Secure Data Structures and Algorithms with C++ 8th edition Carrano Solutions Manual

SKU: 9780138122782-SOLUTIONS

Solutions to a Selection of Exercises

(Version 8.0)

SECURE DATA STRUCTURES and ALGORITHMS with C++

WALLS AND MIRRORS

Frank M. Carrano | Timothy M. Henry

Frank M. Carrano
University of Rhode Island

Timothy M. Henry
Rhode Island College

PART I: The Foundation

Chapter 1 Data Abstraction: The Walls

1. The price of an item in the park gift shop is given in dollars and cents. Customers can pay for it in cash by giving the clerk d dollars and c cents. Write specifications for a function that computes the change, if any, that the customer should receive. Include a statement of purpose, the preconditions and postconditions, and a description of the arguments.

```
const CENTS_PER_DOLLAR = 100;

// Computes the change a customer should receive by paying d dollars and
// c cents for an item costing dollarCost dollars and centsCost cents.
// Preconditions:  dollarCost, centsCost, d, and c are all nonnegative
//   integers, and centsCost and c are both less than CENTS_PER_DOLLAR.
// Postconditions:  d and c contain the computed remainder values in
//   dollars and cents respectively.  If input value d < dollarCost, the
//   proper negative values for the amount owed in d dollars and/or c
//   cents is returned.
void computeChange(int dollarCost, int centsCost, int& d, int& c);
```

---

2. A date consists of three integers that represent a month, day, and year. For example, July 4, 1776, is month 7, day 4, and year 1776. Consider the class Date of such dates.

a. Write specifications for a member function within Date that advances any given date by one day. Include a statement of purpose, the preconditions and postconditions, a description of the arguments, and a description of any return value.

```
const MONTHS_PER_YEAR = 12;
int daysPerMonth[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

// Increments the input Date values (month, day, year) by one day.
// Preconditions: 1 <= month <= MONTHS_PER_YEAR,
//                1 <= day <= number of days in the month.
// Postcondition:  The valid numeric values for the succeeding month, day,
//                 and year are returned.
void incrementDate(int& month, int& day, int& year);

// Tests whether the input year is a leap year.
// Precondition:  year > 0.
// Postconditions:  Returns true if year is a leap year; false otherwise.
bool isLeapYear(int year);
```

b. Write a C++ implementation of this member function. Design and specify any other member function that you need. Include comments that are helpful to someone who maintains your implementation in the future.

```
#include <iostream>

bool isLeapYear(int year)
{   // Example:  2000 is a leap year but 1900 was not
    return ((year % 400 == 0) ||
            ((year % 4 == 0) && (year % 100 != 0)));
} // end isLeapYear


void incrementDate(int& month, int& day, int& year)
{
   const int MONTHS_PER_YEAR = 12;
   int daysPerMonth[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
   const int FEB = 2; // Index to daysPerMonth for February

   bool goodDate = true;
   if ((month <= 0) || (month > MONTHS_PER_YEAR))
   {
      std::cout << "Bad input value for month." << std::endl;
      goodDate = false;
   } // end if
   // month is valid

   if (year <= 0)
   {
      std::cout << "Bad input value for year." << std::endl;
      goodDate = false;
   } // end if
   // year is valid

   // Get number of days in February
   if (month == FEB)
   {
      if (isLeapYear(year))
         daysPerMonth[2] = 29;
      else
         daysPerMonth[2] = 28;
   } // end if

   if ((day <= 0) || (day > daysPerMonth[month]))
   {
      std::cout << "Bad input value for day." << std::endl;
      goodDate = false;
   } // end if

   if (goodDate)
   {  // Increment the date
      day++; // Increment the day

      if (day > daysPerMonth[month])
      {
         day = 1;
         month++;

         if (month > MONTHS_PER_YEAR)
         {  // Increment to next year
            month = 1;
            year++;
         } // end if
      } // end if
   }
   else
   {
```

```
      std::cout << "Cannot increment date due to bad input." << std::endl;
   } // end if
} // end incrementDate

void testIncrementDate(int month, int day, int year)
{
   std::cout << "Today is " << month << "/" << day << "/" << year << std::endl;
   incrementDate(month, day, year);
   std::cout << "Tomorrow is " << month << "/" << day << "/" << year << std::endl << std::endl;
} // end testIncrementDate

int main()
{
   testIncrementDate(4, 8, 2024);   // Ordinary day mid month
   testIncrementDate(5, 31, 2024);  // End of month
   testIncrementDate(2, 28, 2024);  // End of February in a leap year
   testIncrementDate(2, 28, 2025);  // End of February when not a leap year
   testIncrementDate(12, 31, 2024); // End of year
} // end main

/*
 Today is 4/8/2024
 Tomorrow is 4/9/2024

 Today is 5/31/2024
 Tomorrow is 6/1/2024

 Today is 2/28/2024
 Tomorrow is 2/29/2024

 Today is 2/28/2025
 Tomorrow is 3/1/2025

 Today is 12/31/2024
 Tomorrow is 1/1/2025
 */
```

