

Homework 13 - Exceptions, simple and sortable vectors

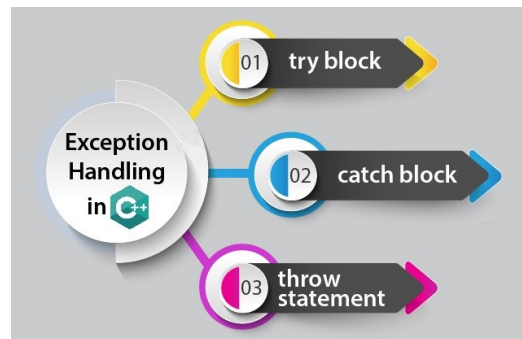
[New Attempt](#)

Due Monday by 11:59pm **Points** 30 **Submitting** a file upload

Advanced C++ Programming

Module 13 – Exceptions and the STL

Homework Exercises



Summary

1. Review Questions #1, 2, 4, 5, 7, 8, 11, and 12 from chapter 16 of our textbook (9th edition: pages 1032 - 1033, 10th ed: 1030)
1. Programming Challenges 1, 7, and 9 (extra credit) (9th edition: pages 1034 - 1035, 10th ed: 1032-1033)

Please remember to add comments throughout your code, and place each programming project in its own program file or files.

Review Questions (10 points)

1. The line containing a throw statement is known as the _____ .
2. The _____ block should enclose code that directly or indirectly might cause an exception to be thrown.
4. When writing function or class templates , you use a(n) _____ to specify a generic data type.

5. The beginning of a template is marked by a(n) _____ .
7. A(n) _____ container organizes data in a sequential fashion similar to an array.
8. A(n) _____ container uses keys to rapidly access elements.
11. Write a function template that takes a generic array of a given size as a parameter and reverses the order of the elements in the *same* array. The first parameter of the function should be the array, and the second parameter should be the size of the array.
12. Write a function template that is capable of adding any two numeric values and returning the result.

Project 1 – String Bound Exceptions (10 points)

Programming Challenge #1: Write a class named **BCheckString** that is derived from the STL (Standard Template Library) **string** class. This new class should have two member functions:

1. A **BCheckString (string s)** constructor that receives a string object passed by value and passes it on to the base class constructor.
2. A **char operator [] (int k)** function that throws a **BoundsException** object if **k** is negative or is greater than or equal to the length of the string. If **k** is within the bounds of the string, this function will return the character at position **k** in the string.

Write the definition of the **BoundsException** class. Make this class an inner class to the **BCheckString** class. Test your class with a **main** function which attempts to access characters that are within suitably initialized **BCheckString** object. Also demonstrate the exceptions thrown when a negative index and when an index that is too large are used.

Remember that objects of the **BCheckString** class are automatically strings themselves...

Project 2 – SimpleVector Modification (10 points)

Programming Challenge #7: Modify this [SimpleVector \(https://miracosta.instructure.com/courses/31330/files/7816500?wrap=1\)](https://miracosta.instructure.com/courses/31330/files/7816500?wrap=1) ↓ (https://miracosta.instructure.com/courses/31330/files/7816500/download?download_frd=1) class template to include the member functions **push_back** and **pop_back**. (A link to the template is also available at the bottom of this lab.) These functions should emulate the STL **vector** class member functions of the same name as described below.

You do not need to increase the size of the underlying array when needed. Instead throw exceptions:

- The constructor should accept an **int** parameter which is the number of elements in the underlying array (known as the *capacity*), and should set the *size* of the **SimpleVector** to

0, meaning that it currently holds no elements.

- The **push_back** function should throw an exception if the array is full (*size* of the vector equals the *capacity* of the array). Otherwise it accepts an argument and inserts that element into the next available position in the array, increasing the size by 1.
- The **pop_back** function has no parameter and removes the last element from the array, decrements the *size*, and returns the value popped from the vector. It should throw an exception if the *size* is 0, meaning that the vector is empty.

When handling an exception thrown in the **push_back** or **pop_back** functions, handle that exception in **main**, not in the **push_back** or **pop_back** functions or in the **SimpleVector** class.

Test the class with a tester program using a **SimpleVector** of numbers, then pushing and popping several numbers. For example, you could

1. Create a vector of **ints** with a capacity of 5.
2. Try to push 6 numbers (an exception should be thrown on the 6th number).
3. Pop the vector until it is empty (when it throws another exception).

Demonstrate the same thing with a **SimpleVector** of strings.

Don't forget to demonstrate both exception conditions.

Project 3 – **SortableVector** Class Template (10 points extra credit)

Programming Challenge #9: Write a class template named **SortableVector**. The class should be a child of the modified class from Project 2 above.

```
template <class T>
class SortableVector : public SimpleVector<T> . . .
```

It should have a member function that sorts the array elements in ascending order. (Use the sorting algorithm of your choice.)

The new class also should overload the [] operator. If the calling program passes an argument which is less than 0 or greater than or equal to the size, then throw an exception. Catch the exception in **main**.

Test the template in a driver program which sorts a vector of 10 **doubles**, and then sort a vector of 10 or more first names of family members and/or friends. Finally demonstrate what happens when using the [] operator with a negative argument and one larger than the number of elements in a **SortedVector**.

In the output for each of these exercises, please tell the user what you are demonstrating, rather than


simply printing numbers and strings.

Submit all program files, screen snips of the output of your programs, and a document with the answers to the Review Questions.

Links

Additional Files and Programs

[SimpleVector.h](#)

(<https://miracosta.instructure.com/courses/31330/files/7816500/download?wrap=1>) 
(https://miracosta.instructure.com/courses/31330/files/7816500/download?download_frd=1)

Next Assignment

[Homework 14](#)

(<https://miracosta.instructure.com/courses/31330/assignments/842800>)

Lab for this Module

[Lab 13 - Exceptions and Templates](#)

(<https://miracosta.instructure.com/courses/31330/assignments/842811>)

Prior Assignment

[Homework 12](#)

(<https://miracosta.instructure.com/courses/31330/assignments/842798>)