**C++ Workshop – 150018**

**Homework Assignment #2**

**Classes with Dynamic Memory**

**General:**

* Pay attention to readability and indentation, of program.
* Do exactly what you are asked for in each question.
* Define function according to need.
* Don't forget meaningful names for variables, document code and functions, and add running examples for your program at end.
* Homework may be done in teams of two.
* You have one week to give in your homework.

**Question 1**

* 1. Write a class that represents a point on a plane.

The class should include the following attributes:

* x – a whole number which represents the value on the x- axis.
* y – a whole number which represents the value on the y-axis.

The class should include the following constructors:

* an **empty constructor** – which initializes the x and y attributes to zero (you are defining a point at the origin)
* a regular **constructor** – which will receive two parameters and initialize the attributes x and y accordingly
* a **copy constructor**
* (*food for thought – why don’t we define a destructor?*)

The class should include the following methods:

* Setters for all of the attributes
* Getters for all of the attributes
* A method that computes the distance between two points.
  1. Write a class that represents a planar polygon.

The class should include the following attributes:

* + - An array of vertices (points as defined in a).
    - The number of vertices in the polygon.

The class should include the following constructors:

* an **empty constructor** – which initializes the array of points to *nullptr*and the number of points to zero.
* a regular **constructor** – which will receive one parameter which represents the number of points of the polygon. You are to build the array of points accordingly.
* a **copy constructor**
* a **destructor**

The class should include the following methods:

* Getters for all of the attributes
* (*food for thought – why don’t we define setters?*)
* the method *addPoint* that receives an index and a vertex (a parameter of type Point) and adds the vertex to the array of vertices at the index specified by the first parameter.
* A method to compute the perimeter of the polygon (sum of the distances between adjacent vertices).You can assume the vertices are ordered according to how they appear on the polygon. (Do not forget, you must also sum the distance between the last point and the first point.) Have your function return a real number (float or double).
* A boolean method that will receive as a parameter another polygon. The method will check whether the polygon that called the method is equal to the polygon that was received as a parameter. Polygons are defined as equal when they have the same number of vertices and same values for the vertices (not necessarily in the same order in the array).  
  For example the polygon (0,0) (1,1) (2,0) is equal to (1,1) (2,0) (0,0).
  1. To help you understand how constructors and destructors work, you will add the following outputs to your constructors and destructors:
* The empty constructor should print: in empty constructor
* The regular constructor should print:

in one parameter constructor

* The copy constructor should print: in copy constructor
* The destructor should print: in destructor
  1. Write a main program that reads in the data of two polygons and prints their perimeters rounded to the nearest integer value (you may use the round function that is found in the cmath library.)

Printing of the perimeter(s) should be done in the following manner:  
 if they are equal print equal and the perimeter, otherwise print not equal and the perimeter of each polygon.

* 1. In order to read in the values for each polygon, the program will first print: enter number of sides, and then read in the number,
  2. Afterwards, the program will print enter the point values, and then read in the list of point values.  
     The format of the input is:

(x1, y1) (x2, y2) ….. (xn, yn)  
where ***n*** is the number of vertices and ***xi*** and ***y***i are the coordinates of the ***i***th vertex of the polygon. If there is an error in the input, the your program should print ERROR

Example 1 (square and triangle):

enter number of sides:  
4  
in one parameter constructor

enter the point values:

(0,0) (0,2) (2,2) (2,0)

enter number of sides:  
3  
in one parameter constructor

enter the point values:

(1,1) (2,0) (3,1)

in copy constructor

in destructor

not equal

perimeter: 8

perimeter: 5  
in destructor

in destructor

Example (3-4-5 triangle):

enter number of sides:  
3  
in one parameter constructor

enter the point values:

(10,10) (10,14) (13,10)  
enter number of sides:  
3  
in one parameter constructor

enter the point values:

(13,10) (10,10) (10,14)

in copy constructor

in destructor

equal

perimeter: 12  
in destructor

in destructor

**Question 2**

1. Write a class **Vector** that implements an array of integers of any given size.

**Please Note: the class that you are writing for homework is slightly different then the class that you saw in the lecture. The size of the vector is always a power of 2.**

The places in the class where this difference is significant are:

* The constructor which receives a parameter which represents the capacity of the vector.
* The method *strnewcat* which will be used to concatenate two vectors.
* The method *insert* which will be used to insert elements into the vector.

