Names: Tomas Kmet, Meet Bhatt
Course Name: Principles of Software Design
Lab Section: B02
Course Code: ENSF 480
Assignment Number: Lab-3
Submission Date and Time: 11/10/2023

Exercise A

Circle.h

```
//
// Created by Meet Bhatt on 2023-10-10.
//

#ifndef EXERCISEA_CIRCLE_H
#define EXERCISEA_CIRCLE_H

#include "Shape.h"

class Circle: public virtual Shape{
public:
   Circle(double x, double y, double radius, const char* name);
   //constructor that initializes its data members with given arguments
   Circle(const Circle& source);//copy ctor
   Circle& operator =(const Circle& S);//assignment operator
   virtual double area();//calculate area of shape//not virtual
   virtual double perimeter();//calculate perimeter//not virtual
   const double getRadius() const;//gets the radius of Circle
   void setRadius(double rad);
   void display();//displays the name, x and y coordinates of origin, Radius,
perimeter, and area, in a format
private:
   double radius;
};

#endif //EXERCISEA_CIRCLE_H
```

Circle.cpp

```
//
// Created by Meet Bhatt on 2023-10-10.
//

#include "Circle.h"
```

```cpp
#include "Shape.h"
#include "Point.h"
#include <string.h>
using namespace std;
#include <math.h>

#define PI 3.14159265358979323846;
Circle::Circle(double x, double y, double radius, const char*name): Shape(x, y, name),
radius(radius){}
Circle::Circle(const Circle& source): Shape(source.getOrigin().getx(),
source.getOrigin().gety(), source.getName()), radius(source.getRadius()){}
Circle& Circle::operator=(const Circle &S){
   if(this == &S)
       return *this;
   delete[] this->shapeName;
   this->shapeName = new char[strlen(S.getName()) + 1];
   strcpy(this->shapeName, S.getName());
   return *this;
}
double Circle::area()
{
   double ans = 3.14159265358979323846 * radius * radius;
   return ans;
}

double Circle::perimeter()
{
   double ans = 2 * 3.14159265358979323846 * radius;
   return ans;
}

const double Circle::getRadius() const {
   return this->radius;
}

void Circle::setRadius(double rad) {
   this->radius = rad;
}

void Circle::display() {
   cout << "Circle Name: " << this->getName() << endl;
   cout << "X-Coordinate: " << this->getOrigin().getx() << endl;
```

```
    cout << "Y-Coordinate: " << this->getOrigin().gety() << endl;
    cout << "Radius: " << this->getRadius() << endl;
    cout << "Area: " << this->area() << endl;
    cout << "Perimeter: " << this->perimeter() << endl;
}
```

CurveCut.h

```
//
// Created by Meet Bhatt on 2023-10-11.
//

#ifndef EXERCISEA_CURVECUT_H
#define EXERCISEA_CURVECUT_H

#include "Circle.h"
#include "Rectangle.h"

class CurveCut: public Rectangle, public Circle
{
public:
    CurveCut(double x, double y, double side_a, double side_b, double radius, const
char *name);
    //constructor that initializes its data members with given arguments
    CurveCut(const CurveCut& source);//copy constructor
    CurveCut& operator =(const CurveCut&S);//assignment operator

    double area();//calculate area
    double perimeter();//calculate perimeter

//    const double getSideA();
//    const double getSideB();
//    const double getRadius();

//    void setRadius(double rad);
//    void set_side_a(double side);
//    void set_side_b(double side);

    void display();


};
#endif //EXERCISEA_CURVECUT_H
```

