```
    BstNode *left;
    BstNode * right;
};
```

BSTNode

Singly LinkedList

```
struct Node
{
    int info;
    struct node * next;
};
typedef Node* Nptr;
class SLL
{
    Nptr Head;
    public:
        SLLC)
        { Head = NULL; }
        Nptr Getnode( )
        {
            return new Node;
        }
        void CreateSLL(int Value)
        {
            Head = Getnode C);
            Head->info = value;
            Head->next = NULL;
        }
```

Nptr

```
Nptr Search_Node (int Value)
{
   //Start from the first Node
      Nptr current = Head;
   // if list is empty
      if(Head == NULL)
      {   return NULL; }
   // navigate through list
      while (current->info != value)
      {   // if it is last node
         if(current->next == NULL)
         {   return NULL; }
         else
         {  // go to next link
            current = (current->next; }
   // if data found, return the current node
      return current;
}

void InsertAfter (int elt1, int elt2)
{
   Nptr p = Search_Node (elt1);
      if (P != NULL)
      {
         Nptr q = getnode();
            q->info = elt2;
            q->next = p->next;
               p->next = q;
      }
   else
      cout<<"\n Can't Insert";
}
```

```
Void InsertBefor(int elt1, int elt2)
{
    Nptr p = SearchNode(elt1);
    if (P != NULL)
    {
        Nptr q = GetNode();
        q->info = elt2;
        q->next = p;
        if (P != Head)
        {
            Nptr r = Head;
            while (r->next != p)
                r = r->next;
            r->next = q;
        }
        else
            Head = q;
    }
    else
        cout<<"\n Can't Insert";
}

Void DeleteNode(int elt)
{
    Nptr P = SearchNode(elt);
    if (P == NULL)
    {
        cout<<"\n Element Can't be deleted";
    }
    else if (P == Head)
    {
        Head = Head->next;
    }
}
```

```
else
{   Nptr current = Headd;
    while(current->next != p)
        current = current->next;

    current->next = p->next;

}

    delete p;
}

void update_Node (int elt1, int elt2)
{   Nptr p = Search_Node(elt1);
    if (P != NULL)
    {  p->info = elt2;
       cout<<"\n Node updated";
    }
    else
    {  cout<<"\n Error";

}
int
void Size( )
{  Nptr p = int count=0;
   for(Nptr p = Head ; p != NULL; p=p->next, count++)
       return count;
}
```

```
void Display()
{
    Nptr p = Head;
    if(Head == NULL)
    {
        cout<<"\n Empty List";
    }
    else
    {
        cout<<"\n SLL is =>";
        while( p != NULL)
        {
            cout<<p->info<<"->";
            p = p->next;
        }
        cout<<"NULL"<<endl;
    }
}
```

