Sean Armbruster   
CSC 134   
Chapter 16 Review Questions  
1-17, 19, 20 p. 1024

1. A “throw point” is the line of code that contains a “throw” statement.  
  
2. When a throw statement is executed, control is passed to another part of the program called the “Exception Handler.”  
  
3. A “try block” contains a block of code that might directly or indirectly cause an exception to be thrown. The “catch block” is the exception handler.   
  
4. In some instances, if an exception is thrown but not caught, the exception will cause the entire program to abort execution.   
5. “Unwinding The Stack” is when an exception has been thrown but a chain of nested function calls continue from the throw point to the try block.  
  
6. If an exception is thrown by a class’s member function, the class’s destructor will be called.   
  
7. To prevent a program from halting when the new operator fails to allocate memory, you must utilize the “bad\_alloc” exception in the std namespace.  
  
8. It is more convenient to write a function template than a series of overloaded functions because you can write a single function that works with many different data types versus having many overloaded functions for each data type.  
  
9. You must be careful when using operators such as [] when writing a function template because the types passed to the template must support all the operators the function will use.   
  
10. Throw point  
11. Try   
12. Catch   
13. Type Parameter   
14. Template Prefix   
15. Data Type   
16. Specialized   
17.   
 unsigned getIndex( int arr[], const unsigned SIZE, int value) {  
 for( unsigned i = 0; i < SIZE; i++) {  
 if( arr[i] == value)   
 return i;  
 }  
 throw ValueNotFoundException();  
 }  
   
19.   
 template <class T>

T getIndex( T arr[], const unsigned SIZE, T value) {  
 for( unsigned i = 0; i < SIZE; i++) {  
 if( arr[i] == value)   
 return i;  
 }  
 throw ValueNotFoundException();  
 }  
20.  
 template <class T>   
 void printArray( T arr[], const unsigned SIZE) {  
 for(unsigned i = 0; i < SIZE; i++) {  
 cout << i + 1 : arr[i] << endl;  
 }  
 }