---

3. Write a pseudocode function in terms of the ADT AppointmentBook, described in Section 1.4.1, for each of the following tasks. Do you need to add operations to the ADT to perform these tasks?

a. Change the purpose of the appointment at a given date and time.

```
changeAppointmentPurpose(in apptDate:Date, in apptTime:Time,
                         in  purpose:string):boolean
{
   if (isAppointment(apptDate, apptTime))
      cancelAppointment(apptDate, apptTime)

   return makeAppointment(apptDate, apptTime, purpose)
}
```

---

b. Display all the appointments for a given date.

```
displayAllAppointments(in apptDate:Date)
{
   time = startOfDay
```

```
        while (time < endOfDay)
        {
           if (isAppointment(apptDate, time))
              displayAppointment(apptDate, time)

           time = time + halfHour
        }
    }
```

This implementation requires the definition of a new operation,
`displayAppointment(date, time)`
as well as definitions for the constants
`startOfDay, endOfDay, halfHour.`

---

4. Consider the ADT Polynomial—in a single variable *x*—whose operations include the following:

```
degree()        // Returns the degree of a polynomial.
coefficient(power) // Returns the coefficient of the x to the power term.
changeCoefficient(newCoefficient, power) // Replaces the coefficient of the x
                    // to the power term with
                    // newCoefficient.
```

For this problem, consider only polynomials whose exponents are nonnegative integers. For example,

p = 4x5 + 7x3 − x2 + 9

The following examples demonstrate the ADT operations on this polynomial.

p.degree() is 5 (the highest power of a term with a nonzero coefficient)
p.coefficient(3) is 7 (the coefficient of the x3 term)
p.coefficient(4) is 0 (the coefficient of a missing x4 term is implicitly 0)
p.changeCoefficient(–3, 7) changes the polynomial p to –3x^7 + 4x^5 + 7x^3 − x^2 + 9

Using these ADT operations, write statements to perform the following tasks:

a. Display the coefficient of the term that has the highest power.

```
display(p.coefficient(p.degree()))
```

b. Increase the coefficient of the x^3 term by 8.

```
p.changeCoefficient(p.coefficient(3) + 8, 3)
```

c. Compute the sum of two polynomials.

```
for (power = 0; power < p.degree() || power < q.degree(); power++)
{
    r.changeCoefficient(p.coefficient(power) + q.coefficient(power), power)
}
```