When you build (using the constructor) and concatenate, you are going to in essence build a whole new array. If the capacity of the array that you wish to build is already a power of 2 then you are done; however, if it is not then you will need to find the closest number that is greater than the requested capacity but is a power of 2.

You can use the following formula to find this number:

capacity = pow(2, ceil(log(requested\_size) / log(2)))

where *requested\_size* is the capacity of the vector that was requested.

The method *insert* will be able to maintain a capacity which is a power of 2, simply by multiplying the capacity of the array by 2.

The class should include the following attributes:

* + data – a pointer to an array of integers.
  + capacity – the capacity of the array (maximum number of elements that can be stored in the array).
  + size – actual number of elements found in the vector

1. The class should include the following constructors/destructor:

* a regular **constructor -** which receives as a parameter the capacity of the vector. If it doesn’t receive a value then the constructor initializes the capacity of the vector to be 2. If it does receive a size, then the constructor should make sure that the capacity is indeed a power of 2. If it is not, then the constructor should set the capacity of the vector to be the number that is a power of 2 and is closest but not less the number that it received as a parameter.
* a **copy constructor**
* a **destructor**

1. The class should include the following methods:

* The method *getCapacity* which returns the capacity of the vector.
* The method *getSize* which returns the actual number of elements in the vector.
* (*food for thought – why don’t we define setters?*)
* The method *print* which prints the following: the capacity, the actual size, and the elements of the vector.  
  For example, for a vector of a capacity of 4 that contains the elements 5 and 19, the method would print:

capacity: 4 size: 2 val: 5 19

* The method *assign* (which acts like the operator =) to assign one vector to another. The method receives a Vector as a parameter and assigns its values to the Vector that called the method.
* The method s *isEqual*(which acts like the operator == ) which decides if two items are the same. The method receives a Vector as a parameter and returns true if the two Vectors (the one that called the method and the one that was received as a parameter) are the same and otherwise returns false.

The two vectors are considered to be the same if their sizes are equal (it **does not check capacity)** and if the values stored in each index are equal.

i.e. the following two arrays

[3, 5, 7, 9] , [5, 3, 7, 9]

are not equal.

* The method *at* (which acts like operator []) receives as a parameter an index for assigning and retrieving a value in the vector. If the requested index is not in range, then print ERROR and return first element.

The method’s signature is:

int& at(int index)

* The method *scalmul* which computes the scalar (dot) product of two vectors (i.e., the sum of the product of corresponding elements in the vectors). If the vectors do not have the same number of elements it should print ERROR and return -1.
* The method *strnewcat* for concatenation. (Concatenation means adding the contents of one item to the end of another item. i.e. If I concatenated the string *hi* with the string *there*, I would get the new string *hithere.* The method receives a Vector as a parameter. It will create a new Vector as follows:
  + The elements in the array of the new Vector will be the concatenation of the elements stored in the array of the Vector that called the method with the elements stored in the array of the Vector that was passed as a parameter (where the elements of the Vector that called the method will be stored first).
  + The size will obviously be the number of elements that are stored in the array
  + The capacity should ideally be the sum of the capacity of the Vector that called the method with the Vector that was passed as a parameter. However, since the capacity must be a power of 2, we need to make sure that the sum is indeed a power of 2. If not, then we need to find the closest number to the sum that is greater than the sum and is indeed a power of 2. The formula to do so appears at the beginning of this question
* The method *clear* which clears the **contents** of the vector
* The method *delLast* which deletes the last element of the vector. If the vector was empty, it prints **ERROR**
* The method *insert(int val)* which insert the value of the parameter in the next available location in the array. If the array was full, then it expands the array dynamically by doubling its capacity and copying the values from the old array to the new array and then finally deletes the old array.

