Project 5 – Vector Computations

# Overview

In this project, you will:

* Practice basic C++ syntax including branching structures
* Write classes and instantiate those classes using a constructor
* Create a templated data structure (queue or stack)
* Use simple file input
* Use overloaded operators to access templated data structure

# Background

For this project, we are going to be looking at another programming language named R. R is a programming language and free software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing. The R language is widely used among statisticians and data miners for developing statistical software and data analysis.

For our project, we are going to use C++ to build a templated data structure similar to one found in R named a Vector. For our purposes, we will build a templated linked list to hold the data to do some vector computations.

# Description

**Class 1 – Vector** – This is a very important class. This is the first class you should write. The **Vector** uses a listed list to store all of the data. It is a full data structure that allows users to insert (to the end) and remove from anywhere in the linked list. Additionally, it allows for users to insert into a sorted structure, display, return the size, and check to see if it is empty. It also has a copy constructor and overloaded assignment operator (which you must implement and test!). As it is templated, it should be able to hold just about anything (some types of dynamic objects might gum it up). There are several overloaded operators including +, \*, <, ==. Additionally, it has several statistical functions including mean, median, and standard deviation. You must implement all functions defined in order to earn all of the points for the project.

Hint: To call the overloaded operator[](), you can use: **operator[](i)**

**Class 2: Runner –** This is the class that tests the Vectors. A runner is made up of two Vectors. There are only two functions – a constructor for the Runner and a menu. The menu has the ability to do the following:

1. Display Vectors

2. Vector comparison (<)

3. Vector comparison (==)

4. Vector addition

5. Vector multiplication

6. Compute median

7. Compute mean

8. Compute Standard Derivation

9. Exit

Display just displays both Vectors in the runner.

Vector comparisons (< and ==) display the results of the comparison between Vector 1 and Vector 2. So, the first item in Vector 1 would be compared to the first item in Vector 2 and so on for the entire size of the Vector. Assume all vectors will be the same size.

Vector arithmetic (+ and \*) display the results of the addition or multiplication with Vector 1 and Vector 2. So, add or multiple the first item in Vector 1 and the first item in Vector 2.

For the three computations (median, mean, and standard deviation), you would calculate the result individually for each Vector. So you would display the median for all of Vector 1. On the second line you would display the median for all of Vector 2.

**Hint:** For some of the Runner functions, the vector comparisons (< and ==) and the vector addition and multiplication, the functions return a “result” Vector of type char in some cases. The Vector is then displayed. Do not forget to deallocate the Vector when you are done using it!

# Requirements:

This is a list of the requirements of this application. You **MUST** meet all the defined requirements.

* YOU CAN ONLY USE THE LIBRARIES ALLOWED
* The project must be completed in C++. You may not use any libraries or data structures that we have not learned in class. Libraries we have learned include **<iostream>, <fstream>, <iomanip>, <vector>, <cmath>, <ctime>, <cstdlib>,** and **<string>**. You should only use **namespace std**.   
    
  Do not use **<vector>** or **<list>** for this project.
* Using the provided files, **Vector.cpp, Runner.cpp, makefile, data1.txt, data2.txt, data3.txt, data4.txt and the proj5.cpp file** write the program as described. (Finish Vector first though!)
* As a reminder, **Vector.cpp and Runner.cpp**  are templated and all functions must exist in ONE file.
* I have attached all the files you will need
* All user input must be validated. For example, if a menu allows for 1, 2, or 3 to be entered and the user enters a 4, it will re-prompt the user. However, the user is expected to always enter the correct data type. i.e. If the user is asked to enter an integer, they will. If they are asked to enter a character, they will. You do not need to worry about checking for correct data types.
* **The code must not have memory leaks or errors.**

# Recommendations

* We will test your **Vector** file separately from your application (as well as part of it). It must be fully implemented with a destructor, copy constructor, and an assignment operator. Test it first using something simple like ints. Remember the special cases such as the structure being empty.
* After the **Vector** works (including all tests), write **Runner** (it is relatively easy). **Vector** is tricky though.

