Assignment 6 - Object Oriented Programming

Due - 9PM, Aug 15th 2014

1 Managing Dynamic Arrays

Managing dynamically sized arrays can be a pain while dealing with memory on the heap. Memory leaks, dangling pointers, etc can lead to invalid memory accesses/segmentation faults that halt your program abruptly. Your task is to design a **DataVector** class that encapsulates the implementation of a dynamically sized double array. The interface should be such that the users of your class should *not* be exposed to the underlying implementation details such as memory allocation (new), release (delete), resize, etc. Your **DataVector** class must have the following public interface. The difficulty levels are marked as $L1$, $L2$ and $L3$, going from easy to hard.

1.
```
DataVector(); //L2
```
   This is the default constructor with no parameters. By default, this allocates space for a double array of size 10 and assigns a default value of 0 to each of them.

2.
```
DataVector(unsigned int initSize, double initValue);//L2
```
   This is another constructor of the class that takes in arguments and allocates space for a double array of size *initSize* and assigns a default value of *initValue* to each of them.

3.
```
~DataVector();//L2
```
   This is the destructor that does any cleanup necessary such as releasing all the memory on the heap using the *delete* operator.

4.
```
void PrintItems();//L2
```
   This prints all the items in a single line and successive items are separated by a single space. After the last item there should be a space and a newline character.

5.
```
unsigned int GetSize();//L1
```
   This should return the current size of the double array.

6.
```
void Reserve(unsigned int newSize);//L3
```
   This function should increase the size of the array to *newSize*. Note that you would have to allocate a fresh double array, to which you will have to copy the items from the old double array, and then release the memory for the old double array. If *newSize* is less than the old size, then you copy just the first *newSize* items from the old array. If *newSize* is greater than the old size, then you copy all items from the old array.

7. `double GetItemAt(unsigned int index);//L1`

   This returns the item at the specified *index*. Assume *index* is in the range 0 to *GetSize() - 1*.

8. `void SetItemAt(unsigned int index, double val);//L1`

   This sets the item at the specified *index* to *val*. Assume *index* is in the range 0 to *GetSize() - 1*.

9. `double GetSum();//L2`

   This returns the sum of all items.

Note that you must use *new* and *delete* to manage the double array internally. All pointers, counters, etc for the array bookkeeping must be made private in your class.

2 Grade Management

Now you will apply your **DataVector** class to manage the student grades of CS31. You will implement a **GradeManager** class that internally uses an array of **DataVector** objects, that were created dynamically using the *new* operator, to record the homework grades for a set of students. Again, the idea is to encapsulate the underlying implementation such that the user of **GradeManager** will not know that **DataVector** is being used internally. Your **GradeManager** class must have the following public interface:

1. `GradeManager(unsigned int nStudents, unsigned int nHWs);//L3`

   This is the only constructor of the class, and it specifies the number of students *nStudents* and number of homeworks *nHWs* for the class. You should use these to dynamically set the array sizes. Also, if *nStudents* is 100 then the student IDs will range from 0 to 99. Similarly, if *nHWs* is 10 then the homework IDs will range from 0 to 9.

2. `~GradeManager();//L2`

   This is the destructor that does any cleanup necessary such as releasing all the memory on the heap using the *delete* operator.

3. `void PrintGrades();//L2`

   This prints all the homework grades for all students. Each student's grades are printed on a separate line, and the successive homework grades of each student are separated by a single space. After the last grade on a line, there should be a space and a newline character.

4. `unsigned int GetClassSize();//L1`

   Returns the number of students.

5. `unsigned int GetHWCount();//L1`

   Returns the number of homeworks.

6. `void SetGrade(unsigned int sID, unsigned int hwID, double val);//L2`

   This sets the grade of student *sID* to *val* for the homework *hwID*.

7. `double GetGrade(unsigned int sID, unsigned int hwID);//L2`

   This returns the grade of student *sID* for the homework *hwID*.

8. `double GetTotalScoreForStudent(unsigned int sID);//L2`

   This returns the sum of the respective student's scores on all homeworks.

9. `double GetTotalScoreForStudentWithDrop2Least(unsigned int sID);//L3`

   This returns the sum of the respective student's scores on all homeworks except the least 2 homeworks. Henceforth, this is referred to as the *Drop2* policy.

10. `unsigned int GetBestStudent();//L2`

    This returns the *sID* of the student who has the best total score across homeworks with the *Drop2* policy.

11. `unsigned int GetNumStudentsInRange(double low, double high);//L3`

    This returns the number of students whose total homework scores (with *Drop2* policy) lie in the range $low <= totalscore < high$. (Including *low* and excluding *high*).

