HW #4 (Integer Set)

- Design and implement, using C++, or Java if you prefer, a simple ADT called **IntSet** for sets of integers. Each object of class **IntSet** can hold zero or more integers in the range 0 through 100.
- A set should be represented internally as an array of ones and zeros. Array element *a[i]* is 1 if integer *i* is in the set. Array element *a[j]* is 0 if integer *j* is not in the set.
- The default constructor should initialize a set to an empty set, i.e., a set whose array representation contains all zeros.
- The ADT should support the following set of operations:

HW #4 (2)

- default constructor **IntSet()** – creates an empty set.
- constructor with one integer parameter – creates a set containing just the given integer. Example:

  IntSet(0) creates the set {0}.
- function **isEmpty()** – returns true if the set is empty, else false; does not change the set.
- function **size()** – returns the number of elements in the set, an integer between 0 and 100; does not change the set.

HW #4 (3)

- functiin **setPrint()** – prints set to *cout* with entries surrounded by curly braces and separated by commas (or a blank, if you prefer); returns nothing; does not change the set.

  Example: if the set *A* has element 1, 4, and 7,

  A.setPrint() prints {1, 4, 7}.
- function **setUnion()** – creates a third set which is the set-theoretic union of two existing sets.
- function **setIntersetion()** – creates a third set which is the set-theoretic intersection of two existing sets.

HW #4 (4)

- function **relativeComplement()** – creates a third set which is the set of elements that are in set A and not in set B.
- function **symmetricDifference()** – creates a third set whose elements belong to set A or to set B but not both.
- function **isEqualTo()** – returns true of the two sets are equal, false otherwise; does not change either set.

HW #4 (5)

- The class must be organized as two files, a header file, *intset.h*, containing the class definition and an implementation file, *intset.cpp*, containing the code for the functions of the class.
- Order of values in a set is unimportant but the set should not contain duplicates.
- The following is a suggestion of how your *intset.h* could possibly look like.

```
#ifndef INTSET_H
#define INTSET_H

class IntSet {
public:
  IntSet() { emptySet(); } // default constructor
  IntSet( int ); // alternate (overloaded) constructor
  IntSet setUnion( const IntSet& );
  IntSet setIntersection( const IntSet& );
  bool isEmpty( void );
  int size( void );
  IntSet relativeComplement( const IntSet& );
  IntSet symmetricDifference( const IntSet& );
  void setPrint( void ) const;
  bool isEqualTo( const IntSet& ) const;
  // Auxiliary functions
  ……..
private:
  int set[ 101 ];  // range of 0 – 100
  // Private member functions, if necessary
};
#endif
```

HW #4 (6)

- **Testing your solution**: Run the following test driver program to test you class. Make sure that all results are correct before submitting your solution.
- Feel free to write your own test program to ensure your solution is OK.

```
// Test program for the IntSet class

#include <iostream.h>
#include "intset.h“

int main()
{
  IntSet Empty; // the empty set

  // for singleton sets {0} .. {3}
  IntSet S0(0), S1(1), S2(2), S3(3);

  IntSet A, B, C, D, E, F, G; // to hold computed sets

  // Show and test empty set
  cout << "\nShow and test the empty set...\n";
  cout << "Empty = ";
  Empty.setPrint();
  cout << " has "<< Empty.size() << " elements." << endl;
```

```
if ( Empty.isEmpty() )
  cout << "The set is empty\n" << endl;
else
  cout << "The set is *not* empty\n" << endl;

// Show and test {1}
cout << "S1 = ";
S1.setPrint();
cout << " has "<< S1.size() << " elements." << endl;
if ( S1.isEmpty() )
  cout << "Set S1 is empty\n" << endl;
else
  cout << "Set S1 is *not* empty\n" << endl;

// Compute some unions
A = S0.setUnion(Empty);
S0.setPrint();
cout << " union ";
Empty.setPrint();
cout << " = ";
A.setPrint();
cout << endl << endl;
```

```
A = S0.setUnion(S1);
B = S3.setUnion(S2);
A.setPrint();
cout << " union ";
B.setPrint();
cout << " = ";
C = A.setUnion(B);
C.setPrint();
cout << endl << endl;

A = A.setUnion(S3);
B = B.setUnion(S0);
A.setPrint();
cout << " union ";
B.setPrint();
cout << " = ";
D = A.setUnion(B);
D.setPrint();
cout << endl << endl;
```

```
// Compute intersection, relative complement, and symmetric difference

E = A.setIntersection(S3);
cout << "Intersection of ";
A.setPrint();
cout << " and ";
S3.setPrint();
cout << " is: ";
E.setPrint();
cout << endl << endl;

G = D.relativeComplement(S0);
cout << "Relative complement of ";
D.setPrint();
cout << "and ";
S0.setPrint();
cout << " is: ";
G.setPrint();
cout << endl << endl;
```

```
F = B.symmetricDifference(A);
cout << "Symmetric difference of ";
B.setPrint();
cout << " and ";
A.setPrint();
cout << " is: ";
F.setPrint();
cout << endl << endl;

// Test if two sets are equal
cout << "Set A: ";
A.setPrint();
cout << endl;
cout << "Set B: ";
B.setPrint();
cout << endl;
if ( A.isEqualTo(B) )
  cout << "Set A is equal to set B\n";
else
  cout << "Set A is not equal to set B\n";

cout << endl;
return 0; }
```