CurveCut.cpp

```
//
// Created by Meet Bhatt on 2023-10-11.
//

#include "CurveCut.h"
#include <string.h>
#include <iostream>
CurveCut::CurveCut(double x, double y, double side_a, double side_b, double radius,
const char *name):Shape(x,y,name),

Rectangle(x,y,side_a,side_b,name),

Circle(x,y,radius,name) {
   if(radius > side_a || radius > side_b)
   {
       cout << "Error: The cut radius is too large" << endl;
       exit(1);
   }
}

CurveCut::CurveCut(const CurveCut &S):
Shape(S.getOrigin().getx(),S.getOrigin().gety(),S.getName()),Rectangle(S.getOrigin().g
etx(),S.getOrigin().gety(),S.getSideA(),S.getSideB(),S.getName()),

Circle(S.getOrigin().getx(),S.getOrigin().gety(),S.getRadius(),S.getName()){
   if(S.getRadius() > S.getSideA() || S.getRadius() > S.getSideB())
   {
       cout << "Error: The cut radius is too large" << endl;
       exit(1);
   }
}



CurveCut& CurveCut::operator=(const CurveCut &S) {
   if(S.getRadius() > S.getSideA() || S.getRadius() > S.getSideB())
   {
       cout << "Error: The cut radius is too large" << endl;
       exit(1);
   }
```

```
    if(this == &S)
        return *this;
    delete[] this->shapeName;
    this->shapeName = new char[strlen(S.getName()) + 1];
    strcpy(this->shapeName, S.getName());

    //now copy the radius, side_a, and side_b
    this->setRadius(S.getRadius());
    this->set_side_a(S.getSideA());
    this->set_side_b(S.getSideB());
    return *this;
}

//const double CurveCut::getRadius() {return this->getRadius();}
//const double CurveCut::getSideA() {return this->getSideA();}
//const double CurveCut::getSideB() {return this->getSideB();}

//void CurveCut::setRadius(double rad) {this->setRadius(rad);}
//void CurveCut::set_side_a(double side) {this->set_side_a(side);}
//void CurveCut::set_side_b(double side) {this->set_side_b(side);}

void CurveCut::display() {
    cout << "CurveCut Name: " << this->getName() << endl;
    cout << "X-Coordinate: " << this->getOrigin().getx() << endl;
    cout << "Y-Coordinate: " << this->getOrigin().gety() << endl;
    cout << "Side a: " << this->getSideA() << endl;
    cout << "Side b: " << this->getSideB() << endl;
    cout << "Radius of the cut: " << this->getRadius() << endl;

}
```

Shape.h

```
/*
* File Name: Shape.h
* Assignment: Lab 1 Exercise B
* Lab Section: B02
* Completed by: Tomas Kmet and Meet Bhett
* Submission Date: Oct 2, 2023
*/
#ifndef EXERCISEA_SHAPE_H
#define EXERCISEA_SHAPE_H
```

```
#include "Point.h"
#include <iostream>
using namespace std;

class Shape
{
public:
   Shape(double x, double y, const char* name); //creates a new shape origin
   const Point getOrigin() const; //returns a reference to an origin point.
   // Does not allow modifications to the x and y values
   const char* getName() const; //returns the name of the shape
   virtual void display(); //Prints on the screen the shape's name, x and y
coordinates of origin in a format
   double distance(Shape& other); //calculates distance of two shape origins
   static double distance(Shape& the_shape, Shape& other); //calculates distance of
two shape origins
   void move (double dx, double dy); //changes the position of the shape by x+dx and
y+dy
   virtual ~Shape(); //shape destructor
protected:
   char* shapeName;
   Point origin;
};

#endif //EXERCISEA_SHAPE_H
```

