Assignment Purpose

This assignment aims to create a C# class representing sets of integers ranging from 0 to 50.
You should work in your groups (2 members) if you wish.

Mandatory Instructions

Begin by creating a solution folder. The folder's name should be one or two last names followed by Lab1 (e.g., Smith-Lab1 or Smith-Jones-Lab1) if both group members work on the solution.

You will create a C# source file IntegerSet.cs, that will be compiled into IntegerSet.dll. Begin by creating a makefile that you will use for this assignment. The application created by the makefile should be called Lab1.exe. Also, implement the "clean" target so that you can type "make clean" for the current directory. Use the command-line compiler csc for this assignment. **Do not use Visual Studio GUI or any other GUI.**

Define the IntegerSet class in the file IntegerSet.cs. Use the same namespace (SetNS) that is used for the main program. Remember to make your class public. Because the only integers allowed are those in the range 0 to 50 inclusive, you don't need the overhead of linked lists to implement the set. Use a bool array of length 51 to represent the set. If the value of position i is true, that means that i is in the set. (Example: If arr[5] is true, that means that 5 is in the set.)

These are the public member functions of IntegerSet that you will implement:

- ***Constructor*** with no arguments: Create an empty set
- ***Overloaded constructor*** with integer array argument: The elements to be put into the set are found in the array values. Suggestion: Use the foreach statement to go through each element of the array and add it to the set. If any value is invalid (less than zero or greater than 50), print a message and keep going. Don't terminate the program because of an invalid integer.
- ***EmptySet***: This member function accepts no arguments and removes all elements from the set.
- ***InsertFromArray***: Accepts an integer array as an argument and adds each integer element to the set. If any integer value is invalid, print a message. Don't terminate the program because of this error condition.
- ***InsertElement***: Adds one integer element to the set. If the integer is invalid, print a message. Don't terminate the program because of this error condition.
- ***DeleteElement***: Removes one integer element from the set. If the integer argument is invalid, print a message. Don't terminate the program because of this error condition.
- ***InputSet***: Allow elements to be added to the set. Read integers from the console, one at a time, and add them to the set. Stop the input when the enter key is pressed. For invalid integers, print an error message and keep going.
- ***Union***: Calculate the union of the set with another set. An IntegerSet is an argument to the function, and the function returns an IntegerSet.
- ***Intersection***: Calculate the intersection of the set with another set.
- ***Equals***: Override Equals to determine if the set is equal to another set (i.e., contains the same elements). This function requires the override keyword and returns a bool value.

- ***GetHashCode***: This function must be overridden whenever Equals is overridden. You can return some constant integer value.
- ***ToString***: Override the ToString method to return a string representing the set, using the format { 12 15 30 }. If the set is empty, return string literal "{ <empty> }".

You might find it convenient to define one or more private worker functions for use in your required function implementations. Make sure to document your code and to use meaningful variable names.

When you have fully debugged your code, upload your compressed solution folder to Canvas (both group members should upload the same solution). Your folder should contain only three files (App.cs, IntegerSet.cs, and the makefile) for this assignment.

Example of code you will place in Main() to produce requested output for set A:

```
Console.WriteLine("// create A by passing an array of integers to
overloaded constructor");

int[] myary = new int[5];
for (int i=1; i<5; i++)
     myary[i] = i;
myary[0] = 99; // invalid value
IntegerSet A = new IntegerSet(myary);
Console.WriteLine("A is: " + A);
```

Example of code you will place in Main() to produce requested output for set B:

```
// create new set
Console.WriteLine("// create B by adding one element at a time");
IntegerSet A = new IntegerSet();
B.InsertElement(31);
B.InsertElement(42);
B.InsertElement(11);
B.InsertElement(21);
B.InsertElement(29);
Console.WriteLine($"B is: {B}");
```

Demonstrate the following program output by calling appropriate functions in your App.cs client code. **DO NOT** hardcode these values. Produce output in the same order and showing the same comments.

```
c:\cs3160\w3\A1>App
// create A by passing an array of integers to overloaded constructor
Error: 99 is an invalid value
A is: { 1 2 3 4 }
// create B by adding one element at a time
B is: { 11 21 29 31 42 }
Intersection of A and B is: { <empty> }
Union of A and B is: { 1 2 3 4 11 21 29 31 42 }
// Is A equal to B?
Set A is not equal to set B
Copy A to B
// Is A equal to B?
Set A is equal to set B
// Insert 35 into A
Set A is now: { 1 2 3 4 35 }
// Delete 35 from A
Set A is now: { 1 2 3 4 }
// Enter elements into set C;
Enter set C:
Enter an element (Enter to end):1
Enter an element (Enter to end):10
Enter an element (Enter to end):15
Enter an element (Enter to end):20
Enter an element (Enter to end):25
Enter an element (Enter to end):
Set C is: { 1 10 15 20 25 }
Emptying set C...
Set C is: { <empty> }

c:\cs3160\w3\A1>
```

Code Documentation

Each source file should contain a documentation header with a description, author name, and class information (CS3160, Spring or Fall/Spring 20XX). Each function should have a documentation header with a minimally function name, description, parameters, return value. Use proper program style at all times, which means indentation, alignment, and whitespace. Utilize self-documenting code which means use mnemonic, i.e., meaningful variable/function/namespace/class names, etc.

What to turn in?

Submit a compressed (.zip) solution folder via Canvas.
Ensure that all files necessary to compile/link/execute your projects are provided.
Here are some files that may be required:

1. All required C# source files (.cs)
2. makefile (if the solution requires it)
3. The entire Visual Studio 2019 solution folder (if the solution requires it)