# Assignment 9

**Title:** Set Operations

**Problem Statement:** To create ADT that implements SET concepts

- Add new element
- Removal element
- Returns true if element is present
- Returns size of set
- Intersection
- Union
- Difference
- Subset

**Objective:** To implement set ADT and learn set operations, like intersection, union, difference, subset.

**Outcome:** We will have a ready set ADT for applications.

**Requirements:** Dell Optiplex 3020 MT, keyboard, monitor, Fedora 20, Eclipse.

**Theory:**

Sets:- Abstract data type that can store unique values without any particular order. It is the complete implementation of finite set.

Operations:

Union (S,T) - returns S ⋃ T

Intersection (S,T) - returns S ∩ T

Difference (S, T) - returns S – T

Subset (S, T) - tells whether T is subset of S

**Class Definition:**

```
class node

{

   int data;

   node* next;
```

```
public:
    node(int x)
    {
        data = x;
        next = NULL;
    }
    friend class SLL;
};

class SLL
{
    node* head;
public:
    SLL()
    {
        head = NULL;
    }
    void create();
    void display();
    void add(int);
    void remove();
    int size();
    void intersection(SLL,SLL);
    void unio(SLL,SLL);
    void diff(SLL,SLL);
    void subset(SLL);
```

```
};
```

**Pseudo code:**

```
void SLL::create()
{
    char arr[5];
    int x;
    node* p;
    while(1)
    {
        cout << "Enter data: ";
        cin.getline(arr,5);
        if(strcmp(arr,"stop") == 0)
        {
            return;
        }
        x = atoi(arr);
        if(head == NULL)
        {
            head = new node(x);
            p = head;
        }
        else
        {
            int flag = 0;
            node* q = head;
            while(q != NULL)
            {
                if(q->data == x)
                {
                    flag = 1;
                    break;
                }
                q = q->next;
            }
            if(flag == 1)
            {
                cout << "Repeat.\n";
            }
            else
            {
                p->next = new node(x);
                p = p->next;
            }
        }
    }
}

void SLL::display()
{
    node* p = head;
    while(p != NULL)
    {
        cout << p->data << " ";
        p = p->next;
    }
    cout << endl;
}

void SLL::add(int x)
{
    /*int x;
    cout << "Enter element to be added: ";
    cin >> x;*/
    node* p = head;
    if(p == NULL)
    {
        head = new node(x);
    }
    else
    {
        while(p->next != NULL)
        {
            p = p->next;
        }
        p->next = new node(x);
    }
}
```

```
void SLL::remove()
{
    int x;
    cout << "Enter element to be removed: ";
    cin >> x;
    node* p = head;
    if(p == NULL)
    {
        cout << "List empty.\n";
        return;
    }
    else
    {
        node* q = NULL;
        while(p != NULL)
        {
            if(p->data == x)
            {
                break;
            }
            q = p;
            p = p->next;
        }
        if(q == NULL)
        {
            head = head->next;
            delete p;

            return;

        }

        if(p == NULL)

        {

            cout << "Element not found.\n";

            return;

        }

        q->next = p->next;

        delete p;

    }

}


int SLL::size()

{

    node* p = head;

    int cnt = 0;

    while(p != NULL)

    {

        cnt++;

        p = p->next;

    }

    return cnt;

}


void SLL::unio(SLL a,SLL b)
```

```
{
   node* p = a.head;
   node* q = b.head;
   node* r = head;
   int flag = 0;
   while(p != NULL)
   {
      add(p->data);
      p = p->next;
   }
   while(q != NULL)
   {
      node* p = a.head;
      while(p != NULL)
      {
         if(q->data == p->data)
         {
            flag = 0;
            break;
         }
         else
         {
            flag = 1;
            p = p->next;
         }
      }
```

```
        if(flag == 1)
        {
            add(q->data);
            flag = 0;
        }
        q = q->next;
    }
}

void SLL::intersection(SLL a,SLL b)
{
    node* p = a.head;
    node* q = b.head;
    int flag = 0;
    while(q != NULL){
        node* p = a.head;
        while(p != NULL){
            if(q->data == p->data){
                flag = 1;
                break;
            }
            else{
                flag = 0;
                p = p->next;
            }
        }
        if(flag == 1){
            add(q->data);
            flag = 0;
        }
        q = q->next;
    }
}

void SLL::diff(SLL a,SLL b)
{
    node* p = a.head;
    node* r = head;
    while(p != NULL)
    {
        node* q = b.head;
        int flag = 0;
        while(q != NULL)
        {
            if(q->data == p->data)
            {
                flag = 1;
                break;
            }
            q = q->next;
        }
        if(flag == 0)
        {
            add(p->data);
        }
        p = p->next;
    }
}

void SLL::subset(SLL a)
{
    node* q = a.head;
    while(q != NULL)
    {
        node* p = head;
        int flag = 0;
        while(p != NULL)
        {
            if(p->data == q->data)
            {
                flag = 1;
                break;
            }
            p = p->next;
        }
        if(flag == 0)
        {
            cout << "Not a subset.\n";
            return;
        }
        q = q->next;
    }
    cout << "Subset";
}
```

**Test cases:**

| Input | Output | Result |
|---|---|---|
| i)A={2,1,3,7,9,-1} | A={2,1,3,7,9} | success |
| ii)  Add 4 | A={2,1,3,7,9,4} | success |
| iii)Remove 7 | A={2,1,3,9,4} | success |
| iv)size | 5 | success |
| v) find 3 | true | success |
| vi) union: {5, 6} | union={2,1,3,9,4,5,6} | success |
| vii)intersection:{2,5} | inter={2} | success |
| viii)difference:{2,3} | diff={1,4,9} | success |
| ix)subset:{1,4} | Subset | success |

**Conclusion:** We have understood and implemented set and performed basic operations successfully.