Shape.cpp

```
/*
* File Name: Shape.cpp
* Assignment: Lab 1 Exercise B
* Lab Section: B02
* Completed by: Tomas Kmet and Meet Bhett
* Submission Date: Oct 2, 2023
*/
#include "Shape.h"
#include "Point.h"
#include <string.h>
using namespace std;

Shape::Shape(double x, double y, const char* name):origin(double (x), double (y)){
```

```
    shapeName = new char[strlen(name) + 1];
    strcpy(shapeName, name);
}

const Point Shape::getOrigin() const {
    return origin;
}

const char* Shape::getName() const {
    return shapeName;
}

double Shape::distance(Shape &other) {
    return this->origin.distance(other.origin);
}

double Shape::distance(Shape& the_shape, Shape& other){
    Point::distance(the_shape.getOrigin(), other.getOrigin());
}

void Shape::move(double dx, double dy){
    origin.setx(origin.getx() + dx);
    origin.sety(origin.gety() + dy);
}

void Shape::display(){
    cout << "Shape Name: " << getName() << endl;
    this->origin.display();
}

Shape::~Shape(){
    delete []shapeName;
}
```

Square.h

```
/*
* File Name: Square.h
* Assignment: Lab 1 Exercise B
* Lab Section: B02
* Completed by: Tomas Kmet and Meet Bhett
```

```
* Submission Date: Oct 2, 2023
*/

#ifndef EXERCISEA_SQUARE_H
#define EXERCISEA_SQUARE_H

#include "Shape.h"

class Square: public virtual Shape{
public:
   Square(double x, double y, double side, const char* name);
   //constructor that initializes its data members with given arguments
   Square(const Square& source); //copy constructor
   Square& operator =(const Square& S); //assignment operator
   virtual double area(); //calculates area of shape
   virtual double perimeter(); //calculates perimeter
   const double getSideA() const; //gets the side_a
   void setSideA(double side);
   void display(); //displays the name, x and y coordinates of origin, side lengths,
perimeter, and area, in a format
private:
   double side_a;
};

#endif //EXERCISEA_SQUARE_H
```

Square.cpp

```
/*
* File Name: Square.cpp
* Assignment: Lab 1 Exercise B
* Lab Section: B02
* Completed by: Tomas Kmet and Meet Bhett
* Submission Date: Oct 2, 2023
*/
#include "Square.h"
#include "Shape.h"
#include "Point.h"
#include <string.h>
using namespace std;
```

```
Square::Square(double x, double y, double side, const char* name): Shape(x, y, name),
side_a(side){}

Square::Square(const Square& source): Shape(source.getOrigin().getx(),
source.getOrigin().gety(), source.getName()), side_a(source.getSideA()){}
Square& Square::operator =(const Square &S) {
   if(this == &S)
       return *this;
   delete[] this->shapeName;
   this->shapeName = new char[strlen(S.getName())+1];
   strcpy(this->shapeName, S.getName());
   return *this;
}
double Square::area(){
   return side_a*side_a;
}

double Square::perimeter() {
   return 4*side_a;
}

const double Square::getSideA() const {
   return side_a;
}

void Square::setSideA(double side) {
   side_a = side;
}

void Square::display(){
   cout << "Square Name: " << this->getName() << endl;
   cout << "X-Coordinate: " << this->getOrigin().getx() << endl;
   cout << "Y-Coordinate: " << this->getOrigin().gety() << endl;
   cout << "Side a: " << this->getSideA() << endl;
   cout << "Area: " << area() << endl;
   cout << "Perimeter: " << perimeter() << endl;
}
```

Rectangle.h

```
/*
* File Name: Rectangle.h
```

```
* Assignment: Lab 1 Exercise B
* Lab Section: B02
* Completed by: Tomas Kmet and Meet Bhett
* Submission Date: Oct 2, 2023
*/

#ifndef EXERCISEA_RECTANGLE_H
#define EXERCISEA_RECTANGLE_H

#include "Square.h"

class Rectangle: public Square {
public:
   Rectangle(double x, double y, double side_a, double side_b, const char *name);
   //constructor that initializes its data members with given arguments
   Rectangle(const Rectangle &source); //copy constructor
   Rectangle& operator =(const Rectangle& S); //assignment operator
   double area(); //calculates area of shape
   double perimeter(); //calcualtes perimeter
   const double getSideA() const; //gets the side_a
   void set_side_a(double side); //sets sideA
   const double getSideB() const; //gets the side_a
   void set_side_b(double side); //sets sideB
   void display(); //displays the name, x and y coordinates of origin, side lengths,
perimeter, and area, in a format
private:
   double side_a{};
   double side_b;
};


#endif //EXERCISEA_RECTANGLE_H
```

