geeksforgeeks.org

Functors in C++ - GeeksforGeeks

3-4 minutes

Please note that the title is **Functors** (Not Functions)!!

Consider a function that takes only one argument. However, while calling this function we have a lot more information that we would like to pass to this function, but we cannot as it accepts only one parameter. What can be done?

One obvious answer might be global variables. However, good coding practices do not advocate the use of global variables and say they must be used only when there is no other alternative.

Functors are objects that can be treated as though they are a function or function pointer. Functors are most commonly used along with STLs in a scenario like following:

Below program uses transform() in STL to add 1 to all elements of arr[].

```
#include <bits/stdc++.h>
using namespace std;
int increment(int x) {  return (x+1); }
int main()
{
    int arr[] = {1, 2, 3, 4, 5};

    int n = sizeof(arr)/sizeof(arr[0]);

    transform(arr, arr+n, arr, increment);

    for (int i=0; i<n; i++)

        cout << arr[i] << S" ";

    return 0;

}
```

Output:

```
2 3 4 5 6
```

This code snippet adds only one value to the contents of the arr[]. Now suppose, that we want to add 5 to contents of arr[].

See what’s happening? As transform requires a unary function(a function taking only one argument) for an array, we cannot pass a number to increment(). And this would, in effect, make us write several different functions to add each number. What a mess. This is where functors come into use.

A functor (or function object) is a C++ class that acts like a function. Functors are called using the same old function call syntax. To create a functor, we create a object that overloads the *operator()*.

```
The line,
MyFunctor(10);

Is same as
MyFunctor.operator()(10);
```

Let’s delve deeper and understand how this can actually be used in conjunction with STLs.

```
#include <bits/stdc++.h>
using namespace std;
class increment
{
private:
    int num;
public:
    increment(int n) : num(n) {  }
    int operator () (int arr_num) const {
        return num + arr_num;
    }
};
int main()
{
    int arr[] = {1, 2, 3, 4, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    int to_add = 5;
    transform(arr, arr+n, arr,
increment(to_add));
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
}
```

Output:

```
6 7 8 9 10
```

Thus, here, Increment is a functor, a c++ class that acts as a function.

```
The line,
transform(arr, arr+n, arr, increment(to_add));

is the same as writing below two lines,
// Creating object of increment
increment obj(to_add);

// Calling () on object
transform(arr, arr+n, arr, obj);
```

Thus, an object *a* is created that overloads the *operator()*. Hence, functors can be used effectively in conjunction with C++ STLs.

This article is contributed by **Supriya Srivatsa**. If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above