DigiPen
INSTITUTE OF TECHNOLOGY

Midterm Programming Exam

Module 2 – High-level Programming II

Purpose

This exam is testing you on most concepts we covered so far in class in object-oriented design and coding (classes, objects, constructors, operator overloading, friend functions).

Information

The task is to define and implement a class called ***myArray***, which at its most basic level, this class is simply a wrapper around a dynamically-allocated array. The ***myArray*** class shields the user from dealing with the allocations and deallocations of memory. It provides many conveniences that a built-in array doesn't provide, such as automatic resizing and array addition.

Some code is already provided for you:

```
#pragma once
#include <iostream>

class myArray
{
public:
    myArray(void);

    int get_capacity(void) const;
    int get_size(void) const;

private:
    int *numbers;
    int size;
    int capacity;
};
```

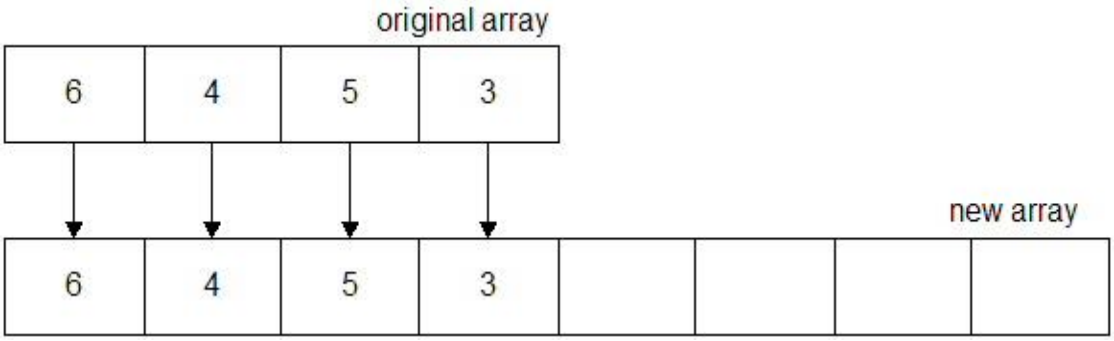
Member data **Description**
`int *numbers;`
Member array that will hold all the integers. This array will be managed by the class methods. The array will grow if needed.
`int size;`
This integer represents how many integers we currently have in the array.
`int capacity;`
This integer represents the size of the array. In other words, how many integers we can totally hold before we have to grow our array. ***PS: Every time we grow the array we will double its capacity.***

Example:

Assume we have an array that is full and its capacity is 4.

If we would like to add a fifth element, we will first need to create a new array that hold 8 elements

Then copy all the elements from the original array to the new array



And finally, delete the original array and add the new integer to the new_array after the value 3. At the end of all that, the "size" value will be 5 and "capacity" will be 8.

```
myArray(void);
```

This is the class' default constructor (already implemented for you). By default, the numbers array will be dynamically allocated and its size will be 1. Since no integers are stored in it yet, the size value will be 0 and the capacity will be 1.

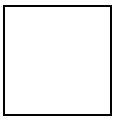
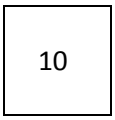
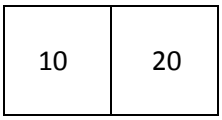
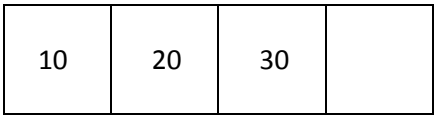
```
int get_capacity(void) const;
  int get_size(void) const;
```

Two member functions used as gettors. ***get_capacity*** returns the capacity value, and ***get_size*** returns the size value.

Your task is to create all the required methods in order to pass the 4 tests that I provided in ***main.cpp***.

Now, I will describe some of the required methods:

Member functions **Description**
`void push(int value_);`

This methods adds an integer, represented by the ***value_*** parameter, to the end of the array. If the array does not have a place for the new integer, you will have to grow the array by 2 and then place the given integer value.
 	
Example:
 	
`myArray a;`
 
size = 0
capacity = 1
 	
`a.push(10);`
 
size = 1
capacity = 1
 	
`a.push(20);`
 
size = 2
capacity = 2
 	
`a.push(30);`
 
size = 3
capacity = 4

```
void pop();
```

This method removes an the last integer in the array. Removing an integer is done by simply decreasing the size value of the array.

Example:

```
myArray a;
a.push(10);
a.push(20);
a.push(30);
```

10	20	30	

size = 3
capacity = 4

```
a.pop();
```

10	20	30	

size = 2
capacity = 4

As you can see, we simply decreased the value of size by one. The next time an integer is pushed, its value will replace 30.

```
a.push(25);
```

10	20	25	

size = 3
capacity = 4

```
operator+
```

This overloaded operator allows us to add two ***myArray*** instances. The result will contain the added value of the element of instance one with the elements of instance two respectively. The **size** and the **capacity** of the returned ***myArray*** instance will be the same as the ***myArray*** instance that has the largest **size**.

Example:

```
myArray a;
a.push(10);
a.push(20);
```

a

10	20

size = 2
capacity = 2

```
myArray b;
b.push(1);
b.push(2);
b.push(3);
```

b

1	2	3	

size = 3
capacity = 4

```
myArray c = a + b;
```

b

11	22	3	

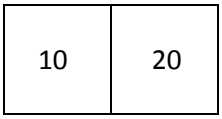
size = 3
capacity = 4

`operator+=`

This overloaded operator allows us to add the elements of the right hand side ***myArray*** instance to the elements of the left hand side ***myArray*** instance. If the right hand side's size is greater than the left hand side's size, you will need to resize your left hand side's array.

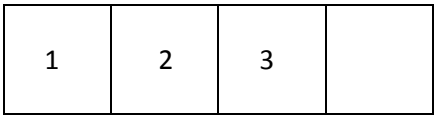
Example:

a



size = 2
capacity = 2

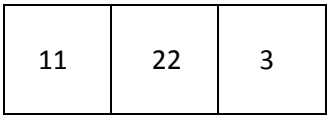
b



size = 3
capacity = 4

```
a += b;
```

a



size = 3
capacity = 3

Testing your code

A "Grading Scripts" folder is provided in order for you to check if your implementation is correct. In order to run the tests follow these steps:

- grab your **myArray.h** and **myArray.cpp** files (which has all of your implementation) and place them in the **"Grading Scripts**" folder.
- Double click on the **"DigiPen - RunAll.cmd"** file.
    - All tests will run automatically. If a test fails, the script will stop at that test and you will get a message on the console that explains which test failed and why.

      NOTE: ***If you are working on a personal machine you need to run the "RunAll.cmd" file instead of "DigiPen - RunAll.cmd"***

What to submit

You must submit two files ***(myArray.h, myArray.cpp)*** in a single .zip file (go to the class moodle page and you will find the submition link).

Do not submit any other files than the ones listed.