Rectangle.cpp

```
/*
* File Name: Rectangle.cpp
* Assignment: Lab 1 Exercise B
* Lab Section: B02
* Completed by: Tomas Kmet and Meet Bhett
* Submission Date: Oct 2, 2023
*/
```

```cpp
#include "Rectangle.h"
#include "Square.h"
#include "Point.h"
#include "Shape.h"
#include <string.h>
using namespace std;

Rectangle::Rectangle(double x, double y, double side_a, double side_b, const char
*name) : Shape(x, y, name), Square(x, y, side_a, name), side_a(side_a), side_b(side_b)
{}


Rectangle::Rectangle(const Rectangle &source) : Shape(source.getOrigin().getx(),
source.getOrigin().gety(), source.getName()), Square(source.getOrigin().getx(),

source.getOrigin().gety(),

source.getSideA(),

source.getName()), side_a(source.getSideA()), side_b(source.getSideB()){}
Rectangle& Rectangle::operator =(const Rectangle &S) {
   if(this == &S)
       return *this;
   delete[] this->shapeName;
   this->shapeName = new char[strlen(S.getName())+1];
   strcpy(this->shapeName, S.getName());
   this->side_a = S.getSideA();
   this->side_b = S.getSideB();
   this->origin.setx(S.getOrigin().getx());
   this->origin.sety(S.getOrigin().gety());
   return *this;
}

double Rectangle::area(){
   return side_a*side_b;
}

double Rectangle::perimeter() {
   return 2*side_a+2*side_b;
}

const double Rectangle::getSideA() const {
```

```
    return side_a;
}

const double Rectangle::getSideB() const {
    return side_b;
}

void Rectangle::set_side_a(double side) {
    side_a = side;
}

void Rectangle::set_side_b(double side) {
    side_b = side;
}

void Rectangle::display(){
    cout << "Rectangle Name: " << this->getName() << endl;
    cout << "X-Coordinate: " << this->getOrigin().getx() << endl;
    cout << "Y-Coordinate: " << this->getOrigin().gety() << endl;
    cout << "Side a: " << this->getSideA() << endl;
    cout << "Side b: " << this->getSideB() << endl;
    cout << "Area: " << area() << endl;
    cout << "Perimeter: " << perimeter() << endl;
}
```