Chapter 2 Bags

1. Imagine that you have just left the park gift shop with a bag of souvenirs. You are concerned that the fragile items will not survive the trip home, so when you reach your car, you place those items into their own bag. If Bag is a class of bags, write C++ statements that remove all the items from storeBag and place them into one of two new bags, as follows: Place all occurrences of fragile items into fragileBag, and all other items into giftBag. When you are done, storeBag should be empty. Assume that strings represent souvenir items. The first character in each string is F if the item is fragile or X if not.

```
#include <iostream>
#include "Bag.h"

void organizeSouvenirs(Bag<std::string>& storeBag, Bag<std::string>& fragileBag,
Bag<std::string>& giftBag)
{
   std::vector<std::string> items = storeBag.toVector();
   int itemCount = storeBag.getCurrentSize();
   for (int counter = 0; counter < itemCount; counter++)
   {
      std::string nextItem = items[counter];
      if (nextItem[0] == 'F')
         fragileBag.add(nextItem);
      else
         giftBag.add(nextItem);
   } // end for

   storeBag.clear();
} // end organizeSouvenirs

void displayBag(const BagInterface<std::string>& aBag)
{
   std::cout << "The bag contains " << aBag.getCurrentSize()
             << " items:" << std::endl;
   std::vector<std::string> bagItems = aBag.toVector();

   int numberOfEntries = (int)bagItems.size();
   for (int i = 0; i < numberOfEntries; i++)
   {
      std::cout << bagItems[i] << " ";
   }  // end for
   std::cout << std::endl << std::endl;
}  // end displayBag

int main()
{
   std::string gift1 = "FGlassVase";
   std::string gift2 = "FSunGlasses";
   std::string gift3 = "XTeddyBear";
   std::string gift4 = "XFeltPenant";
   std::string gift5 = "FCeramicPhotoFrame";
   std::string gift6 = "XBallCap";
   std::string gift7 = "FMirror";
   std::string gift8 = "XT-Shirt";

   Bag<std::string> storeBag;
   Bag<std::string> fragileBag;
   Bag<std::string> giftBag;

   storeBag.add(gift1);
```

```
   storeBag.add(gift2);
   storeBag.add(gift3);
   storeBag.add(gift4);
   storeBag.add(gift5);
   storeBag.add(gift6);
   storeBag.add(gift7);
   storeBag.add(gift8);

   displayBag(storeBag);
   organizeSouvenirs(storeBag, fragileBag, giftBag);
   displayBag(fragileBag);
   displayBag(giftBag);
} // end main
/*
 The bag contains 6 items:
 FGlassVase FSunGlasses XTeddyBear XFeltPenant FCeramicPhotoFrame XBallCap

 The bag contains 3 items:
 FGlassVase FSunGlasses FCeramicPhotoFrame

 The bag contains 3 items:
 XTeddyBear XFeltPenant XBallCap
 */
```

