NEW THINK TANK





# C++ Tutorial 12

Posted by Derek Banas on Apr 25, 2018 in C Video Tutorial | 0 comments

In this part of my C++ tutorial we'll cover Operator Overloading, File I/O and will present you with another problem to solve. We'll learn to use unary and Binary operators. We'll specifically overload ++, print customer strings, +, [], ==, <, >, and =. Then we'll write and read from files.

All of the code and a transcript follows the video below. I hope you enjoy the tutorial.

If you like tutorials like this, consider donating $1, or simply turn off Ad Blocking software when watching my videos. Either helps me to keep my videos 100% free.

### Code & Transcript

```
// ---------- C++ TUTORIAL 12 ----------

// ---------- OPERATOR OVERLOADING ----------

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

// Needed for ostringstream
#include <sstream>

// Create a custom Box class with overloaded operators
class Box{
public:
    double length, width, breadth;

    // Used to hold a string representation of a box
    std::string boxString;

    Box(){
        length = 1, width = 1, breadth = 1;
    }
    Box(double l, double w, double b){
        length = l, width = w, breadth = b;
    }

    // You can define customer operators just like
    // you define functions
    // This is a unary operator because it operates
    // on 1 object
```

```
35      // Other Unary Operators : --, *(pointer dereference),
36      // -> (Member Selection), !, & (Address of), +, -
37      Box& operator ++ (){
38          length++;
39          width++;
40          breadth++;
41          return *this;
42      }
43
44      // Creates a C string representation which is a
45      // pointer to an array that is null terminated
46      operator const char*() {
47          // Creates a stream that can be loaded with
48          // characters that can then be accessed as
49          // a string object
50          std::ostringstream boxStream;
51          boxStream << "Box : " <<
52                  length << ", " <<
53                  width << ", " <<
54                  breadth;
55
56          // Return a string representation of the stream
57          boxString = boxStream.str();
58
59          // Returns the pointer to the string array
60          return boxString.c_str();
61      }
62
63      // Binary operators operate on 2 objects
64      // +, -, *, /, %, ==, !=, >, <, >=, <=, &&, ||,
65      // !, =, +=, -=, *=, /=, ^, [], &, |
66
67      // Let's add boxes
68      Box operator + (const Box& box2){
69          Box boxSum;
70          boxSum.length = length + box2.length;
71          boxSum.width = width + box2.width;
72          boxSum.breadth = breadth + box2.breadth;
73          return boxSum;
74      }
75
76      // Access items using a subscript operator
77      double operator [] (int x){
78          if(x == 0){
79              return length;
80          } else if(x == 1){
81              return width;
82          } else if(x == 2){
83              return breadth;
84          } else {
85              return 0;
86          }
87      }
88
89      // Check for box equality
90      bool operator == (const Box& box2){
91          return ((length == box2.length) &&
92                  (width == box2.width) &&
93                  (breadth == box2.breadth));
94      }
95
96      // Check for which is bigger
97      bool operator < (const Box& box2){
98          double thisSize = length + width + breadth;
99          double box2Size = box2.length + box2.width +
```

```
100         box2.breadth;
101         if (thisSize < box2Size){
102             return true;
103         } else {
104             return false;
105         }
106     }
107 
108     bool operator > (const Box& box2){
109         double thisSize = length + width + breadth;
110         double box2Size = box2.length + box2.width +
111         box2.breadth;
112         if (thisSize > box2Size){
113             return true;
114         } else {
115             return false;
116         }
117     }
118 
119     // Overload the assignment operator
120     void operator = (const Box& boxToCopy){
121         length = boxToCopy.length;
122         width = boxToCopy.width;
123         breadth = boxToCopy.breadth;
124     }
125 };
126 
127 int main()
128 {
129     Box box(10,10,10);
130 
131     // Will increment all values in the box by 1
132     ++box;
133     std::cout << box << "\n";
134 
135     // Add boxes
136     Box box2(5,5,5);
137     std::cout << "Box1 + Box2 = " <<
138             box + box2 << "\n";
139 
140     // Access data with subscript operator
141     std::cout << "Box Length : " <<