Use the following main program to check the methods that you wrote.

#include "Vector.h"

#include <iostream>

using namespace std;

enum options

{

STOP, ASSIGN, IS\_EQUAL, SCALAR\_MULTIPLY , ADD, CLEAR, DELETE\_LAST, AT, INSERT

};

int main()

{

cout << "Test 1 - Constructors" << endl << endl;

Vector v1(4), v2(10), v3;

cout << "v1 capacity: " << v1.getCapacity()

<< " v1 size: " << v1.getSize()<< endl;

cout << "v2 capacity: " << v2.getCapacity()

<< " v2 size: " << v2.getSize() << endl;

cout << "v3 capacity: " << v3.getCapacity()

<< " v3 size: " << v3.getSize() << endl;

cout << endl << "Test 2 - Assign" << endl << endl;

for (int i = 1; i <= 4; i++)

{

v1.insert(i);

v2.insert(i);

v3.insert(i + 4);

}

cout << "v1 capacity: " << v1.getCapacity()

<< " v1 size: " << v1.getSize()<< endl;

cout << "v2 capacity: " << v2.getCapacity()

<< " v2 size: " << v2.getSize() << endl;

cout << "v3 capacity: " << v3.getCapacity()

<< " v3 size: " << v3.getSize() << endl;

int choice, val, index;

cout << endl << "Test 3 - Operations" << endl << endl;

cout << "enter your choice 0-8:\n";

cin >> choice;

while (choice)

{

switch (choice)

{

case ASSIGN: v3.assign(v1);

break;

case IS\_EQUAL:

if (v1.isEqual(v2))

cout << "v1==v2\n";

else

cout << "v1!=v2\n";

if (v1.isEqual(v3))

cout << "v1==v3\n";

else

cout << "v1!=v3\n";

break;

case SCALAR\_MULTIPLY:

cout << "v1\*v2=" << v1.scalmul(v2) << endl;

break;

case ADD:

v3.assign(v1.strnewcat(v2));

break;

case CLEAR:

v1.clear();

break;

case DELETE\_LAST:

v2.delLast();

break;

case AT:

cout << "enter index:" << endl;

cin >> index;

cout << "enter value:" << endl;

cin >> val;

v3.at(index) = val;

break;

case INSERT:

cout << "enter value:" << endl;

cin >> val;

v3.insert(val);

break;

default: cout << "ERROR";

}

v1.print();

v2.print();

v3.print();

cout << endl << "enter your choice 0-8:\n";

cin >> choice;

}

return 0;

}

Example 1: Example 2:

Ex

Test 1 - Constructors

v1 capacity: 4 v1 size: 0

v2 capacity: 16 v2 size: 0

v3 capacity: 2 v3 size: 0

Test 2 - Assign

v1 capacity: 4 v1 size: 4

v2 capacity: 16 v2 size: 4

v3 capacity: 4 v3 size: 4

Test 3 - Operations

enter your choice 0-8:

3

v1\*v2=30

capacity: 4 size: 4 values: 1 2 3 4

capacity: 16 size: 4 values: 1 2 3 4

capacity: 4 size: 4 values: 5 6 7 8

enter your choice 0-8:

0

Test 1 - Constructors

v1 capacity: 4 v1 size: 0

v2 capacity: 16 v2 size: 0

v3 capacity: 2 v3 size: 0

Test 2 - Assign

v1 capacity: 4 v1 size: 4

v2 capacity: 16 v2 size: 4

v3 capacity: 4 v3 size: 4

Test 3 - Operations

enter your choice 0-8:

4

capacity: 4 size: 4 values: 1 2 3 4

capacity: 16 size: 4 values: 1 2 3 4

capacity: 32 size: 8 values: 1 2 3 4 1 2 3 4

enter your choice 0-8:

0

**בהצלחה רבה!!**