



- [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 14

Posted by [Derek Banas](#) on May 6, 2018 in [C Video Tutorial](#) | [0 comments](#)



In this tutorial I cover many topics many people have asked for. I show how to include outside files, Preprocessor Directives, Macro Constants, Macro Functions, Template Functions, Template Classes, and Iterators.

Like always all of the code follows the video below. It is heavily commented and it is recommended to print it out and take notes on it as you watch.

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



Code from the Video

```
1  --- ANIMAL.H ---
2
3  // This guards against including this header in multiple
4  // files that make up the program along with #endif
5  #ifndef ANIMAL_H
6
7  // Read the following lines once
8  #define ANIMAL_H
9
10 #include <string>
11
12 class Animal {
13 public:
14     Animal();
15     Animal(const Animal& orig);
16     virtual ~Animal();
17     std::string name;
18 private:
19
20 };
21
22 #endif /* ANIMAL_H */
23
24 --- ANIMAL.CPP ---
25
26 #include "Animal.h"
27
28 Animal::Animal() {
29 }
30
31 Animal::Animal(const Animal& orig) {
32 }
33
34 Animal::~~Animal() {
```

```
35 }
36
37 #include <cstdlib>
38 #include <iostream>
39 #include <string>
40 #include <vector>
41 #include <ctime>
42 #include <numeric>
43 #include <cmath>
44 #include <sstream>
45
46 #include <deque>
47 #include <iterator>
48
49 // ----- PREPROCESSOR DIRECTIVES -----
50
51 // Anything that starts with a # is a preprocessor
52 // directive and they run before the program compiles
53
54 // Right click Header Files -> New -> C++ Header File
55 // and then include it here
56 #include "Animal.h"
57
58 // This is a macro constant that will replace
59 // PI with 3.14159 in the code before execution
60 #define PI 3.14159
61
62 // This is a macro function that will do the same with
63 // a function
64 #define AREA_CIRCLE(radius) (PI * (std::pow(radius, 2)))
65
66 // ----- END PREPROCESSOR DIRECTIVES -----
67
68 // ----- TEMPLATE FUNCTIONS -----
69
70 // We use templates to create functions or classes
71 // that can work with many types
72 // Templates differ from function overloading in that
73 // instead of having a function that does similar
74 // things with different objects a template does the
75 // same thing with different objects
76
77 // This says this is a function template that generates
78 // functions that except 1 parameter
79 template <typename T>
80 void Times2(T val){
81     std::cout << val << " * 2 = " <<
82         val * 2 << "\n";
83 }
84
85 // Receive multiple parameters and return a value
86 template <typename T>
87 T Add(T val, T val2){
88     return val + val2;
89 }
90
91 // Work with chars and strings
92 template <typename T>
93 T Max(T val, T val2){
94     return (val < val2) ? val2 : val;
95 }
96
97 // ----- END OF TEMPLATE FUNCTIONS -----
98
99 // ----- TEMPLATE CLASSES -----
```

```
100
101 // Template classes are classes that can work with
102 // different data types
103
104 // You can define that you may receive parameters
105 // with different types, but they don't have to
106 // be different
107 template <typename T, typename U>
108 class Person{
109 public:
110     T height;
111     U weight;
112     static int numOfPeople;
113     Person(T h, U w){
114         height = h, weight = w;
115         numOfPeople++;
116     }
117     void GetData(){
118         std::cout << "Height : " <<
119             height <<
120             " and Weight : " <<
121             weight << "\n";
122     }
123 };
124
125 // You have to initialize static class members
126 template<typename T, typename U> int Person<T, U>::numOfPeople;
127
128
129 // ----- END OF TEMPLATE CLASSES -----
130
131 int main()
132 {
133     Animal spot = Animal();
134     spot.name = "Spot";
135     std::cout << "The Animal is named " <<
136         spot.name << "\n";
137
138     std::cout << "Circle Area : " <<
139         AREA_CIRCLE(5) << "\n";
140
141     // ----- TEMPLATE FUNCTIONS -----
142     // The template function can receive ints or floats
143     Times2(5);
144     Times2(5.3);
145
146     // Multiple parameters and returned value
147     std::cout << "5 + 4 = " <<
148         Add(5,4) << "\n";
149     std::cout << "5.5 + 4.6 = " <<
150         Add(5.5,4.6) << "\n";
151
152     // Get biggest value
153     std::cout << "Max 4 or 8 = " <<
154         Max(4, 8) << "\n";
155     std::cout << "Max A or B = " <<
156         Max('A', 'B') << "\n";
157     std::cout << "Max Dog or Cat = " <<
158         Max("Dog", "Cat") << "\n";
159
160     // ----- END OF TEMPLATE FUNCTIONS -----
161
162     // ----- TEMPLATE CLASSES -----
163
164     // When creating the object you must define the
```

```
165 // data types used
166 Person<double, int> mikeTyson (5.83, 216);
167 mikeTyson.GetData();
168
169 // You access static values using the object
170 // and not the class
171 std::cout << "Number of people : " <<
172     mikeTyson.numOfPeople << "\n";
173
174 // ----- END OF TEMPLATE CLASSES -----
175
176 // ----- CONTAINERS -----
177 // We have already seen the STL container vector
178 // There are many other special ways of storing data
179
180 // ----- DOUBLE ENDED QUEUE -----
181
182 // A double ended queue (Deck) is a dynamic array that can
183 // be expanded or contracted on both ends
184 std::deque<int> nums = {1,2,3,4};
185 nums.push_front(0);
186 nums.push_back(5);
187 for(int x: nums)
188     std::cout << x << "\n";
189
190 // You can access index values, but they are costly
191 // because values aren't stored contigously, but
192 // instead use multiple arrays
193 std::cout << nums[0] << "\n";
194
195 // ----- END DOUBLE ENDED QUEUE -----
196
197 // ----- ITERATORS -----
198 // Iterators are used to point at container
199 // memory locations
200 std::vector<int> nums2 = {1,2,3,4};
201
202 // Define an iterator as the same type
203 std::vector<int>::iterator itr;
204
205 // Refer to the vectors begin and end while
206 // incrementing the iterator
207 for(itr = nums2.begin();
208     itr < nums2.end();
209     itr++){
210
211     // Get value at the pointer
212     std::cout << *itr << "\n";
213 }
214
215 // You can also increment a set number of spaces
216 // Create an iterator and point it at the beginning
217 // of the vector
218 std::vector<int>::iterator itr2 = nums2.begin();
219
220 // Advance 2 spaces
221 advance(itr2, 2);
222 std::cout << *itr2 << "\n";
223
224 // Next works like advance, but it returns an
225 // iterator
226 auto itr3 = next(itr2, 1);
227 std::cout << *itr3 << "\n";
228
229 // Previous moves a set number of indexes and
```

```
230 // returns an iterator
231 auto itr4 = prev(itr2, 1);
232 std::cout << *itr4 << "\n";
233
234 // You can also insert at a defined index
235 std::vector<int> nums3 = {1,4,5,6};
236 std::vector<int> nums4 = {2,3};
237 auto itr5 = nums3.begin();
238 advance(itr5, 1);
239 copy(nums4.begin(), nums4.end(),
240      inserter(nums3, itr5));
241
242 for(int &i: nums3)
243     std::cout << i << "\n";
244
245 // ----- END ITERATORS -----
246
247 // ----- END OF CONTAINERS -----
248
249 return 0;
250 }
```

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)

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](#)

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