C++11 Lambda Functions

For Introduction to C++ Programming
By Y. Daniel Liang

C+11 supports lambda functions. To see the need for using lambda function, let us look at the following example from Listing 23.23 in the text book.

Listing 1 ForEachDemo.cpp

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <iterator>
using namespace std;

void display(int& value)
{
  cout << value << " ";
}

int square(int& value)
{
  return value * value;
}

int main()
{
  int array1[] = {1, 2, 3, 4, 5, 6, 7, 8};
  cout << "array1: ";
  for_each(array1, array1 + 8, display);

  vector<int> intVector(8);
  transform(array1, array1 + 8, intVector.begin(), square);
  cout << "\nintVector: ";
  for_each(intVector.begin(), intVector.end(), display);

  return 0;
}
```

Sample output

```
array1: 1 2 3 4 5 6 7 8
intVector: 1 4 9 16 25 36 49 64
```

The program uses the STL **for_each** algorithm to display each element in **array1** (line 21) and in **intVector** (line 26). The **display** function is defined in lines 7-10.

The program also uses the STL **transform** algorithm to apply the **sqaure** function to each element in **intVector** (line 24). The **square** function is defined in lines 12-15.

Both **display** and **square** are short functions and the **square** function is invoked in only one place in the program. Lambda functions can be used to simplify the code by defining the function where it is invoked.

Listing 2 rewrites Listing 1 using lambda functions.

Listing 2 ForEachDemoUsingLambdaFunctions.cpp

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

int main()
{
  int array1[] = {1, 2, 3, 4, 5, 6, 7, 8};
  cout << "array1: ";
  for_each(array1, array1 + 8, [](int v) { cout << v << " "; });

  vector<int> intVector(8);
  transform(array1, array1 + 8, intVector.begin(),
    [](int v) { return v * v; });
  cout << "\nintVector: ";
  for_each(intVector.begin(), intVector.end(),
    [](int v) { cout << v << " "; });

  return 0;
}
```

The program applies the lambda function on each element in array1 using the **for_each** algorithm (line 10). The lambda function is defined as

```
[](int v) { cout << v << " "; }
```

This function does not have a name. So, a lambda function is known as an anonymous function. The parameter **v** refers to the element in the array.

A lambda function may also return a value. The function in line 14

```
[](int v) { return v * v; }
```

returns **v * v**.

You can pass parameters by value or reference. The lambda function in Listing 2 are passed by value. You can rewrite it by passing reference as follows:

```
[](int& v) { cout << v << " "; }

[](int& v) { return v * v; }
```

The values in the collection are passed to the parameter defined within the parentheses **()**. The **[]** is empty in the example here. You can also use other variables in the lambda function. These variables can be defined in the brackets **[]**. This is called *capturing variables*.

Listing 3 gives an example that multiples a given value to the elements in a vector.

Listing 3 CaptureVariable.cpp

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

void multiply(vector<int>& v, int k)
{
  transform(v.begin(), v.end(), v.begin(),
    [k](int& v) { return v * k; });
}

int main()
{
  vector<int> v(4);
  v[0] = 0; v[1] = 1; v[2] = 2; v[3] = 3;

  multiply(v, 3);

  for_each(v.begin(), v.end(),
    [](int value) { cout << value << " "; });

  return 0;
}
```

The variable **k** is captured in the lambda function using the syntax

```
[k](int& v) { return v * k; }
```

If it is not captured in the brackets, it cannot be referenced inside the lambda function. You can capture any number of the variables in a lambda function.

In the general, a lambda function can be defined using the following syntax:

[capture variable**]** **(**params**)** **{** body **}**