**Name : Omkar Nikhal**

**Class : SE-3**

**Roll no : 21342**

**ASSIGNMENT NO.** 9

**TITLE** Set implementation with its operations

**PROBLEM STATEMENT** To create ADT that implements the SET concept.

* Add (new Element) -Place a value into the set
* Remove (element) Remove the value
* Contains (element) Return true if element is in collection
* Size () Return number of values in collection
* Iterator () Return an iterator used to loop over collection
* Intersection of two sets
* Union of two sets
* Difference between two sets
* Subset

**OBJECTIVE** Understand how to implement SET ADT using linked list/ arrays.

**OUTCOME** At the end of this document students will able to implement SET Data Structure

**S/W PACKAGES AND** 1. (64-bit)64-BIT Fedora 17 or latest 64-BIT Update of

**HARDWARE**  Equivalent Open source OS

**APPARATUS USED 2.** Programming Tools (64-Bit) Latest Open source update of

Eclipse Programming frame work, TC++, GTK

**CONCEPTS RELATED THEORY:**

A set is an abstract data type that can store unique values, without any particular order. This is widely used data structure. The value of the element cannot be modified once it is added to the set, though it is possible to remove and add the modified value of that element. Unlike most other collection types, rather than retrieving a specific element from a set, one typically tests a value for membership in a set. There are various types of sets like static sets and dynamic sets. Static/immutable sets that do not change after they are constructed. Static sets allow only query operations on their elements — such as checking whether a given value is in the set, or enumerating the values in some arbitrary order. Other variants, called dynamic or mutable sets, allow also the insertion and deletion of elements from the set. Sets can be implemented using various data structures, which provide different time and space trade-offs for various operations. It can be implemented using arrays or linked list or hash table or vector or list etc. In this assignment we have to use liked list/arrays to implement the SET data structure without using STL concepts like vector or list. We have to use template concept so that sets can be of any type*.*

Following Basic Operations has to implement in assignment:

1. Insert (SetItem): No duplicate elements and can be inserted in any order. For duplicate element it should not provide any error.
2. Remove (SetItem): Remove the SetItem from SET
3. Contains (Setelement): Return true if Setelement is in SET
4. Size () Return number of values in collection
5. Iterator () Return an iterator used to loop over collection
6. Intersection (set1, set2): return new SET object which represent intersection of two sets,
7. Union (set1, set2): return new SET object which represent union of two sets
8. Difference (set1, set2): return new SET object which return difference between two sets
9. Subset (set1, set2): return true if set2 in subset of set1

**Class Diagram:**

**NODE<T>**

**SET<T>**

-data: T

-next: Node\*<T>

-head: node\*<T>

-length: int

+add(T): void

+remove(T): Boolean

+contains(T): Boolean

+size ():int

+union (SET): SET

+intersection (SET): SET

+difference (SET): SET

+subset (SET): Boolean

**ITERATOR<T>**

-it: Node\*<T>

+set (Node\*<T>): void

+begin (): void

+end (): Boolean

+next (): Node\*

+operator++: void

Hints for implementation: We can store the elements in increasing (sorted) order in list so that insertion order will not maintain.

Set, Node and Iterator can be template classes so that we can create sets of any type.

After calling set() of Iterator class which sets its it data member to head of list. Iterate through set by calling next() and test end of set until end() will return true.

We can also use operator overloading concepts for union, intersection and difference operations.

**algorithm** INTERSECTION (SET b)

{

SET C;

Node \*first := head;

Node \*second := b.head;

while ((i < length) && (j < b.length))

{

if (first.data < second.data)

{

first=first.next; i++

}

else if (first.data > second.data)

{

second=second.next ; j++;

}

else

{

C.add(first.data);

first=first.next;

second=second.next;

i++; j++;

}

}// end of while

return C

}

**algorithm** Union (SET b)

{

SET C;

i=j=0;

Node \*first := head;

Node \*second := b.head;

while ((i < length) && (j < b.length))

{

if (first.data < second.data)

{

C.add(first.data) ; i++;

first=first.next ;

}

else if (first.data > second.data)

{

C.add(second.data);

j++ ;

second=second.next;

}

else

{

C.add(first.data)

first=first.next

second=second.next ;

i++;j++

}

}// end of while

while (i < length)

{

C.add(first.data) ;

i++;

} // copy remaining elements of first set

while (j < b.length)

{

C.add(second.data)

j++;

} // copy remaining elements of second set

return C

}

Test-Cases

|  |  |  |  |
| --- | --- | --- | --- |
| Description | Input | Output | Result |
| Insert into set A | 2 5 6 1 9 | - | Pass |
| Display set A | - | 1 2 5 6 9 |  |
| Insert into set B | 4 5 9 2 7 | - | Pass |
| Display set B | - | 2 4 5 7 9 | Pass |
| Union | - | 1 2 4 5 6 7 9 | Pass |
| Intersection | - | 2 5 9 | Pass |
| A - B | - | 1 6 | Pass |

**CONCLUSION:** Able to implement set data structure using linked list