Exercise B
iterator.cpp

```
/*
* File Name: iterator.cpp
* Assignment: Lab 3 Exercise B
* Lab Section: B02
* Completed by: Tomas Kmet and Meet Bhatt
* Submission Date: Oct 11, 2023
*/
#include <iostream>
#include <assert.h>
#include "mystring2.h"

using namespace std;

/*
template <class T>
      ostream & operator << (ostream &, vector <T>&);
```

```
*/
template <class T>
class Vector {
public:
//friend ostream & operator << <T> (ostream &, Vector <T>&);
 class VectIter{
   friend class Vector;
 private:
   Vector<T> *v; // points to a vector object of type T
   int index;    // represents the subscript number of the vector's
                 // array.
 public:
   VectIter(Vector& x);

   T operator++();
   //PROMISES: increments the iterator's indes and return the
   //          value of the element at the index position. If
   //          index exceeds the size of the array it will
   //          be set to zero. Which means it will be circulated
   //          back to the first element of the vector. Prefix

   T  operator++(int);
   // PRIMISES: returns the value of the element at the index
   //           position, then increments the index. If
   //           index exceeds the size of the array it will
   //           be set to zero. Which means it will be circulated
   //           back to the first element of the vector. Postfix

   T  operator--();
   // PROMISES: decrements the iterator index, and return the
   //           the value of the element at the index. If
   //           index is less than zero it will be set to the
   //           last element in the aray. Which means it will be
   //           circulated to the last element of the vector. Prefix

   T  operator--(int);
   // PRIMISES: returns the value of the element at the index
   //           position, then decrements the index. If
   //           index is less than zero it will be set to the
   //           last element in the aray. Which means it will be
   //           circulated to the last element of the vector. Postfix

   T operator *();
   // PRIMISES: returns the value of the element at the current
   //           index position.
 };

 Vector(int sz);
 ~Vector();
```

```
 T & operator[](int i);
 // PROMISES: returns existing value in the ith element of
 //           array or sets a new value to  the ith element in
 //           array.

   void ascending_sort();
 // PROMISES: sorts the vector values in ascending order.

private:
 T *array;               // points to the first element of an array of T
 int size;               // size of array
 void swap(T&, T&); // swaps the values of two elements in array
public:
};

template <class T>
void Vector<T>::ascending_sort()
{
   for(int i=0; i< size-1; i++)
      for(int j=i+1; j < size; j++)
         if(array[i] > array[j])
            swap(array[i], array[j]);
}

template <class T>
void Vector<T>::swap(T& a, T& b)
{
   T tmp = a;
   a = b;
   b = tmp;
}

template <class T>
T Vector<T>::VectIter::operator ++(){
   if (index + 1 >= v -> size){
       index = 0;
       return v -> array[0];
   }
   index ++;
   return v -> array[index];
}

template <class T>
T Vector<T>::VectIter::operator ++(int){
   if (index > v -> size){
       index = 0;
       return v -> array[v -> size];
   }
```

```
   index ++;
   return v -> array[index - 1];
}

template <class T>
T Vector<T>::VectIter::operator --(){
   if (index < 0){
       index = v -> size;
       return v -> array[index];
   }
   index --;
   return v -> array[index];
}

template <class T>
T Vector<T>::VectIter::operator--(int){
   T return_val = v->array[index];  // Store the current index
   index--;
   if (index < 0){
       index = v->size - 1;  // If index is already at the beginning, set it
to the last valid index
   }
   return return_val;  // Return the previous value before decrementing
}

template <class T>
T Vector<T>::VectIter::operator *()
{
 return v -> array[index];
}

template <class T>
Vector<T>::VectIter::VectIter(Vector<T>& x)
{
 v = &x;
 index = 0;
}

template <class T>
Vector<T>::Vector(int sz)
{
 size=sz;
 array = new T [sz];
 assert (array != NULL);
}

template <class T>
Vector<T>::~Vector()
{
```

```
 delete [] array;
 array = NULL;
}

template <class T>
T & Vector<T> ::operator [] (int i)
{
 return array[i];
}

template <>
class Vector <const char *>{
public:
   class VectIter{
       friend class Vector;
       friend class VectIter;
   public:
       VectIter(Vector<const char*>& x) : v(&x), index(0) {}
   private:
       Vector<const char*>* v;
       int index;    };

   Vector(int s): size(s){
       array = new const char *[size];
       assert(array != nullptr);
       strcpy(reinterpret_cast<char *>(array), reinterpret_cast<const char
*>(s));
   }
   ~Vector() {
       for (int i = 0; i < size; ++i) {
           delete[] array[i];
       }
       delete[] array;
   }
   const char*& operator[](int i) {
       assert(i >= 0 && i < size);
       return array[i];
   }

   Vector<const char*>ascending_sort()
   {
       for(int i=0; i< size-1; i++)
           for(int j=i+1; j < size; j++)
               if(array[i] > array[j])
                   swap(array[i], array[j]);
   }
   const char* operator++(){

   }
```

```
   const char* operator++(int);
   const char* operator--();
   const char* operator--(int);
private:
   int size;
   const char** array;
};

template <>
class Vector <Mystring *>{
public:
   friend class Vector;
   friend class VectIter;
   Vector(int s): size(s){
       array = new char* [size];
       assert(array != nullptr);
       strcpy(reinterpret_cast<char *>(array), reinterpret_cast<const char
*>(s));
   }
   ~Vector() {
       for (int i = 0; i < size; ++i) {
           delete[] array[i];
       }
       delete[] array;
   }
private:
   int size;
   char** array;
};

int main()
{

Vector<int> x(3);
x[0] = 999;
x[1] = -77;
x[2] = 88;

Vector<int>::VectIter iter(x);

cout << "\n\nThe first element of vector x contains: " << *iter;

// the code between the  #if 0 and #endif is ignored by
// compiler. If you change it to #if 1, it will be compiled

#if 1
   cout << "\nTesting an <int> Vector: " << endl;

   cout << "\n\nTesting sort";
```

```
   x.ascending_sort();

   for (int i=0; i<3 ; i++)
      cout << endl << iter++;

   cout << "\n\nTesting Prefix --:";
   for (int i=0; i<3 ; i++)
      cout << endl << --iter;

   cout << "\n\nTesting Prefix ++:";
   for (int i=0; i<3 ; i++)
      cout << endl << ++iter;

   cout << "\n\nTesting Postfix --";
   for (int i=0; i<3 ; i++)
      cout << endl << iter--;

   cout << endl;
#endif
/*
   cout << "Testing a <Mystring> Vector: " << endl;
   Vector<Mystring> y(3);
   y[0] = "Bar";
   y[1] = "Foo";
   y[2] = "All";;

   Vector<Mystring>::VectIter iters(y);

   cout << "\n\nTesting sort";
   y.ascending_sort();
   /*
   for (int i=0; i<3 ; i++)
      cout << endl << iters++;

   cout << "\n\nTesting Prefix --:";
   for (int i=0; i<3 ; i++)
      cout << endl << --iters;

   cout << "\n\nTesting Prefix ++:";
   for (int i=0; i<3 ; i++)
      cout << endl << ++iters;

   cout << "\n\nTesting Postfix --";
   for (int i=0; i<3 ; i++)
      cout << endl << iters--;
   */
#if 0
   cout << endl; cout << "Testing a <char *> Vector: " << endl;
   Vector<const char*> z(3);
```

```
   z[0] = "Orange";
   z[1] = "Pear";
   z[2] = "Apple";;

   Vector<const char*>::VectIter iterchar(z);

   cout << "\n\nTesting sort";
   z.ascending_sort();

   for (int i=0; i<3 ; i++)
      cout << endl << iterchar++;

#endif
   cout << "\nPrgram Terminated Successfully." << endl;

   return 0;
}
```