12. `double GetClassAverage();//L2`

    This returns the average of the total homework scores (with *Drop2* policy), across all students.

13. `double GetClassSTD();//L2`

    This returns the population standard deviation of the total homework scores (with *Drop2* policy), across all students. The standard deviation of a set of n total scores $t_1, t_2, t_3 ... t_n$ whose average is m, is given by,

    $$\sqrt[2]{\frac{(t_1 - m)^2 + (t_2 - m)^2 + (t_3 - m)^2 + ... + (t_n - m)^2}{n}}$$

    You can use the standard library *sqrt()* function from *cmath* header to find the square root.

3 Dos/Donts/General Advice

1. All your code should be written inside the two classes **DataVector** and **GradeManager**. It is recommended that you do not modify anything outside these classes (*main()* function especially). But if you really want to write your own test cases in the *main()* function, then it is *your* responsibility to revert those changes before submitting on the server.

2. A set of basic tests that call into the public inferface on the 2 classes is given to you in the *main()* function. These tests might *not* be complete, and if your output matches the output given at the end of this document, then it does *not* necessarily mean that you will get full credit on the assignment. We will be designing new test cases for grading purposes, and hence, it is your responsibility to make sure the implementation works as expected. Nevertheless, these tests may give you some ideas on how to use the public interface.

3. It is recommended to start implementing the easy functions first. You can also follow the order listed above. This order will help you build the class one step at the time, and some of the earlier functions can be called/reused in later functions if needed.

4. Reusability is one of the main objectives of Object Oriented Programming. So, feel free to reuse any function/code you might have already written.

5. You might run into *Segmentation Faults* (a.k.a *SIGSEGV* on unix systems), or *Invalid memory access/Bad access* or *Protection faults*, or any kind of memory corruption issues. If you do, then the following checks might help.

   - Check if your indexes to the dynamically allocated arrays are within the legal range. For example, if you allocated a 10 element *double* array using the syntax *new double[10]*, then make sure your index values to this array are in the range $0 - 9$.
   - Check if you have any *dangling pointers*. These are pointers that hold addresses to memory that have already been released using the *delete* operator. It is usually a good practice to set your pointer to *NULL* after you have called *delete* on it. For example,

     ```
     delete[] ptr;
     ptr = 0;
     ```

   - Check if you are dereferencing a *NULL* pointer.

4 What to turn in

Guidelines on where to submit the project will be available 48 hours before the assignment. Your submission file must be in the following format: the completed *oop.cpp* must be compressed into a single .zip file. The ZIP file name must be in the following format:

```
StudentID_proj6.zip
```

For instance if my student ID is 123456789 and I am submitting my solution for assignment 6, then I am going to compress *oop.cpp* file and rename the zip file to:

```
123456789_proj6.zip
```

You should only include the *oop.cpp* file that you modified into the zip file. Do not submit/include any other file such as the executable or the contents of the debug folder. Same as other assignments, a good sanity check is to check your zip file for corruption by extracting it and testing whether it did compress it successfully.

5 Solution Output

After implementing the above 2 classes, the given *main()* function in *oop.cpp* should output the following. This is also contained in the *sampleout.txt* text file given to you with the starter code.

```
DataTest 1
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00


DataTest 2
```

```
3.56 3.56 3.56 3.56 3.56 3.56 3.56 3.56 3.56 3.56 3.56 3.56 3.56 3.56

DataTest 3
130

DataTest 4
17
5
3.56 3.56 3.56 3.56 3.56

DataTest 5
3.56 3.56 3.56 3.56 3.56 1.11 3.56 3.56 3.56 3.56 3.56 3.56 3.56 3.99 5.66

DataTest 6
88.50

GradeTest 1
10 9

GradeTest 2
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

GradeTest 3
67.00 34.00 56.00 64.00 87.00
78.00 66.00 91.00 79.00 90.00
45.00 59.00 91.00 99.00 100.00
70.00 73.00 54.00 89.00 33.00
89.00 81.00 67.00 95.00 98.00

GradeTest 4
67.00 34.00 56.00 64.00 87.00
78.00 66.00 91.00 79.00 90.00
45.00 59.00 91.00 99.00 100.00
70.00 73.00 54.00 89.00 33.00
89.00 81.00 67.00 95.00 98.00

GradeTest 5
308.00 218.00
404.00 260.00
```

```
394.00 290.00
319.00 232.00
430.00 282.00
2

GradeTest 6
[0.00,50.00] - 0
[50.00,100.00] - 0
[100.00,150.00] - 1
[150.00,200.00] - 1
[200.00,250.00] - 1
[250.00,300.00] - 2

GradeTest 7
4
247.45
21.63
```