Assignment 1

Submission date: 9th, April 2021 **Submission time:** 2359hrs

Note: Submission should be on LMS. Assignment submission through email will not be accepted. Submit your complete code in .docx file. Late Submission will not be marked.

Do your own work. Any kind of plagiarism found will result in negative marks

Design a templated data holder '**DynamicDataHolder**' capable to hold a dynamic data holder of any type If you attempt to add an element to the DynamicDataHolder's object and there is no more room, then a new dynamic array with twice the capacity of the old dynamic array is created and the values of the old dynamic array are copied to the new dynamic array. Your data type should have following functionalities.

```
void main()
{
DynamicDataHolder <int> obj;    // declare an object holding array of any default size
DynamicDataHolder <char> obj2(5); // declare an object holding array of size 5
DynamicDataHolder <char> obj3=obj2; // creates copy
cout<< (obj3==obj2); // compares two objects with respect to their data holder values and returns
                     //true if both are same
for(int i=0; i<10; i++)
{
  obj.insert(i*2);  // every time insert function get called, it should insert value at the end of data holder
}

obj.capacity() ; // should return the number of empty spaces in data holder
obj.size() ; // should return the size of dataHolder
obj.resize(15) ; // should resize the dataHolder
cout<< obj2[4]; // should display 5th element of data holder
cin>> obj2[10]; // should take input for 10th element of data holder, here the size of data holder is 5, so it should be //
                catered before taking the input
DynamicDataHolder <int> obj4;
obj4=obj;          // should assign values of obj to obj4

}
```

Note: There should be no memory leakage, dangling pointers and shallow copies

To test your code, copy paste the main function and attach your output along with your code.

Assessment Rubrics

Trait	Exceptional [10-8]	Acceptable [7-6]	Amateur [5-3]	Unsatisfactory [2-0]
Functionality 15%	Application compiles with no warnings. Robust operation of the application, with good recovery.	Application compiles with few or no warnings. Consideration given to unusual conditions with reasonable recovery	Application compiles and runs without crashing. Some attempt at detecting and correcting errors.	Application does not compile or compiles but crashes. Confusing. Little or no error detection or correction.
Specification & Efficient implementation 70%	The program works and meets all of the specifications. Well-designed and Excellent implementation of data structure. Providing all the required functions that might be useful and aids in performing the operations on data structure. The code is extremely efficient without sacrificing readability and understanding. No memory Leakage, Dangling pointers and bad pointers	The program works and produces the correct results and displays them correctly. It also meets most of the other specifications. Good implementation of data structure. The code is fairly efficient without sacrificing readability and understanding. Small memory Leakage, dangling pointers or bad pointers	The program produces correct results but does not display them correctly. Fair implementation of data structure. The code is brute force and unnecessarily long. Inefficiently using pointers & dynamic memory.	The program is producing incorrect results. Worst implementation of data structure. The code is huge and appears to be patched together and hasn't managed any memory leaks, dangling pointers and bad pointers.
Input Validation 15%	All the inputs entered by user are properly validated, a message displayed to user in case of wrong input entered by user and prompt till user enters the correct input.	Error Message displayed to user in case of in correct input entered by user. The program continues execution without prompting again till the correct input is entered	Some of the inputs are validated leaving the others.	No input validation.