---

2. Suppose that a bag—that is, an instance of the class Bag—contains strings that represent various souvenirs. Write a C++ stand-alone function that removes and counts all occurrences of a given souvenir from such a bag. Your function should return this number. Use comments in `javadoc` style to fully specify your function. Accommodate the possibility that the given bag is either empty or does not contain any occurrences of the given souvenir.

```
#include <iostream>
#include "Bag.h"

/** Counts the number of times a given souvenir occurs in a given bag of souvenir.
 @param storeBag  A bag of souvenirs.
 @param aSouvenir  A string naming the souvenir to be counted.
 @return  The number of times aSouvenir occurs in storeBag. */
int countSouvenir(Bag<std::string> storeBag, std::string aSouvenir)
{
   int counter = 0;
   bool done = false;
   while (!done)
   {
      if (storeBag.remove(aSouvenir))
      {
         counter++; // Count removed souvenir
      }
      else
      {
         done = true;  // Bag no longer contains given souvenir
      } // end if
   } // end while

   return counter;
} // end countSouvenir



void displayBag(const BagInterface<std::string>& aBag)
{
   std::cout << "The bag contains " << aBag.getCurrentSize()
```

```
            << " items:" << std::endl;
   std::vector<std::string> bagItems = aBag.toVector();

   int numberOfEntries = (int)bagItems.size();
   for (int i = 0; i < numberOfEntries; i++)
   {
      std::cout << bagItems[i] << " ";
   }  // end for
   std::cout << std::endl << std::endl;
}  // end displayBag

int main()
{
   std::string gift1 = "T-Shirt";
   std::string gift2 = "GlassVase";
   std::string gift3 = "SunGlasses";
   std::string gift4 = "TeddyBear";
   std::string gift5 = "T-Shirt";
   std::string gift6 = "FeltPenant";
   std::string gift7 = "CeramicPhotoFrame";
   std::string gift8 = "BallCap";
   std::string gift9 = "Mirror";
   std::string gift10 = "T-Shirt";

   Bag<std::string> storeBag;

   storeBag.add(gift1);
   storeBag.add(gift2);
   storeBag.add(gift3);
   storeBag.add(gift4);
   storeBag.add(gift5);
   storeBag.add(gift6);
   storeBag.add(gift7);
   storeBag.add(gift8);
   storeBag.add(gift9);
   storeBag.add(gift10);

   displayBag(storeBag);

   int number = countSouvenir(storeBag, gift8);
   std::cout << "The bag contains " << number << " Ball Caps." << std::endl;

   number = countSouvenir(storeBag, gift1);
   std::cout << "The bag contains " << number << " T-Shirts." << std::endl;

   number = countSouvenir(storeBag, "Roller Skates");
   std::cout << "The bag contains " << number << " Roller Skates." << std::endl;
} // end main
/*
 The bag contains 10 items:
 T-Shirt GlassVase SunGlasses TeddyBear T-Shirt FeltPenant CeramicPhotoFrame BallCap Mirror
T-Shirt

 The bag contains 1 Ball Caps.
 The bag contains 3 T-Shirts.
 The bag contains 0 Roller Skates.
 */
```

3. The union of two bags is a new bag containing the combined contents of the original two bags. Design and specify an operation `union` for the ADT Bag that returns as a new bag the union of the bag receiving the call to the operation and the bag that is the operation's one argument. Include sufficient comments to fully specify the operation.

Note that the union of two bags might contain duplicate items. For example, if object x occurs five times in one bag and twice in another, the union of these bags contains x seven times. Specifically, suppose that bag1 and bag2 are bags; bag1 contains the strings a, b, and c; and bag2 contains the strings b, b, d, and e. The expression bag1.union(bag2) returns a bag containing the strings a, b, b, b, c, d, and e. Note that union does not affect the contents of bag1 and bag2.

```
/** Creates a new bag that combines the contents of this bag and another bag.
    @param anotherBag  The second bag.
    @return  The union of the two bags. */
+union(anotherBag : BagInterface): BagInterface
```

---

4. The intersection of two bags is a new bag containing the entries that occur in both of the original two bags. Design and specify an operation `intersection` for the ADT Bag that returns as a new bag the intersection of the bag receiving the call to the operation and the bag that is the operation's one argument. Include sufficient comments to fully specify the operation.

Note that the intersection of two bags might contain duplicate items. For example, if object x occurs five times in one bag and twice in another, the intersection of these bags contains x two times. Specifically, suppose that bag1 and bag2 are bags; bag1 contains the strings a, b, and c; and bag2 contains the strings b, b, d, and e.The expression bag1.intersection(bag2) returns a bag containing only the string b. Note that `intersection` does not affect the contents of bag1 and bag2.

```
/** Creates a new bag that contains those objects occurring
    in both this bag and another bag.
    @param anotherBag  The second bag.
    @return  The intersection of the two bags. */
+intersection(anotherBag: BagInterface: BagInterface
```

---

5. The difference of two bags is a new bag containing the entries that would be left in one bag after removing those that also occur in the second. Design and specify an operation `difference` for the ADT Bag that returns as a new bag the difference of the bag receiving the call to the operation and the bag that is the operation's one argument. Include sufficient comments to fully specify the operation.

