an expression by itself, returns the address of a function. For example, if isunit() and isdoubleunit() are declared as follows:

```
int isunit(int n);
double isdoubleunit(double x);
```

the names of these functions, are equivalent to address for these functions. We can assign them to pointer variables:

```
fp = isunit;
dptr = isdoubleunit;
```

The functions can now be accessed, i.e. called, by dereferencing the function pointers as shown below:

```
int m, n=6; double j, k=6.8;
m = (*fp)(n);         /* calls isunit() with n as argument */
j = (*dptr)(k);          /* calls isdoubleunit() with k as argument
*/
```

Function pointers can be passed as parameters in function calls and can be returned as function values. Use of function pointers as parameters makes for flexible functions and programs.

Following is a program using function pointers: -

Function Pointers Implementation

C

```c
#include<stdio.h>

int sum (int num1, int num2)
{
        return num1+num2;
}

int main()
{
  int (*f2p) (int, int);
  f2p = sum;
  int op1 = f2p(10, 9);        //Calling function using function pointer
  int op2 = sum(10, 9);        //Calling function in normal way using function name
  printf("Output1: Call using function pointer: %d\n",op1);
  printf("Output2: Call using function name: %d\n", op2);
  return 0;
}

```

C++

```cpp
#include<iostream>
using namespace std;

int sum (int num1, int num2) {
  return num1+num2;
}

int main() {
  int (*f2p) (int, int);
```

```
  f2p = sum;
  int op1 = f2p(10, 9);        //Calling function using function pointer
  int op2 = sum(10, 9);        //Calling function in normal way using function name
  cout<<"Output1: Call using function pointer: "<<op1<<endl;
  cout<<"Output2: Call using function name: "<<op2<<endl;
  return 0;
}

```

Output:

```
Output1: Call using function pointer: 19
Output2: Call using function name: 19
```

Function pointers are useful in code reuse, Following program is defining a basic calculator, and by using the function pointers, it will perform all basic arithmetic operations on two integers: -

```c
#include<stdio.h>

int add(int x,int y)
{
        return (x+y);
}

int sub(int x,int y)
{
        return (x-y);
}

int mul(int x,int y)
{
```

```
        return (x*y);
}

int div(int x,int y)
{
        return (x/y);
}

void print(int x, int y, int (*func)(int,int))
{    // function pointer as argument
        printf(" value is : %d\n", ((*func)(x,y)));
 // calling function pointer with arguments.
}

int main()
{
  int x=100, y=20;
  printf("x=%d , y=%d \n",x,y);
  printf("Addition Result");
  print(x,y,add);
  printf("\nSubtraction Result");
  print(x,y,sub);
  printf("\nMultiply Result");
  print(x,y,mul);
  printf("\nDivision Result");
  print(x,y,div);
  return 0;
}

```

C++

```
#include<iostream>
using namespace std;

int add(int x,int y) {
        return (x+y);
}

int sub(int x,int y) {
```

```
9           return (x-y);
10  }
11
12  int mul(int x,int y) {
13          return (x*y);
14  }
15
16  int div(int x,int y) {
17          return (x/y);
18  }
19
20  void print(int x, int y, int (*func)(int,int)) {    //
    function pointer as argument
21          cout<<" value is : " << ((*func)(x,y))<<endl;
       // calling function pointer with arguments.
22  }
23
24  int main() {
25    int x=100, y=20;
26    cout<<"x="<<x<<", y="<<y<<endl;
27    cout<<"Addition Result";
28    print(x,y,add);
29    cout<<"Subtraction Result";
30    print(x,y,sub);
31    cout<<"Multiply Result";
32    print(x,y,mul);
33    cout<<"Division Result";
34    print(x,y,div);
35    return 0;
36  }
37
```

Output:

```
x=100 , y=20
Addition Result value is : 120
Subtraction Result value is : 80
Multiply Result value is : 2000
Division Result value is : 5
```