



- [Home](#)
- [About](#)
- [Business Plan »](#)
- [Communication »](#)
- [Dieting](#)
- [Sales](#)
- [Sitemap](#)
- [Videos »](#)
- [Web Design »](#)
- [Communication »](#)
- [Diet Nutritional](#)
- [Flash Tutorial](#)
- [How To »](#)
- [Investing](#)
- [iPad »](#)
- [Marketing »](#)
- [Most Popular](#)
- [Royalty Free Photos](#)
- [Sales](#)
- [Web Design »](#)

[share](#)

C++ Tutorial 9

Posted by [Derek Banas](#) on Apr 11, 2018 in [C Video Tutorial](#) | [0 comments](#)



This part of my C++ tutorial provides numerous examples of what you can do with Lambda Expressions. We'll cover how to Sort, Filter, Sum, Edit and Generate Lists based on conditions. We'll also perform operations on multiple lists, create Recursive Lambda Functions and cover the Ternary Operator.

Of course we will solve problems and I provide all of the code along with a transcript of the tutorial below.

If you like videos like this, consider donating \$1, or simply turn off Ad Blockers. Either helps me to continue making free educational videos.



Code & Transcript

```
1 // ----- C++ TUTORIAL 9 -----
2
3 #include <cstdlib>
4 #include <iostream>
5 #include <string>
6 #include <vector>
7 #include <numeric>
8 #include <sstream>
9 #include <ctime>
10 #include <cmath>
11
12 std::vector<int> GenerateRandVec(int numOfNums,
13     int min, int max);
14
15 std::vector<int> GenerateFibList(int maxNums);
16
17 int main() {
18
19     // ----- LAMBDA EXPRESSIONS -----
20
21     std::vector<int> vecVals = GenerateRandVec(10, 1, 50);
22
23     // Lambda Expressions make it easy to perform list
24     // operations in one line of code. You designate
25     // them with []
26     // Here we sort a vector
27     std::sort(vecVals.begin(), vecVals.end(),
28         [](int x, int y){ return x < y; });
29
30     for(auto val: vecVals)
31         std::cout << val << "\n";
32
33     std::cout << "\n";
34 }
```

```

35 // copy_if works like filter does in other languages
36 // Here we keep only even values in a new vector
37 std::vector<int> evenVecVals;
38
39 std::copy_if(vecVals.begin(), vecVals.end(),
40             std::back_inserter(evenVecVals),
41             [](int x){ return (x % 2) == 0; });
42
43 for(auto val: evenVecVals)
44     std::cout << val << "\n";
45
46 // ----- SUM A LIST -----
47 int sum = 0;
48
49 // for_each cycles through all values
50 // [&] captures all variables used
51 // in the body of the lambda by reference
52 std::for_each(vecVals.begin(), vecVals.end(),
53             [&] (int x) {sum += x; });
54
55     std::cout << "SUM : " << sum << "\n";
56
57 // ----- END SUM A LIST -----
58
59 // ----- PROBLEM DYNAMIC LIST DIVISIBLE BY A VALUE -----
60
61 // You can define what value is checked for divisability
62 // by passing the value to check in the capture list
63 // which lies between []
64 // Send a value entered by the user through the capture
65 // list and create a list based on it
66 int divisor;
67 std::vector<int> vecVals2;
68 std::cout << "List of values divisible by : ";
69 std::cin >> divisor;
70 std::copy_if(vecVals.begin(), vecVals.end(),
71             std::back_inserter(vecVals2),
72             [divisor](int x){ return (x % divisor) == 0; });
73 for(auto val: vecVals2)
74     std::cout << val << "\n";
75
76 // ----- END PROBLEM DYNAMIC LIST DIVISIBLE BY A VALUE -----
77
78 // ----- MULTIPLY ALL VALUES BY 2 -----
79 std::vector<int> doubleVec;
80
81 // For_each cycles through all values in the vector
82 // and doubles them.
83 std::for_each(vecVals.begin(), vecVals.end(),
84             [&](int x){doubleVec.push_back(x * 2);});
85
86 for(auto val: doubleVec)
87     std::cout << "DBL : " << val << "\n";
88
89 // ----- END MULTIPLY ALL VALUES BY 2 -----
90
91 // ----- PERFORMING OPERATIONS USING MULTIPLE VECTORS -----
92 // Add values in separate vectors and save them to another
93 std::vector<int> vec1 = {1,2,3,4,5};
94 std::vector<int> vec2 = {1,2,3,4,5};
95 std::vector<int> vec3(5);
96 transform(vec1.begin(), vec2.end(),
97           vec2.begin(), vec3.begin(),
98           [](int x, int y) {return x + y;});
99