142             box[0] << "\n";
143 
144     // Displays true or false for bolleans
145     std::cout << std::boolalpha;
146     std::cout << "Are boxes equal : " <<
147             (box == box2) << "\n";
148 
149     // Is box < box2
150     std::cout << "Is box < box2 : " <<
151             (box < box2) << "\n";
152 
153     // Is box > box2
154     std::cout << "Is box < box2 : " <<
155             (box > box2) << "\n";
156 
157     box = box2;
158     std::cout << box << "\n";
159     return 0;
160 }
161 
162 // ---------- OPERATOR OVERLOADING ----------
163 
164 // ---------- FILE I/O & PROBLEM ----------
```

```
165
166 #include <cstdlib>
167 #include <iostream>
168 #include <string>
169 #include <vector>
170 #include <ctime>
171 #include <numeric>
172 #include <cmath>
173 #include <sstream>
174
175 // iostream allows use to read from the standard
176 // input (keyboard) and write to the standard output
177 // (console)
178 // fstream is needed for working with files
179 #include <fstream>
180
181 // ----- PROBLEM FUNCTION PROTOTYPE -----
182
183 std::vector<std::string> StringToVector(std::string,
184         char separator);
185
186 // ----- END OF PROBLEM FUNCTION PROTOTYPE -----
187
188 int main()
189 {
190     std::ofstream writeToFile;
191     std::ifstream readFromFile;
192     std::string txtToWrite = "";
193     std::string txtFromFile = "";
194
195     // We open the file by providing a name and then either
196     // ios::app : Append to the end of the file
197     // ios::trunc : If the exists delete content
198     // ios::in : Open file for reading
199     // ios::out : Open file for writing
200     // ios::ate : Open writing and move to the end of the file
201     writeToFile.open("test.txt", std::ios_base::out |
202             std::ios_base::trunc);
203
204     if(writeToFile.is_open()){
205         // You can write with the stream insertion operator
206         writeToFile << "Beginning of File\n";
207
208         // You can write data in a string
209         std::cout << "Enter data to write : ";
210         getline(std::cin, txtToWrite);
211         writeToFile << txtToWrite;
212
213         // Close the file
214         writeToFile.close();
215     }
216
217     // Open the file for reading
218     readFromFile.open("test.txt", std::ios_base::in);
219
220     if(readFromFile.is_open()){
221
222         // Read text from file
223         while(readFromFile.good()){
224             getline(readFromFile, txtFromFile);
225
226             // Print text from file
227             std::cout << txtFromFile << "\n";
228
229             // ----- PROBLEM -----
```

```
230             // After each line print both the number of
231             // words in each line and the average word length
232
233             std::vector<std::string> vect =
234                     StringToVector(txtFromFile, ' ');
235
236             int wordsInLine = vect.size();
237
238             std::cout << "Words in Line : " <<
239                      wordsInLine << "\n";
240
241             int charCount = 0;
242
243             for(auto word: vect){
244                 for(auto letter: word){
245                     charCount++;
246                 }
247             }
248
249             int avgNumChars = charCount/wordsInLine;
250
251             std::cout << "Avg Word Length : " <<
252             avgNumChars << "\n";
253
254             // ----- END OF PROBLEM -----
255         }
256         readFromFile.close();
257     }
258
259     return 0;
260 }
261
262 // ----- PROBLEM FUNCTION -----
263
264 std::vector<std::string> StringToVector(std::string theString,
265         char separator){
266
267     // Create a vector
268     std::vector<std::string> vecsWords;
269
270     // A stringstream object receives strings separated
271     // by a space and then spits them out 1 by 1
272     std::stringstream ss(theString);
273
274     // Will temporarily hold each word in the string
275     std::string sIndivStr;
276
277     // While there are more words to extract keep
278     // executing
279     // getline takes strings from a stream of words stored
280     // in the stream and each time it finds a blanks space
281     // it stores the word proceeding the space in sIndivStr
282     while(getline(ss, sIndivStr, separator)){
283
284         // Put the string into a vector
285         vecsWords.push_back(sIndivStr);
286     }
287
288     return vecsWords;
289 }
290
291 // ----- END OF PROBLEM FUNCTION -----
292
293 // ---------- FILE I/O & PROBLEM ----------
```

## 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
- 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