Show and test the empty set...
Empty = {--- } has 0 elements.
The set is empty

S1 = { 1 } has 1 elements.
Set S1 is *not* empty

{ 0 } union {--- } = { 0 }
{ 0 1 } union { 2 3 } = { 0 1 2 3 }
{ 0 1 3 } union { 0 2 3 } = { 0 1 2 3 }
Intersection of { 0 1 3 } and { 3 } is: { 3 }
Relative complement of { 0 1 2 3 } and { 0 } is: { 1 2 3 }
Symmetric difference of { 0 2 3 } and { 0 1 3 } is: { 1 2 }

Set A: { 0 1 3 }
Set B: { 0 2 3 }
Set A is not equal to set B

```
import lib.IntSet;

public class test {
	public static void main(String [] args) {
	IntSet Empty = new IntSet();
	IntSet S0 = new IntSet(0);
	IntSet S1 = new IntSet(1);
	IntSet S2 = new IntSet(2);
	IntSet S3 = new IntSet(3);

	IntSet A, B, C, D, E, F, G;

	System.out.println("\nShow and test the empty set...");
	System.out.print("Empty = ");
	Empty.setPrint();
	System.out.println(" has " + Empty.size() + " elements.");
	if (Empty.isEmpty())
	  System.out.println("The set is empty\n");
	else
	  System.out.println("The set is *not* empty\n");

	System.out.print("S1 = ");
	S1.setPrint();
	System.out.println(" has " + S1.size() + " elements.");
	if (S1.isEmpty())
	  System.out.println("Set S1 is empty\n");
	else
	  System.out.println("Set S1 is *not* empty\n");
```

```
A = S0.setUnion(Empty);
S0.setPrint();
System.out.print(" union ");
Empty.setPrint();
System.out.print(" = ");
A.setPrint();
System.out.print("\n\n");

A = S0.setUnion(S1);
B = S3.setUnion(S2);
A.setPrint();
System.out.print(" union ");
B.setPrint();
System.out.print(" = ");
C = A.setUnion(B);
C.setPrint();
System.out.print("\n\n");

A = A.setUnion(S3);
B = B.setUnion(S0);
A.setPrint();
System.out.print(" union ");
B.setPrint();
System.out.print(" = ");
D = A.setUnion(B);
D.setPrint();
System.out.print("\n\n");
```

```
E = A.setIntersection(S3);
System.out.print("Intersection of ");
A.setPrint();
System.out.print(" and ");
S3.setPrint();
System.out.print(" is ");
E.setPrint();
System.out.print("\n\n");

G = D.relativeComplement(S0);
System.out.print("Relative complement of ");
D.setPrint();
System.out.print(" and ");
S0.setPrint();
System.out.print(" is ");
G.setPrint();
System.out.print("\n\n");

F = B.symmetricDifference(A);
System.out.print("Symmetric difference of");
B.setPrint();
System.out.print(" and ");
A.setPrint();
System.out.print(" is ");
F.setPrint();
System.out.print("\n\n");
```

```
        System.out.print("Set A:");
        A.setPrint();
        System.out.println();
        System.out.print("Set B:");
        B.setPrint();
        System.out.println();
        if (A.isEqualTo(B))
          System.out.println("Set A is equal to set B\n");
        else
          System.out.println("Set A is not equal to set B\n");
        System.out.println();
    }
}
```