Doubly linked list

```
struct Node
{   int info;
    struct Node * next;
    struct Node * previous;
};
typedef Node * Nptr;
class DLL
{       Nptr Head;
        public()
          DLL()
        {     Head = NULL;
        }
        Nptr  GetNode()
        {     return new Node;
        }
        Void  CreateDLL(int elt)
       {      Head = GetNode();
              Head->info = elt;
             Head->next = Head->previous = NULL;
       }
    Nptr searchNode(int elt1)
                  current
    {     Nptr P = Head;
          if (Head == NULL)
          {   return NULL; }
          while( current->info != elt1)
          {     if(current->next == NULL)
                {  return NULL; }
```

```
        else
        { current = current->next; }
    }
    return current;
}.

Void InsertAfter(int elt1, int elt2)
{
    Nptr p = SearchNode(elt1)
    if(p!=NULL)
    {
        Nptr q = GetNode();
        q->info = elt2;
        q->next = p->next;
        q->previous = p;
        p->next->previous = q;
        p->next = q;
    }
    cout<<"\n Error";
}.

Void InsertBefore(int elt1, int elt2)
{
    Nptr p = SearchNode(elt1);
    if(p!=NULL)
    {
        Nptr q = GetNode();
        q->info = elt2;
        q->next = p;
        q->previous = p->previous
        if(p==Head)
        {
            Head = q;
            p->previous = q;
        }
```

```
        else
        {   p->previous->next = q;
            p->previous = q;
        }
    }
void DeleteNode(int elt1)
{   Nptr qq = SearchNode(elt1);
    if (P == NULL)
    {  cout<<"\n Error"; }
    else
    {  if (P->previous != NULL)
           P->previous->next = p->next;
       if (p->next != NULL)
        p->next->previous = p->previous;
        if (qq == Head)
            Head = p->next;
        delete p;
    }
}
void size()
{   Nptr p = Head;
    int count = 0;
for( ; p != NULL; p = p->next, count++);
    cout<<"size of DLL - " << count;
}
```

```
Void Reverse()
{
    Nptr ptr1, ptr2;
    if(Head==NULL)
    { cout<<"\n Empty list"; }
    if (Head->next==NULL)
    { cout<<"\n only one element";
    }
    else
        ptr1 = Head;
        ptr2 = Ptr1->next;
        Ptr1->next = NULL;
        ptr1->previous = ptr2;
        while (ptr2!=NULL)
        {
            ptr2->previous = ptr2->next;
            ptr2->next = ptr1;
            ptr1 = ptr2;
            ptr2 = ptr2->previous;
        }
        Head = ptr1;
        cout<<"\n List Reversed"<<endl;
}
Void Display()
{
    if(Head==NULL)
        cout<<"\n Empty list";
    else
    {
        Nptr p = Head;
        cout<<"\n List is";
        while(p!=NULL)
        {
            cout<<p->info<<"->";
            p=p->next;
        }
        cout<<"NULL"<<endl;
}
```

Circular Linked List

```
struct Node
{   int info;
    struct Node* next;
};

typedef Node* Nptr;
class CLL
{
    Nptr Head;
    public:
        CLL()
        {
            Head = NULL;
        }

        Nptr GetNode()
        {
            return new Node;
        }

        Void CreateCLL(int elt)
        {
            Nptr p = Getnode();
            p->info = elt;
            p->next = p;
            Head = p;
        }

        Nptr Search_Node(int elt1)
        {
            else{
            Nptr p = Head;
if(Head==NULL)
{
    return
    NULL;
}
            while( p->info != elt1 && p->next != Head)
                p = p->next;
            if( p->info == elt1)
                return p;
            else
                return NULL;
        }
};
```

```
Void InsertAfter (int elt1, int elt2)
{
    Nptr p = SearchNode(elt1);
    if(P != NULL)
    {
        Nptr q = GetNode();
        q->info = elt2;
        q->next = p->next;
        p->next = q;
    }
    else
        cout<<"\n Error";
}

Void InsertBefore (int elt1, int elt2)
{
    Nptr p = SearchNode(elt1);
    if(P != NULL)
    {
        Nptr q = GetNode();
        q->info = elt2;
        q->next = p;
        if(P != Head)
        {
            Nptr r = Head;
            while(r->next != p)
                r = r->next;
            r->next = q;
        }
        else
            Head = q;
    }
    else
        cout<<"\n error";
}
```

```
Void DeleteNode(int ell I)
{   Nptr P= searchNode(ell I);
    if (P==NULL)
    { cout<<"\n ELEMENT NOT FOUND-";
    }
    else if(P==Head)
    { Head = Head->next;
    }
    else
    { Nptr current = Head;
        while(current->next != p)
            Current = Current->next;
        current->next = p->next;
    }
        delete p;
}

Void size()
{   Nptr   if(Head == NULL)
           { cout<<"\n List is empty";
           }
           int count =0;
        else
           { Nptr p = Head;
             while(p->next != Head)
             { p=p->next;
               count++;
             }
        cout<<"\n size of CLL ="<<count+1;
}
```

```
void display()
{
    if(Head==NULL)
    {
        cout<<"\n Nothing to display";
    }
    else{
        cout<<"\n List is ";
        Nptr p = Head;
        while(p->next!=Head)
        {
            cout<<p->info<<"->";
            p=p->next;
        }
        cout<<p->info;
    }
}
```