Note that the difference of two bags might contain duplicate items. For example, if object x occurs five times in one bag and twice in another, the difference of these bags contains x three times. Specifically, suppose that bag1 and bag2 are bags; bag1 contains the strings a, b, and c; and bag2contains the strings b, b, d, and e. The expression `bag1.difference(bag2)` returns a bag containing only the strings a and c. Note that `difference` does not affect the contents of bag1 and bag2.

```
/** Creates a new bag that contains the objects remaining after
    removing from this bag the objects in another bag.
    @param anotherBag  The second bag.
    @return  The difference of the two bags. */
+difference(anotherBag: BagInterface: BagInterface
```

Chapter 3 Array-Based Implementations

1. Consider a bag of integers. Write a client function that computes the sum of the integers in the bag `aBag`.

```
int sumOf(Bag<int> aBag)
{
   std::vector<int> bagContents = aBag.toVector();
   int bagSize = aBag.getCurrentSize();
   int sum = 0;

   for (int i = 0; i < bagSize; i++)
   {
      sum = sum + bagContents[i];
   } // end while

   return sum;
} // end sumOf
```

---

2. Write a client function `replace` that replaces a given item in a given bag with another given item. The function should return a Boolean value to indicate whether the replacement was successful.

```
** Replaces a given entry in a given bag with another given entry.
 @param givenEntry  The entry in the bag to be replaced.
 @param replacement  The entry to replace givenEntry.
 @return  True if the replacement is successful, or false otherwise; */
bool replace(Bag<std::string>& aBag, std::string givenEntry,
             std::string replacement)
{
   bool success = aBag.remove(givenEntry);
   if (success)
      success = aBag.add(replacement);

   return success;
} // end replace
```

---

3. The previous exercise describes the function `replace`. This operation exists outside of the ADT Bag; that is, it is not an ADT Bag operation. Instead, its implementation is written in terms of the ADT Bag's operations.

a. What is an advantage and a disadvantage of the way that replace is implemented?

Advantages: The function `replace` is independent of the implementaiton of the ADT Bag. The function is relatively easy to define.
Disadvantage: Since `replace` cannot directly access the bag's entries, it likely is less efficient than if it were a member of a class of bags.

b. What is an advantage and a disadvantage of adding the operation replace to the ADT Bag?

Advantage: The function `replace` can efficiently and directly replace an entry in the data structure containing the bag's entries.
Disadvantage: As a member function, the definition of `replace` likely is harder and more error-prone to write.

4. Write a client function that merges two bags into a new third bag. Do not destroy the original two bags.

```
#include <iostream>
#include "Bag.h"

/** Merges two given bags into a new third bag.
 @param bag1  A bag.
 @param bag2  Another bag.
 @return  A new bag of merged entries. */
Bag<std::string> merge(Bag<std::string> bag1, Bag<std::string> bag2)
{
   // Add bag2's entries to the copy of bag1:
   std::vector<std::string> bag2Entries = bag2.toVector();
   for (int i = 0; i < bag2.getCurrentSize(); i++)
      bag1.add(bag2Entries[i]);

   return bag1;
} // end merge

void displayBag(const BagInterface<std::string>& aBag)
{
   int numberOfEntries = aBag.getCurrentSize();
   std::cout << "The bag contains " << numberOfEntries
             << " items:" << std::endl;
   std::vector<std::string> bagItems = aBag.toVector();

   for (int i = 0; i < numberOfEntries; i++)
   {
      std::cout << bagItems[i] << " ";
   }  // end for
   std::cout << std::endl << std::endl;
}  // end displayBag

int main()
{
   std::string gift1 = "T-Shirt";
   std::string gift2 = "GlassVase";
   std::string gift3 = "SunGlasses";
   std::string gift4 = "TeddyBear";
   std::string gift5 = "T-Shirt";
   std::string gift6 = "BallCap";
   std::string gift7 = "Bracelet";

   Bag<std::string> myBag;
   myBag.add(gift1);
   myBag.add(gift2);
   myBag.add(gift3);
   myBag.add(gift4);

   Bag<std::string> yourBag;
   yourBag.add(gift5);
   yourBag.add(gift6);
   yourBag.add(gift7);

   std::cout << "Here are the entries in my bag:" <<std:: endl;
   displayBag(myBag);

   std::cout << "Here are the entries in your bag:" <<std:: endl;
   displayBag(yourBag);
```

```
   Bag<std::string> mergedBag = merge(yourBag, myBag);

   std::cout << "Here are the entries in the merged bag:" <<std:: endl;
   displayBag(mergedBag);

   std::cout << "Here are the entries in my bag:" <<std:: endl;
   displayBag(myBag);

   std::cout << "Here are the entries in your bag:" <<std:: endl;
   displayBag(yourBag);

} // end main
/*
Here are the entries in my bag:
The bag contains 4 items:
T-Shirt GlassVase SunGlasses TeddyBear

Here are the entries in your bag:
The bag contains 3 items:
T-Shirt BallCap Bracelet

Here are the entries in the merged bag:
The bag contains 7 items:
T-Shirt BallCap Bracelet T-Shirt GlassVase SunGlasses TeddyBear

Here are the entries in my bag:
The bag contains 4 items:
T-Shirt GlassVase SunGlasses TeddyBear

Here are the entries in your bag:
The bag contains 3 items:
T-Shirt BallCap Bracelet
*/
```

