

4 Definitions

- A *set* is an unordered collection of elements with no repetitions. Examples are the set of real numbers, the set of integer numbers or the set consisting of numbers 1, 2, 30.
- For this assignment we will only be considering representing finite sets of integers. Examples: $\{0, 34, 78, 1000\}$, $\{4, 5, 890, 65535\}$, $\{0, 1, 2 \dots, 65534, 65535\}$, $\{\}$ are all valid sets.
- The *union* of two sets, say A and B , is written as $A \cup B$ and is the set which contains all elements in either A or B or both. Example: If $A = \{3, 8, 14, 15\}$ and $B = \{2, 8, 9, 15, 100\}$, then $A \cup B = \{2, 3, 8, 9, 14, 15, 100\}$ (notice that there are no repeated elements in a set).
- The *intersection* of two sets A and B is written as $A \cap B$ and is the set which contains the elements that are common to A and B . Examples: If $A = \{3, 8, 14, 15\}$ and $B = \{2, 8, 9, 15, 100\}$, then $A \cap B = \{8, 15\}$. If $A = \{17, 20, 38\}$ and $B = \{200\}$, then $A \cap B = \{\}$, which is termed the *empty set*.
- The *difference* of two sets A and B is written as $A \setminus B$ and is the set containing those elements which are in A and are not in B . Example: If $A = \{3, 8, 14, 15\}$ and $B = \{2, 8, 9, 15, 100\}$, then $A \setminus B = \{3, 14\}$ and $B \setminus A = \{2, 9, 100\}$.

5 Specifications

You are given the following C++ class `SLLNode` in your starter code:

```
class SLLNode{
    public:
        int value;
        SLLNode* next;

        SLLNode(int i, SLLNode* n) { value = i; next = n; }
};
```

You are also given the placeholder of the Class `SLLSet` which has only the following instance fields:

- 1) an integer to store the **size of the set**, i.e., the number of its elements;
- 2) a pointer to the beginning of the linked list (a pointer variable of type `SLLNode`).

All instance fields are **private**. You are also given the methods' prototypes in the class. Please note that you are required to complete all the methods in this lab (and correct their returns accordingly).

Class `SLLSet` contains at least the following constructors:

- `SLLSet()` - constructs an empty `SLLSet` ("empty" means with zero elements).

- `SLLSet(int sortedArray[], int arraySize)` - constructs an `SLLSet` object that contains the integers in the input array. Note that the array is **sorted in increasing order** and it does not contain repetitions. This constructor has to be efficient in terms of running time and memory usage.

Class `SLLSet` contains at least the following public methods:

- `int getSize()` - returns the size of **this** set.
- `SLLSet copy()` - returns a **deep copy** of the **current** `SLLSet`. The meaning of **deep** is that the two objects cannot share any piece of memory. Thus the **copy** represents a set with the same elements as **the current** set, but the two linked lists cannot have node objects in common.
- `bool isIn(int v):` - returns **true** if integer `v` is an element of **the current** `SLLSet`. It returns **false** otherwise.
- `void add(int v):` - adds `v` to **the current** `SLLSet` if `v` was not already an element of **the current** `SLLSet`. It does nothing otherwise.
- `void remove(int v):` - removes `v` from **the current** `SLLSet` if `v` was an element of the `SLLSet`. It does nothing otherwise.
- `SLLSet setUnion(SLLSet s):` - returns a new `SLLSet` which represents the union of **the current (this)** `SLLSet` and the input `SLLSet s`. This method has to be efficient in terms of running time and memory usage, in other words the amount of operations may be at most a constant value times m , where m is the sum of the sizes of the two sets. Moreover, the amount of additional memory used, apart from the memory for the input and output sets, may not be larger than a constant value. A "constant value" means here a value which does not grow as the sizes of the lists change. Partial marks will be awarded for inefficient implementations.
- `SLLSet intersection(SLLSet s):` - returns a new `SLLSet` which represents the intersection of **this** `SLLSet` and the input `SLLSet s`. This method has to be efficient in terms of running time and memory usage, in other words the amount of operations may be at most a constant value times m , where m is the sum of the sizes of the two sets. Moreover, the amount of additional memory used, apart from the memory for the input and output sets, may not be larger than a constant value. A "constant value" means here a value which does not grow as the sizes of the lists change. Partial marks will be awarded for inefficient implementations.
- `SLLSet difference(SLLSet s):` - returns a new `SLLSet` which represents the difference between **this** `SLLSet` and the input `SLLSet s`, i.e., **this** \ `s`. No efficiency requirements are imposed here.
- `static SLLSet setUnion(SLLSet sArray[], int size)` returns a new object representing the union of the sets in the array.
- `string toString()` - returns a string representing the set, with the elements listed in increasing order and separated by commas and a space. An example is as follows:
1, 2, 3, 4, 5