mystring2.cpp

```
/*
* File Name: mystring2.cpp
* Assignment: Lab 3 Exercise B
* Lab Section: B02
* Completed by: Tomas Kmet and Meet Bhatt
* Submission Date: Oct 11, 2023
*/
#include "mystring2.h"
#include <string.h>
#include <iostream>
using namespace std;

Mystring::Mystring()
{
 charsM = new char[1];
 charsM[0] = '\0';
 lengthM = 0;
```

```
}

Mystring::Mystring(const char *s)
 : lengthM(strlen(s))
{
 charsM = new char[lengthM + 1];
 strcpy(charsM, s);
}

Mystring::Mystring(int n)
 : lengthM(0), charsM(new char[n])
{
 charsM[0] = '\0';
}

Mystring::Mystring(const Mystring& source):
 lengthM(source.lengthM), charsM(new char[source.lengthM+1])
{
 strcpy (charsM, source.charsM);
}

Mystring::~Mystring()
{
 delete [] charsM;
}

int Mystring::length() const
{
 return lengthM;
}

char Mystring::get_char(int pos) const
{
 if(pos < 0 && pos >= length()){
   cerr << "\nERROR: get_char: the position is out of boundary." ;
 }

 return charsM[pos];
}

const char * Mystring::c_str() const
{
 return charsM;
}

void Mystring::set_char(int pos, char c)
{
 if(pos < 0 && pos >= length()){
   cerr << "\nset_char: the position is out of boundary."
```

```
    << " Nothing was changed.";
   return;
 }

 if (c != '\0'){
   cerr << "\nset_char: char c is empty."
    << " Nothing was changed.";
   return;
 }

 charsM[pos] = c;
}

Mystring& Mystring::operator =(const Mystring& S)
{
 if(this == &S)
   return *this;
 delete [] charsM;
 lengthM = (int) strlen(S.charsM);
 charsM = new char [lengthM+1];
 strcpy(charsM,S.charsM);
 return *this;
}

Mystring& Mystring::append(const Mystring& other)
{
 char *tmp = new char [lengthM + other.lengthM + 1];
 lengthM+=other.lengthM;
 strcpy(tmp, charsM);
 strcat(tmp, other.charsM);
 delete []charsM;
 charsM = tmp;

 return *this;
}

void Mystring::set_str(char* s)
{
   delete []charsM;
   lengthM = (int) strlen(s);
   charsM=new char[lengthM+1];

   strcpy(charsM, s);
}
```