---

5. Specify and define a member function for `ArrayBag` that removes a random entry from the bag.

Add the following to `BagInterface`:

```
/** Removes one entry at random from this bag, if possible.
    @post  If successful, an entry has been removed from the bag
    and the count of items in the bag has decreased by 1.
    @return  The removed entry, if removal was successful;
             otherwise return a default value of itemType. */
virtual ItemType remove() = 0;
```

Add the following to the header file `ArrayBag.h`:

```
ItemType remove();
```

Add the following to the implementation file `ArrayBag.cpp`:

```
template<class ItemType>
ItemType ArrayBag<ItemType>::remove()
{
   ItemType entryToRemove;

   // Get a random integer between 0 and itemCount - 1:
```

```
      int randomIndex = rand() % itemCount;
      bool canRemoveItem = !isEmpty() && (randomIndex > -1);
      if (canRemoveItem)
      {
         entryToRemove = items[randomIndex];
         itemCount--;
         items[randomIndex] = items[itemCount];
      } // end if

      return entryToRemove;
   }  // end remove
```

---

6. Add a constructor to the class ArrayBag that creates a bag from a given array of entries.

```
template<class ItemType>
Bag<ItemType>::Bag(ItemType entries[], int count): itemCount(0), maxItems(DEFAULT_CAPACITY)
{
   for (int i = 0; i < count; i++)
      add(entries[i]);
}  // end constructor
```

---

7. Design and implement an ADT that represents a rectangle. Include typical operations, such as setting and retrieving the dimensions of the rectangle and finding the area and the perimeter of the rectangle.

