Lambda

```
[](){}();
```

(It’s a valid c++ code)

CppIndia - User Group(18-07-2020)

Lambda

- Introduced in c++11 standard
- Allow us to write “un-named” functions in place
- Syntactic sugar for function objects (functors)

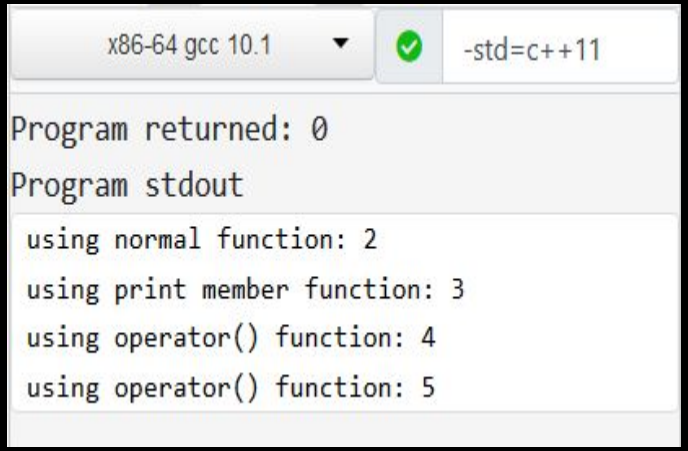
What is a function object a.k.a functors ?

```
void print(int x) {
    cout << " using normal function: " << x + 1 << endl;
}

struct A
{
    void print(int x) { cout << " using print member function: " << x + 1 << endl;}
    void operator() (int x) { cout << " using operator() function: " << x + 1 << endl;}
};

int main()
{
    // calling a normal function
    print(1);

    // Declare the object
    A a;
    a.print(2);
    a.operator()(3);

    // Use it as a function
    a(4);
}
```

- Objects that behave like “functions”
- A function object is simply a class/struct with an overloaded function call operator (operator ())

Output

```
void print(int x) {
    cout << " using normal function: " << x + 1 << endl;
}

struct A
{
    void print(int x) { cout << " using print member function: " << x + 1 << endl;}
    void operator() (int x) { cout << " using operator() function: " << x + 1 << endl;}
};

int main()
{
    // calling a normal function
    print(1);

    // Declare the object
    A a;
    a.print(2);
    a.operator()(3);

    // Use it as a function
    a(4);
}
```

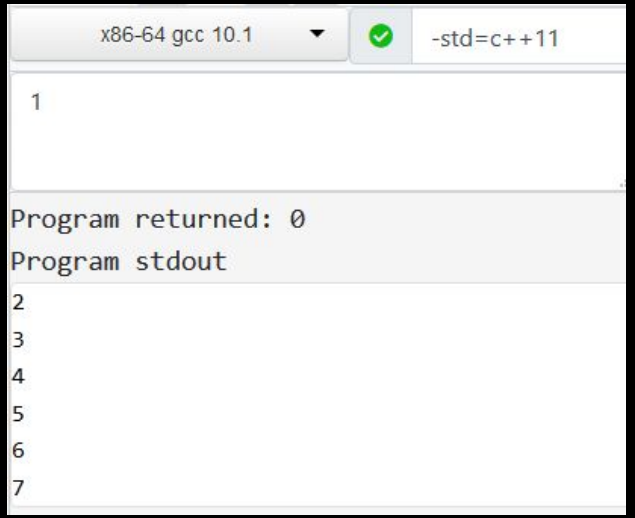


Pre-Lambda(Function objects)

```
struct AddAndPrint {
  AddAndPrint(int increment) : m_increment(increment) {}

  void operator()(const int& x) { cout << x + m_increment << endl; }

  int m_increment;
};

int main() {
  vector<int> v{1, 2, 3, 4, 5, 6};
  int increment;
  cin >> increment;
  AddAndPrint print(increment);
  for_each(v.begin(), v.end(), print);
  // for_each(v.begin(), v.end(), AddAndPrint(increment));
}
```

Output

```
struct AddAndPrint {
  AddAndPrint(int increment) : m_increment(increment) {}

  void operator()(const int& x) { cout << x + m_increment << endl; }

  int m_increment;
};

int main() {
  vector<int> v{1, 2, 3, 4, 5, 6};
  int increment;
  cin >> increment;
  AddAndPrint print(increment);
  for_each(v.begin(), v.end(), print);
  // for_each(v.begin(), v.end(), AddAndPrint(increment));
}
```



Here comes Lambda!

```
int main() {
  vector<int> v{1, 2, 3, 4, 5, 6};
  int increment;
  cin >> increment;
  for_each(v.begin(), v.end(), [increment](const int& x) { cout << x + increment << endl; });
}
```

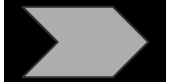
Parts of a lambda expression

```
[ captures] (parameters ) -> return type
{

      body;

};
```

captures -- To capture variables declared outside of lambda
parameters -- Input parameters for lambda, similar to function parameters
return type -- optional. Automatically deduced by compiler.
body -- regular function body

Lambdas “Under the hood”

```
[increment](int x) {
    cout << x + increment << endl;
}
```



```
class _lambda1_
{

    public :
        _lambda1_(int increment) :
            increment(increment) { };

        void operator() (int x) const
        {
            cout << x + increment << endl;
        }
    private :
        int increment;
}
```

Examples

Capture list

- No captures → []
- By value → [increment]
- By reference(*) → [&increment] (*Beware of dangling references)
- Shortcut for “all” value captures → [=]
- Shortcut for “all” reference captures → [&]
- Capture “all” by reference but x → [&, x]
- Capture “all” by value but x → [=, &x]
- Explicitly capture ‘this’ pointer → [this]

Enhancements in lambdas since c++14

- ‘auto’ keyword can be used to deduce the type of parameter

```
auto print = [](const auto & p) {
    cout << p.first << " " << p.second << endl;
};
```

- Allows to define arbitrary new local variables in the lambda object

```
    int x = 4;
    int z = [y = x+1] {
            return y+2;     // return 7 to initialize z
        }(); // invoke lambda
    cout << "z is " << z << endl;
}
```

Conclusion remarks

- Use lambdas to make your code more readable
  - No need to write full class
  - No need to find an appropriate name for the class
  - In-place code improves code readability
- STL algorithms and lambdas work great together!!
- Caution: If the body of the lambda contains more than a few lines of code, then it’s best to have a separate function for the same.