NEW THINK TANK

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.

G+

Code & Transcript

```
// ---------- C++ TUTORIAL 9 ----------

#include <cstdlib>
#include <iostream>
#include <string>
#include <vector>
#include <numeric>
#include <sstream>
#include <ctime>
#include <cmath>

std::vector<int> GenerateRandVec(int numOfNums,
        int min, int max);

std::vector<int> GenerateFibList(int maxNums);

int main() {

    // ----- LAMBDA EXPRESSIONS -----

    std::vector<int> vecVals = GenerateRandVec(10, 1, 50);

    // Lambda Expressions make it easy to perform list
    // operations in one line of code. You designate
    // them with []
    // Here we sort a vector
    std::sort(vecVals.begin(), vecVals.end(),
            [](int x, int y){ return x < y; });

    for(auto val: vecVals)
        std::cout << val << "\n";

    std::cout << "\n";

```

```
    // copy_if works like filter does in other languages
    // Here we keep only even values in a new vector
    std::vector<int> evenVecVals;

    std::copy_if(vecVals.begin(), vecVals.end(),
            std::back_inserter(evenVecVals),
            [](int x){ return (x % 2) == 0; });

    for(auto val: evenVecVals)
        std::cout << val << "\n";

    // ----- SUM A LIST -----
    int sum = 0;

    // for_each cycles through all values
    // [&] captures all variables used
    // in the body of the lambda by reference
    std::for_each(vecVals.begin(), vecVals.end(),
            [&] (int x) {sum += x; });

            std::cout << "SUM : " << sum << "\n";

    // ----- END SUM A LIST -----

    // ----- PROBLEM DYNAMIC LIST DIVISABLE BY A VALUE -----

    // You can define what value is checked for divisability
    // by passing the value to check in the capture list
    // which lies between []
    // Send a value entered by the user through the capture
    // list and create a list based on it
    int divisor;
    std::vector<int> vecVals2;
    std::cout << "List of values divisable by : ";
    std::cin >> divisor;
    std::copy_if(vecVals.begin(), vecVals.end(),
            std::back_inserter(vecVals2),
            [divisor](int x){ return (x % divisor) == 0; });
    for(auto val: vecVals2)
        std::cout << val << "\n";

    // ----- END PROBLEM DYNAMIC LIST DIVISABLE BY A VALUE -----

    // ----- MULTIPLY ALL VALUES BY 2 -----
    std::vector<int> doubleVec;

    // For_each cycles through all values in the vector
    // and doubles them.
    std::for_each(vecVals.begin(), vecVals.end(),
            [&](int x){doubleVec.push_back(x * 2);});

    for(auto val: doubleVec)
        std::cout << "DBL : " << val << "\n";

    // ----- END MULTIPLY ALL VALUES BY 2 -----

    // ----- PERFORMING OPERATIONS USING MULTIPLE VECTORS -----
    // Add values in separate vectors and save them to another
    std::vector<int> vec1 = {1,2,3,4,5};
    std::vector<int> vec2 = {1,2,3,4,5};
    std::vector<int> vec3(5);
    transform(vec1.begin(), vec2.end(),
            vec2.begin(), vec3.begin(),
            [](int x, int y) {return x + y;});

```

```
    for(auto val: vec3)
        std::cout << "vec3 " << val << "\n";

    // ----- END PERFORMING OPERATIONS USING MULTIPLE VECTORS -----

    // ----- TERNARY OPERATOR -----

    // A ternary operator works like a compact if else
    // statement. If the condition is true the first
    // value is stored and otherwise the second
    int age = 43;
    bool canIVote = (age >= 18) ? true : false;
    // Shows bool values as true or false
    std::cout.setf(std::ios::boolalpha);
    std::cout << "Can Derek Vote : " << canIVote << "\n";

    // ----- END TERNARY OPERATOR -----

    // ----- RECURSIVE LAMBDA FUNCTIONS -----

    // Recursive Lambda to calculate Fibonacci Numbers
    std::function<int(int)> Fib =
    [&Fib](int n) {return n < 2 ? n : Fib(n-1) + Fib(n-2);};

    // Fib(0) = 0
    // Fib(1) = 1
    // Fib(2) = 1
    // Fib(3) = 2
    // Fib(4) = 3
    std::cout << "Fib 4 : " << Fib(4) << "\n";

    // ----- END RECURSIVE LAMBDA FUNCTIONS -----

    // ----- PROBLEM GENERATE DYNAMIC VECTOR OF FIBS -----
    std::vector<int> listOfFibs = GenerateFibList(10);
    for(auto val: listOfFibs)
        std::cout << val << "\n";
    // ----- END PROBLEM GENERATE DYNAMIC VECTOR OF FIBS -----

    return 0;
}

std::vector<int> GenerateRandVec(int numOfNums,
        int min, int max){
    std::vector<int> vecValues;
    srand(time(NULL));
    int i = 0, randVal = 0;
    while(i < numOfNums){
        randVal = min + std::rand() % ((max + 1) - min);
        vecValues.push_back(randVal);
        i++;
    }
    return vecValues;
}

// ----- PROBLEM GENERATE DYNAMIC VECTOR OF FIBS -----
std::vector<int> GenerateFibList(int maxNums){
    std::vector<int> listOfFibs;
    int i = 0;
    std::function<int(int)> Fib =
    [&Fib](int n) {return n < 2 ? n : Fib(n-1) + Fib(n-2);};
    while(i < maxNums){
        listOfFibs.push_back(Fib(i));
        i++;
    }
```

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

Leave a Reply

Your email address will not be published.

Comment

Name

Email

Website

Submit Comment

Search

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)

Donate

My Facebook Page

Like Share 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