ADT Rectangle operations:

```
+Rectangle()
// Creates a rectangle and initializes its length and width to default values.

+Rectangle (in length:double, in width:double)
// Creates a rectangle and initilializes its length and width to
// the positive values received as parameters.

+setLength (in length:double)
// Sets or modifies the length of this rectangle.
// Checks to make sure that the new length is greater than 0.



+setWidth (in width:double)
// Sets or modifies the width of this rectangle.
// Checks to make sure that the new width is greater than 0.

+getLength():double {query}
// Returns this rectangle's length.

+getWidth():double {query}
// Returns this rectangle's width.

+getArea():double {query}
// Returns this rectangle's area.

+getPerimeter():double {query}
// Returns this rectangle's perimeter.
```

```
/** An interface for the ADT Rectangle.
    @file RectangleInterface.h */

#ifndef RECTANGLE_INTERFACE_
#define RECTANGLE_INTERFACE_

class RectangleInterface
{
public:
   /** Sets or modifies the length of this rectangle.
       @param newLength  The length of the rectangle as a positive numeric value.
       @pre newLength > 0. */
   virtual void setLength(double newLength) = 0;

   /** Sets or modifies the width of this rectangle.
       @param newWidth  The width of the rectangle as a positive numeric value.
       @pre newWidth > 0. */
   virtual void setWidth(double newWidth) = 0;

   /** Returns the length of this rectangle. */
   virtual double getLength() const = 0;

    /** Returns the width of this rectangle. */
   virtual double getWidth() const = 0;

   /** Returns the area of this rectangle. */
   virtual double getArea() const = 0;

   /** Returns the perimeter of this rectangle. */
   virtual double getPerimeter() const = 0;

   /** Destroys this rectangle and frees its assigned memory. */
   virtual ~RectangleInterface() { }
}; // end RectangleInterface
#endif
```

```
/** Header file for the ADT Rectangle.
 @file Rectangle.h */

#ifndef RECTANGLE_
#define RECTANGLE_

#include "Rectangle.h"

class Rectangle : public RectangleInterface
{
 private:
    double length;
    double width;
public:
   Rectangle();
   Rectangle(double initialLength, double initialWidth);
   void setLength(double newLength);
   void setWidth(double newWidth);
   double getLength() const;
   double getWidth() const;
   double getArea() const;
   double getPerimeter() const;
};  // end Rectangle

#include "Rectangle.cpp"
```

```
#endif
```

```
/** Implementation file for the class Rectangle.
    @file Rectangle.cpp */
#include "Rectangle.h"

Rectangle::Rectangle()
{
   setLength(1.0);
   setWidth(1.0);
} // end default constructor

Rectangle::Rectangle(double initialLength, double initialWidth)
{
   setLength(initialLength);
   setWidth(initialWidth);
} // end constructor

void Rectangle::setLength (double newLength)
{
   if (newLength > 0.0)
      length = newLength;
   else
      std::cout << "The new length of the rectangle is negative." << std::endl;
} // end setLength

void Rectangle::setWidth(double newWidth)
{
   if (newWidth > 0.0)
      width = newWidth;

   else
      std::cout << "The new width of the rectangle is negative." << std::endl;
} // end setWidth

double Rectangle::getLength() const
{
   return length;
} // end getLength

double Rectangle::getWidth() const
{
   return width;
} // end getWidth

double Rectangle::getArea() const
{
   double area = getLength() * getWidth();
   return area;
} // end getArea

double Rectangle::getPerimeter() const
{
   double perimeter = 2 * (getLength() + getWidth());
   return perimeter;
} // end getPerimeter
```

```
#include <iostream>
#include "Rectangle.h"

void displayRectangle(const Rectangle aRectangle)
```

```
{
   std::cout << "The rectangle's width is " << aRectangle.getWidth() << std::endl;
   std::cout << "It's length is " << aRectangle.getLength() << std::endl;
   std::cout << "It has an area of " << aRectangle.getArea()
             << " and a perimeter of " << aRectangle.getPerimeter() << std::endl;
   std::cout << std::endl;
}  // end displayBag

int main()
{
   Rectangle defaultRectangle;
   Rectangle myRectangle(3.0, 4.0);

   std::cout << "Here is a default rectangle:" << std:: endl;
   displayRectangle(defaultRectangle);

   std::cout << "Here is my rectangle:" << std:: endl;
   displayRectangle(myRectangle);

   std::cout << "Let's change the rectangle's dimensions to 4 by 6:" << std:: endl;
   myRectangle.setWidth(4.0);
   myRectangle.setLength(6.0);
   displayRectangle(myRectangle);
} // end main
/*
 Here is a default rectangle:
 The rectangle's width is 1
 It's length is 1
 It has an area of 1 and a perimeter of 4

 Here is my rectangle:
 The rectangle's width is 4
 It's length is 3
 It has an area of 12 and a perimeter of 14

 Let's change the rectangle's dimensions to 4 by 6:
 The rectangle's width is 4
 It's length is 6
 It has an area of 24 and a perimeter of 20
 */
```

---

8. Design and implement an ADT that represents a triangle. The data for the ADT should include the three sides of the triangle but could also include the triangle's three angles. This data should be in the private section of the class that implements the ADT.

