Assignment - 9

Title - Set Operations

Problem Statement -
To create ADT that implements SET concepts
(i) Add new element
(ii) Removes element
(iii) Returns true if element is present.
(iv) Return size of set.
(v) Interaction
(vi) Union
(vii) Difference
(viii) Subset

Objective - To implement SET ADT and learn set operations like intersection, union, difference, subset.

Outcomes - We will have a program ready to perform SET operations for various applications.

H/W and S/W Requirements -
Fedora 20
Dell Optiplex 3020MT
Keyboard
Monitor
Eclipse

Theory -

Sets -

Sets are a type of abstract data type that allows you to store a list of non-repeated values. Unlike an array, sets are unordered and unindexed.

The two properties of sets are that the data is unordered and it is not duplicated.

Set theory is useful if you need to collect data and do not care about their multiplicity or their order.

Operations -

1. Union (A, B)
   returns A ∪ B

2. Intersection (A, B)
   returns A ∩ B

3. Difference (A, B)
   returns A - B

4. Subset (A, B)
   Checks whether A is a subset of B and returns 0 or 1 accordingly.

Class Structure -

```
class Node {
    int data;
    Node * next;
public:  Node() {
             data = 0; next = NULL; }
         Node(int a) {
         data = a; next = NULL;
         } friend class Set;
};
```

```
class set {
  Node * start;
public:   set()
          { start = NULL; }
void ~~delete()~~;   void deleten(int x);
void insert(int x);
void display();
bool find(int x);
set intersection(set a);
set unionset(set a);
set differenceset(set a);
void subset();
};
```

Pseudo codes -

```
void insert(int x) {
  if(!find(x)) {
    if(start == NULL)
    { start = new Node(x);
      return;
    }
    Node * t = start;
    while(t->next != NULL)
      t = t->next;
    t->next = new Node(x);
  }
  else { cout << "Element already present!" << endl;
  }
}
```

2. void display( ) {
cout << " Set elements - " << endl;
Node * t = start;
if (t == NULL) {
cout << " EMPTY " << endl;
return;
}
while (t != NULL) {
cout << t -> data << " ";
t = t -> next;
}
cout << endl;
}

3. void deleten (int x) {
Node * t = start;
if (t -> data == x) {
start = t -> next;
delete t;
return;
} while (t -> next != NULL) {
if ~~while~~ (t -> next -> data == x) {
Node * temp = t -> next;
t -> next = temp -> next;
delete temp;
cout << x << " is deleted! " << endl;
return;
}
t = t -> next;
}
}

4.
```
find (int x) {
  Node * t = start;
  while ( t != NULL)
  { if ( t -> data == x)
      return true;
    t = t -> next;
  }
  return false;
}
```

5.
```
void size ( ) {
  Node * t = start;
  int count = 0;
  while (t) {
    count ++;
    t = t -> next;
  }
  cout << " size is - " << count << endl;
}
```

6.
```
Set intersection (Set a) {
  Set b;
  node * t = start;
  while ( t != NULL) {
    if ( a. find ( t -> data ))
    {
      b: insert ( t -> data );
    }
    t = t -> next;
  }
  return b;
}
```

7.
```
Set unionset (Set a) {
    Node * t = start;
    Set c;
    while (t != NULL) {
        c.insert (t -> data);
        t = t -> next;
    }
    t = a.start;
    while (t != NULL) {
        if (!c.find (t -> data))
            c.insert (t -> data);
        t = t -> next;
    }
    return c;
}
```

8.
```
Set difference (Set a) {
    Set d;
    Node * t = start;
    while (t) {
        if (!a.find (t -> data))
            c.insert (t -> data);
        t = t -> next;
    }
    return d;
}
```

9.
```
void subset (Set sub) {
    Node * t = sub.start;
    while (t != NULL) {
        if (find (t -> data))
            t = t -> next;
```

```
else {
    cout << " Not a subset ! " << endl;
    return ;  }
} cout << " Is a subset ! " << endl;
}
```

Test Cases -

	Case	Expected O/P	Actual O/P
1.	Insert - 1	A = [1|X]	Success
2.	Insert - 2	A = [1|] → [2|X]	Success
3.	Insert - 3	[1|] → [2|] → [3|X]	Success
4.	Delete - 2	[1|] → [3|X]	Success
5.	Insert - 1	"Element already present!"	Success
6.	Union - AUB B - {1, 4, 5}	C = {1, 3, 4, 5}	Success
7.	Intersection - B - {1, 4, 5}	B = {1}	Success
8.	Difference	D = {3}	Success
9.	Subset B = {1, 3, 5, 7}	It is not a subset!	Success

Conclusion -

We successfully implemented SET ADT.