# Sample Input and Output

To start with the testing of the **Vector** file, you can alternate writing functions (starting with the constructor and **Insert**). Here is a full run with 1-9 for the first data set.

|  |
| --- |
| Vector Computations  Choose an option  1. Display Vectors  2. Vector comparison (<)  3. Vector comparison (==)  4. Vector addition  5. Vector multiplication  6. Compute median  7. Compute mean  8. Compute Standard Derivation  9. Exit  1  1  Vector 1: -15.9, 7.17, 10.44, 14.4, 15.011, 16.12, 18.531, 53.44, 119.111, 122.12  Vector 2: -16.1, -11.22, 10.44, 11.211, 15.011, 19.1241, 95.1, 102.33, 107.66, 117.12  Choose an option  1. Display Vectors  2. Vector comparison (<)  3. Vector comparison (==)  4. Vector addition  5. Vector multiplication  6. Compute median  7. Compute mean  8. Compute Standard Derivation  9. Exit  2  2  Vector1 < Vector2: F, F, F, F, F, T, T, T, F, F  Choose an option  1. Display Vectors  2. Vector comparison (<)  3. Vector comparison (==)  4. Vector addition  5. Vector multiplication  6. Compute median  7. Compute mean  8. Compute Standard Derivation  9. Exit  3  Vector1 == Vector2: F, F, T, F, T, F, F, F, F, F  Choose an option  1. Display Vectors  2. Vector comparison (<)  3. Vector comparison (==)  4. Vector addition  5. Vector multiplication  6. Compute median  7. Compute mean  8. Compute Standard Derivation  9. Exit  4  Vector1 + Vector2: -32, -4.05, 20.88, 25.611, 30.022, 35.2441, 113.631, 155.77, 226.771, 239.24  Choose an option  1. Display Vectors  2. Vector comparison (<)  3. Vector comparison (==)  4. Vector addition  5. Vector multiplication  6. Compute median  7. Compute mean  8. Compute Standard Derivation  9. Exit  5  Vector1 \* Vector2: 255.99, -80.4474, 108.994, 161.438, 225.33, 308.281, 1762.3, 5468.52, 12823.5, 14302.7  Choose an option  1. Display Vectors  2. Vector comparison (<)  3. Vector comparison (==)  4. Vector addition  5. Vector multiplication  6. Compute median  7. Compute mean  8. Compute Standard Derivation  9. Exit  6  Vector1 median: 17.3255  Vector2 median: 57.1121  Choose an option  1. Display Vectors  2. Vector comparison (<)  3. Vector comparison (==)  4. Vector addition  5. Vector multiplication  6. Compute median  7. Compute mean  8. Compute Standard Derivation  9. Exit  7  Vector1 mean: 36.0443  Vector2 mean: 45.0676  Choose an option  1. Display Vectors  2. Vector comparison (<)  3. Vector comparison (==)  4. Vector addition  5. Vector multiplication  6. Compute median  7. Compute mean  8. Compute Standard Derivation  9. Exit  8  Vector1 stdev: 45.1816  Vector2 stdev: 50.718  Choose an option  1. Display Vectors  2. Vector comparison (<)  3. Vector comparison (==)  4. Vector addition  5. Vector multiplication  6. Compute median  7. Compute mean  8. Compute Standard Derivation  9. Exit  9  Thank you for using the Vector Tool |

All user entries are in blue. This shows 1,3,5,7, and 9 with two user validations of 0 and 10. Here is a valgrind sample run from that:

|  |
| --- |
| valgrind ./proj5 data1.txt  ==3850331== Memcheck, a memory error detector  ==3850331== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.  ==3850331== Using Valgrind-3.16.0 and LibVEX; rerun with -h for copyright info  ==3850331== Command: ./proj5 data1.txt  ==3850331==  Vector Computations  Choose an option  1. Display Vectors  2. Vector comparison (<)  3. Vector comparison (==)  4. Vector addition  5. Vector multiplication  6. Compute median  7. Compute mean  8. Compute Standard Derivation  9. Exit  1  Vector 1: -15.9, 7.17, 10.44, 14.4, 15.011, 16.12, 18.531, 53.44, 119.111, 122.12  Vector 2: -16.1, -11.22, 10.44, 11.211, 15.011, 19.1241, 95.1, 102.33, 107.66, 117.12  Choose an option  1. Display Vectors  2. Vector comparison (<)  3. Vector comparison (==)  4. Vector addition  5. Vector multiplication  6. Compute median  7. Compute mean  8. Compute Standard Derivation  9. Exit  3  Vector1 == Vector2: F, F, T, F, T, F, F, F, F, F  Choose an option  1. Display Vectors  2. Vector comparison (<)  3. Vector comparison (==)  4. Vector addition  5. Vector multiplication  6. Compute median  7. Compute mean  8. Compute Standard Derivation  9. Exit  5  Vector1 \* Vector2: 255.99, -80.4474, 108.994, 161.438, 225.33, 308.281, 1762.3, 5468.52, 12823.5, 14302.7  Choose an option  1. Display Vectors  2. Vector comparison (<)  3. Vector comparison (==)  4. Vector addition  5. Vector multiplication  6. Compute median  7. Compute mean  8. Compute Standard Derivation  9. Exit  7  Vector1 mean: 36.0443  Vector2 mean: 45.0676  Choose an option  1. Display Vectors  2. Vector comparison (<)  3. Vector comparison (==)  4. Vector addition  5. Vector multiplication  6. Compute median  7. Compute mean  8. Compute Standard Derivation  9. Exit  0  Choose an option  1. Display Vectors  2. Vector comparison (<)  3. Vector comparison (==)  4. Vector addition  5. Vector multiplication  6. Compute median  7. Compute mean  8. Compute Standard Derivation  9. Exit  10  Choose an option  1. Display Vectors  2. Vector comparison (<)  3. Vector comparison (==)  4. Vector addition  5. Vector multiplication  6. Compute median  7. Compute mean  8. Compute Standard Derivation  9. Exit  9  Thank you for using the Vector Tool  ==3850331==  ==3850331== HEAP SUMMARY:  ==3850331== in use at exit: 0 bytes in 0 blocks  ==3850331== total heap usage: 71 allocs, 71 frees, 84,424 bytes allocated  ==3850331==  ==3850331== All heap blocks were freed -- no leaks are possible  ==3850331==  ==3850331== For lists of detected and suppressed errors, rerun with: -s  ==3850331== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0) |

There is a full run of all four test runs available at **proj5\_sample1.txt**

# Compiling and Running

You will need to implement the **makefile** for this project.

Once you have compiled using the **makefile**, enter the command **make run** or **./proj5**  to run your program. If your executable is not proj5, you will lose points. It should look like the sample output provided above.

Because we are using dynamic memory for this project, you are required to manage any memory leaks that might be created. Anything that you use “new” for needs to be deleted. Remember, in general, for each item that is dynamically created, it should be deleted using a destructor.

One way to test to make sure that you have successfully removed any of the memory leaks is to use the **valgrind** command.

Since this project makes extensive use of dynamic memory, it is important that you test your program for memory leaks using **valgrind**:

**valgrind ./proj5**

Note: If you accidently use valgrind make run, you may end up with some memory that is still reachable. Do not test this – test using the command above where you include the input file.

If you have no memory leaks, you should see output like the following:

|  |
| --- |
| ==5606==  ==5606== HEAP SUMMARY:  ==5606== in use at exit: 0 bytes in 0 blocks  ==5606== total heap usage: 87 allocs, 87 frees, 10,684 bytes allocated  ==5606==  ==5606== All heap blocks were freed -- no leaks are possible  ==5606==  ==5606== For counts of detected and suppressed errors, rerun with: -v  ==5606== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 6 from 6) |

The important part is “in use at exit: 0 bytes 0 blocks,” which tells me all the dynamic memory was deleted before the program exited. If you see anything other than "0 bytes 0 blocks" there is probably an error in one of your destructors. We will evaluate this as part of the grading for this project.

Additional information on **valgrind** can be found here: <http://valgrind.org/docs/manual/quick-start.html>