Include at least two initialization operations: one that provides default values for the ADT's data, and another that sets this data to client-supplied values. These operations are the class's constructors. The ADT also should include operations that look at the values of the ADT's data; change the values of the ADT's data; compute the triangle's area; and test whether the triangle is either a right triangle, an equilateral triangle, or an isosceles triangle.

ADT Triangle operations:

```
+Triangle()
// Creates a triangle and initializes its sides to default values.

+ Triangle(in initialSide1:double, in initialSide2:double, in initialSide3:double)
// Creates a triangle and initilializes its sides to
```

```
// the positive values received as parameters.

+setSides(in newSide1:double, in newSide2:double, in newSide3:double): boolean
// Sets or modifies the sides of this triangle.
// Checks to make sure that the new sides form a triangle.

+getSides():vector<double> {query}
// Returns this triangle's sides.

+getArea():double {query}
// Returns this triangle's area.

+getPerimeter():double {query}
// Returns this triangle's perimeter.

+isRightTriangle():boolean {query}
// Returns true if this triangle is a right triangle.
+isEquilateralTriangle():boolean {query}
// Returns true if this triangle is an equilateral triangle.

+isIsoscelesTriangle():boolean {query}
// Returns true if this triangle is an isosoceles triangle.
```

```
/** An interfaca for the ADT Triangle.
    @file TriangleInterface.h */

#ifndef TRIANGLE_INTERFACE_
#define TRIANGLE_INTERFACE_

#include <vector>

class TriangleInterface
{
public:
   /** Sets or modifies the sides of this triangle.
       @param side1  The length of one side of this triangle.
       @param side2  The length of one side of this triangle.
       @param side3  The length of one side of this triangle.
       @pre   side1, side2, and side3 are each greater than zero.
       @post  If the given sides form a triangle,
              sideC is the largest side and sideA is the smallest.
       @return true if the given sides form a triangle. */
   virtual bool setSides(double side1, double side2, double side3) = 0;

   /** Returns a vector containing the three sides of this triangle,
       arranged in descending order, with the largest one in the
       first element. */
   virtual std::vector<double> getSides() const = 0;

   /** Returns the area of this triangle. */
   virtual double getArea() const = 0;

   /** Returns the perimeter of this triangle. */
   virtual double getPerimeter() const = 0;

   /** Tests whether this triangle is a right triangle.
       @return true if the triangle is right triangle;
               otherwise returns false. */
   virtual bool isRightTriangle() const = 0;

   /** Tests whether this triangle is an equilateral triangle.
       @return true if the triangle is equilateral triangle;
```

```
                otherwise returns false. */
      virtual bool isEquilateralTriangle() const = 0;

      /** Tests whether this triangle is an isosceles triangle.
          @return true if the triangle is isosceles triangle;
                otherwise returns false. */
      virtual bool isIsoscelesTriangle() const = 0;

      /** Destroys this triangle and frees its assigned memory. */
      virtual ~TriangleInterface() { }
   }; // end TriangleInterface
   #endif
```

```
   /** Header file for the ADT Triangle.
    @file Triangle.h */

   #ifndef TRIANGLE_
   #define TRIANGLE_

   #include "TriangleInterface.h"
   #include <vector>

   class Triangle : public TriangleInterface
   {
    private:
       double sideA, sideB, sideC;

      bool initializeSides(double x, double y, double z);
      void orderSides(double& x, double& y);

   public:
      Triangle();
      Triangle(double initialSide1, double initialSide2, double initialSide3);
      bool setSides(double newSide1, double newSide2, double newSide3);
      std::vector<double> getSides() const;
      double getArea() const;
      double getPerimeter() const;

      bool isRightTriangle() const;
      bool isEquilateralTriangle() const;
      bool isIsoscelesTriangle() const;
   }; // end Triangle

   #include "Triangle.cpp"
   #endif
```