Functors or function objects are contained in a C++ class which defines the operator **( )**. Functors let you create objects that "look like" functions because you can '*invoke*' the () operator, making the object behave like a function.

The advantage gained by functors is that they have state, i.e. their ability to hold values from one call to another.

Consider the code to add two numbers:

```
struct MyAddFunctor {

	// Constructor
	MyAddFunctor(int inp){
		x = inp;
	}

	// Defining operator()
	int operator()(int y){
		return x+y;
	}

	int x;
};

int main(){
	MyAddFunctor func(5);
	int ret = func(10);
	//ret would be 15.

	int ret2 = func(25);
	// ret would br 30
}
```

func, declared an object of MyAddFunctor, is instantiated to hold the value 5. Every 'call' of the functor, actually a call of the object's () operator, updates func's state.

We could have also combined instantiation and invocation in single statement as `MyAddFunctor(5)(10)`.

Why are they called as Function Objects?

The reason why functors are called function objects is because we can call an object of struct (it could be class) **MyAddFunctor** as if it is a function. Example : `func(10)`.

Why are functors used?

One can argue that the work done by **MyAddFunctor** can simply be done by writing C++ function as below:

```
int addFunction(int x){
	return 5+x;
}
```

But in **addFunction** we are hardcoding the value 5 and in the functor we are not doing so. A **MyAddFunctor** instantiation is more customizable.