mystring2.h

```
/*
* File Name: mystring2.h
* Assignment: Lab 3 Exercise B
```

```
* Lab Section: B02
* Completed by: Tomas Kmet and Meet Bhatt
* Submission Date: Oct 11, 2023
*/

#ifndef MYSTRING_H
#define MYSTRING_H

class Mystring {
public:
 Mystring();
 // PROMISES: Empty string object is created.

 Mystring(int n);
 // PROMISES: Creates an empty string with a total capacity of n.
 //           In other words, dynamically allocates n elements for
 //           charsM,sets the lengthM to zero, and fills the first
 //           element of charsM with '\0'.

 Mystring(const char *s);
 // REQUIRES: s points to first char of a built-in string.
 // REQUIRES: Mystring object is created by copying chars from s.

 ~Mystring(); // destructor

 Mystring(const Mystring& source); // copy constructor

 Mystring& operator =(const Mystring& rhs); // assignment operator
 // REQUIRES: rhs is reference to a Mystring as a source
 // PROMISES: to make this-object (object that this is pointing to, as  a copy
 //           of rhs.

 int length() const;
 // PROMISES: Return value is number of chars in charsM.

 char get_char(int pos) const;
 // REQUIRES: pos >= 0 && pos < length()
 // PROMISES:
 // Return value is char at position pos.
 // (The first char in the charsM is at position 0.)

 const char * c_str() const;
 // PROMISES:
 //   Return value points to first char in built-in string
 //   containing the chars of the string object.

 void set_char(int pos, char c);
 // REQUIRES: pos >= 0 && pos < length(), c != '\0'
 // PROMISES: Character at position pos is set equal to c.
```

```
 Mystring& append(const Mystring& other);

 // PROMISES: extends the size of charsM to allow concatenate other.charsM to
 //           to the end of charsM. For example if charsM points to "ABC", and
 //          other.charsM points to XYZ, extends charsM to "ABCXYZ".
 //

 void set_str(char* s);
 // REQUIRES: s is a valid C++ string of characters (a built-in string)
 // PROMISES:copys s into charsM, if the length of s is less than or equal
lengthM.
 //          Othrewise, extends the size of the charsM to s.lengthM+1, and
copies
 //          s into the charsM.

private:

 int lengthM; // the string length - number of characters excluding \0
 char* charsM; // a pointer to the beginning of an array of characters,
allocated dynamically.
 void memory_check(char* s);
 // PROMISES: if s points to NULL terminates the program.
};
#endif
```

Output

```
The first element of vector x contains: 999
Testing an <int> Vector:


Testing sort
-77
88
999

Testing Prefix --:
999
88
-77

Testing Prefix ++:
88
999
-77

Testing Postfix --
-77
999
88

Prgram Terminated Successfully.
```