Overview

This homework is due at the start of your lab section on March 27th. It asks you to implement a single class, and is worth 75 points toward your cumulative homework grade.

You are to implement a SetOfInt class. This class should have the functionality of a mathematical set, including insert, remove, a membership test, a subset test, union, intersection, and set difference. I assume that you recall a fair amount about sets, but will explain more below. Recall in particular that in a set there are no repeated elements.

Grading will be slightly different from previous assignments, with points assigned for the proper writing, compiling and testing/demonstration of each function. This will allow you to earn significant partial credit even if some of your functions don’t work. You will need to submit three files: SetOfInt.h, SetOfInt.cpp, and main.cpp. The latter is your own code to effectively demonstrate that your class functions work properly. This code does not need to have any input. Instead it should hard-code certain contents of sets, run your functions, and output results. You should output specific information about what’s being tested and make your output readable. We will also test your class on our own main program.

Functions You Must Implement

Here are the functions you must implement, with their prototypes.

- **Default constructor**: Accepts no arguments and creates an empty set.
  
  ```cpp
  SetOfInt::SetOfInt();
  ```

- **Constructor from a single integer**: 
  
  ```cpp
  SetOfInt::SetOfInt( int e );
  ```

  Creates a set containing the single integer whose value is stored in e.

- **Copy constructor**: 

  ```cpp
  //
  ```
Copy a set to construct a new set.

**Assignment operator:**

```cpp
SetOfInt& SetOfInt :: operator= ( const SetOfInt& rhs );
```

Copies the `rhs` set over the current set and returns a reference to the current set.

**IsElement:**

```cpp
bool SetOfInt :: IsElement( int x ) const;
```

Returns true if the integer value stored in `x` is in the set.

**Insert:**

```cpp
bool SetOfInt :: Insert( int x );
```

Insert the value stored in `x` into the set. If the value is already there, do NOT add it again and return false. Return true if the value was not in the set.

**Remove:**

```cpp
bool SetOfInt :: Remove( int x );
```

Remove the value stored in `x` from the set. Return true if it was there and your code removed it. Return false otherwise.

**Output stream:**

```cpp
ostream& operator<< ( ostream& ostr, const SetOfInt& s );
```

Output the contents of the set to the stream and return a reference to the stream. Here’s how the output should look for a set containing the first 5 prime numbers:

```
{ 2, 3, 5, 7, 11 }
```
All the contents should be on a single line of output regardless of the set size. The set contents MUST also be output in increasing order. Here's how the output should look for an empty set

{ }

Note that you do NOT have to write an input stream operator.

• **IsSubset:**

    ```cpp
    bool IsSubset( const SetOfInt& s, const SetOfInt& t );
    ```

    Return true if and only if each element of \( s \) is also an element of \( t \). It is written here as a friend or non-member function, but you may make it a member function if you wish.

• **Equivalent:** This is implemented as `operator==`. It is written here as a friend or non-member function, but you may make it a member function if you wish.

    ```cpp
    bool operator==( const SetOfInt& s, const SetOfInt& t );
    ```

    Return true if sets \( s \) and \( t \) have exactly the same elements.

• **Union:** This is implemented as `operator+`. It is written here as a friend or non-member function, but you may make it a member function if you wish.

    ```cpp
    SetOfInt operator+( const SetOfInt& s, const SetOfInt& t );
    ```

    Construct a new set containing all elements that are in either \( s \) or \( t \). For example,

    \[ \{-3, 4, 6, 8, 10\} + \{-5, -3, 2, 5, 8, 9\} \]

    should give \( \{-5, -3, 2, 4, 5, 6, 8, 9, 10\}\).

• **Intersection:** This is implemented as `operator*`. It is written here as a friend or non-member function, but you may make it a member function if you wish.

    ```cpp
    SetOfInt operator*( const SetOfInt& s, const SetOfInt& t );
    ```
Construct a new set containing all elements that are in BOTH $s$ and $t$. For example,

$$\{-3, 4, 6, 8, 10\} \ast \{-5, -3, 2, 5, 8, 9\}$$

should return $\{-3, 8\}$.

- **Difference:** This is implemented as `operator-`. It is written here as a friend or non-member function, but you may make it a member function if you wish.

  ```cpp
  SetOfInt operator-( const SetOfInt& s, const SetOfInt& t );
  ```

  Construct a new set containing each element of $s$ (the first set) that is NOT in $t$ (the second set). For example,

  $$\{-3, 4, 6, 8, 10\} - \{-5, -3, 2, 5, 8, 9\}$$

  should return $\{4, 6, 10\}$.

**Member Variable**

The sole member variable MUST be either a vector of ints or a list of ints. Soon we will be discussing a better way to do this, but for now you must use a vector or a list.

**Suggestions**

- Implement and test one function (or a few related functions) at a time.
- Think about how to use STL algorithms to simplify the problems.
- Once you have one member function working (like `IsElement`), think about how to use it in the other member functions. For example, `IsSubset` can be made very easy by using `IsElement`. 