```

```

100     for(auto val: vec3)
101         std::cout << "vec3 " << val << "\n";
102
103     // ----- END PERFORMING OPERATIONS USING MULTIPLE VECTORS -----
104
105     // ----- TERNARY OPERATOR -----
106
107     // A ternary operator works like a compact if else
108     // statement. If the condition is true the first
109     // value is stored and otherwise the second
110     int age = 43;
111     bool canIVote = (age >= 18) ? true : false;
112     // Shows bool values as true or false
113     std::cout.setf(std::ios::boolalpha);
114     std::cout << "Can Derek Vote : " << canIVote << "\n";
115
116     // ----- END TERNARY OPERATOR -----
117
118     // ----- RECURSIVE LAMBDA FUNCTIONS -----
119
120     // Recursive Lambda to calculate Fibonacci Numbers
121     std::function<int(int)> Fib =
122     [&Fib](int n) {return n < 2 ? n : Fib(n-1) + Fib(n-2);};
123
124     // Fib(0) = 0
125     // Fib(1) = 1
126     // Fib(2) = 1
127     // Fib(3) = 2
128     // Fib(4) = 3
129     std::cout << "Fib 4 : " << Fib(4) << "\n";
130
131     // ----- END RECURSIVE LAMBDA FUNCTIONS -----
132
133     // ----- PROBLEM GENERATE DYNAMIC VECTOR OF FIBS -----
134     std::vector<int> listOfFibs = GenerateFibList(10);
135     for(auto val: listOfFibs)
136         std::cout << val << "\n";
137     // ----- END PROBLEM GENERATE DYNAMIC VECTOR OF FIBS -----
138
139     return 0;
140 }
141
142 std::vector<int> GenerateRandVec(int numOfNums,
143     int min, int max){
144     std::vector<int> vecValues;
145     srand(time(NULL));
146     int i = 0, randVal = 0;
147     while(i < numOfNums){
148         randVal = min + std::rand() % ((max + 1) - min);
149         vecValues.push_back(randVal);
150         i++;
151     }
152     return vecValues;
153 }
154
155 // ----- PROBLEM GENERATE DYNAMIC VECTOR OF FIBS -----
156 std::vector<int> GenerateFibList(int maxNums){
157     std::vector<int> listOfFibs;
158     int i = 0;
159     std::function<int(int)> Fib =
160     [&Fib](int n) {return n < 2 ? n : Fib(n-1) + Fib(n-2);};
161     while(i < maxNums){
162         listOfFibs.push_back(Fib(i));
163         i++;
164     }

```

```
165     return listOfFibs;
166 }
167 // ----- END PROBLEM GENERATE DYNAMIC VECTOR OF FIBS -----
```

Leave a Reply

Your email address will not be published.

Comment

Name


Email

Website

Search

Social Networks

 Facebook

 YouTube

 Twitter

 LinkedIn



Buy me a Cup of Coffee

"Donations help me to keep the site running. One dollar is greatly appreciated." - (Pay Pal Secured)





My Facebook Page

6K people like this. Be the first of your friends.

Archives

- [October 2018](#)
- [September 2018](#)
- [August 2018](#)
- [July 2018](#)
- [June 2018](#)
- [May 2018](#)
- [April 2018](#)
- [March 2018](#)
- [February 2018](#)

- [January 2018](#)
- [December 2017](#)
- [November 2017](#)
- [October 2017](#)
- [September 2017](#)
- [August 2017](#)
- [July 2017](#)
- [June 2017](#)
- [May 2017](#)
- [April 2017](#)
- [March 2017](#)
- [February 2017](#)
- [January 2017](#)
- [December 2016](#)
- [November 2016](#)
- [October 2016](#)
- [September 2016](#)
- [August 2016](#)
- [July 2016](#)
- [June 2016](#)
- [May 2016](#)
- [April 2016](#)
- [March 2016](#)
- [February 2016](#)
- [January 2016](#)
- [December 2015](#)
- [November 2015](#)
- [October 2015](#)
- [September 2015](#)
- [August 2015](#)
- [July 2015](#)
- [June 2015](#)
- [May 2015](#)
- [April 2015](#)
- [March 2015](#)
- [February 2015](#)
- [January 2015](#)
- [December 2014](#)
- [November 2014](#)
- [October 2014](#)
- [September 2014](#)
- [August 2014](#)
- [July 2014](#)
- [June 2014](#)
- [May 2014](#)
- [April 2014](#)
- [March 2014](#)
- [February 2014](#)
- [January 2014](#)
- [December 2013](#)
- [November 2013](#)
- [October 2013](#)
- [September 2013](#)
- [August 2013](#)

- [July 2013](#)
- [June 2013](#)
- [May 2013](#)
- [April 2013](#)
- [March 2013](#)
- [February 2013](#)
- [January 2013](#)
- [December 2012](#)
- [November 2012](#)
- [October 2012](#)
- [September 2012](#)
- [August 2012](#)
- [July 2012](#)
- [June 2012](#)
- [May 2012](#)
- [April 2012](#)
- [March 2012](#)
- [February 2012](#)
- [January 2012](#)
- [December 2011](#)
- [November 2011](#)
- [October 2011](#)
- [September 2011](#)
- [August 2011](#)
- [July 2011](#)
- [June 2011](#)
- [May 2011](#)
- [April 2011](#)
- [March 2011](#)
- [February 2011](#)
- [January 2011](#)
- [December 2010](#)
- [November 2010](#)
- [October 2010](#)
- [September 2010](#)
- [August 2010](#)
- [July 2010](#)
- [June 2010](#)
- [May 2010](#)
- [April 2010](#)
- [March 2010](#)
- [February 2010](#)
- [January 2010](#)
- [December 2009](#)

Powered by [WordPress](#) | Designed by [Elegant Themes](#)
[About the Author](#) [Google+](#)