DSA(Lecture#5)

Prepared By Bal Krishna Subedi

Linked List:

a) Concept and definition
b) Inserting and deleting nodes
c) Linked implementation of a stack (PUSH / POP)
d) Linked implementation of a queue (insert / delete)
e) Circular linked list
   - Stack as a circular list (PUSH / POP)
   - Queue as a circular list (Insert / delete)
f) Doubly linked list (insert / delete)

Self referential structure:

It is sometimes desirable to include within a structure one member that is a pointer to the parent structure type. Hence, a structure which contains a reference to itself is called self-referential structure. In general terms, this can be expressed as:

```
struct node
{
        member 1;
        member 2;
        .......
        struct node *name;
};
```

For example,

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

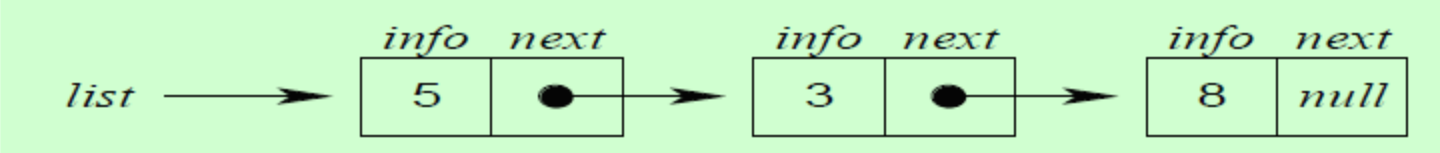
This is a structure of type node. The structure contains two members: a *info* integer member, and a pointer to a structure of the same type (i.e., a pointer to a structure of type node), called next. Therefore this is a ***self-referential*** structure.

Linked List:

A linked list is a collection of nodes, where each node consists of two parts:

- **info:** the actual element to be stored in the list. It is also called data field.
- **link:** one or two links that points to next and previous node in the list. It is also called next or pointer field.

Illustration:



- The nodes in a linked list are not stored contiguously in the memory
- You don't have to shift any element in the list.
- Memory for each node can be allocated dynamically whenever the need arises.
- The size of a linked list can grow or shrink dynamically

Operations on linked list:

The basic operations to be performed on the linked list are as follows:

- **Creation:** This operation is used to create a linked list
- **Insertion:** This operation is used to insert a new nose in a kinked list in a specified position. A new node may be inserted
  - ✔At the beginning of the linked list
  - ✔At the end of the linked list
  - ✔At he specified position in a linked list
- **Deletion:** The deletion operation is used to delete a node from the linked list. A node may be deleted from
  - ✔The beginning of the linked list
  - ✔the end of the linked list
  - ✔the specified position in the linked list.
- **Traversing:** The list traversing is a process of going through all the nodes of the linked list from on end to the other end. The traversing may be either forward or backward.
- **Searching or find:** This operation is used to find an element in a linked list. In the desired element is found then we say operation is successful otherwise unsuccessful.
- **Concatenation:** It is the process of appending second list to the end of the first list.

Types of Linked List:

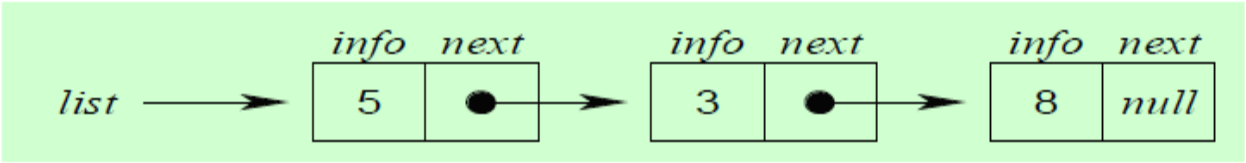
basically we can put linked list into the following four types:

- Singly linked list
- doubly linked list
- circular linked list
- circular doubly linked list

Singly linked list:

A singly linked list is a dynamic data structure which may grow or shrink, and growing and shrinking depends on the operation made. In this type of linked list each node contains two fields one is info field which is used to store the data items and another is link field that is used to point the next node in the list. The last node has a NULL pointer.

The following example is a singly linked list that contains three elements 5, 3, 8.



Representation of singly linked list:

We can create a structure for the singly linked list the each node has two members, one is **info** that is used to store the data items and another is **next** field that store the address of next node in the list.

We can define a node as follows:

```
struct Node
{
        int info;
        struct Node *next;
};
typedef struct Node NodeType;
NodeType *head;  //head is a pointer type structure variable
```

This type of structure is called self-referential structure.

- ***The NULL value of the next field of the linked list indicates the last node and we define macro for NULL and set it to 0 as below:***

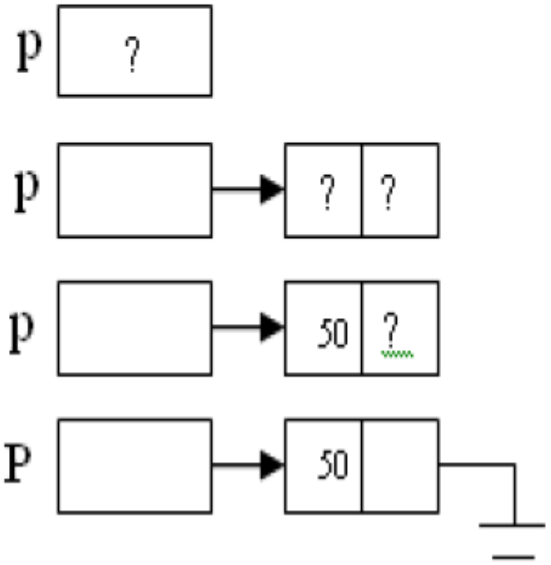
#define NULL 0

Creating a Node:

- To create a new node, we use the **malloc** function to dynamically allocate memory for the new node.
- After creating the node, we can store the new item in the node using a pointer to that nose.

The following steps clearly shows the steps required to create a node and storing an item.

```
Nodetype *p;
P=(NodeType*)malloc(sizeof(Nodetype));
p->info=50;
p->next=NULL;
```



Note that p is not a node; instead it is a pointer to a node.

The getNode function:

we can define a function getNode() to allocate the memory for a node dynamically. It is user-defined function that return a pointer to the newly created node.

```
Nodetype *getNode()
{
        NodeType *p;
        p==(NodeType*)malloc(sizeof(NodeType));
        return(p);
}
```

Creating the empty list:

```
void createEmptyList(NodeType *head)
{
        head=NULL;
}
```

Inserting Nodes:

To insert an element or a node in a linked list, the following three things to be done:

- Allocating a node
- Assigning a data to info field of the node
- Adjusting a pointer and a new node may be inserted
- At the beginning of the linked list
- At the end of the linked list
- At the specified position in a linked list

Insertion requires obtaining a new node ans changing two links

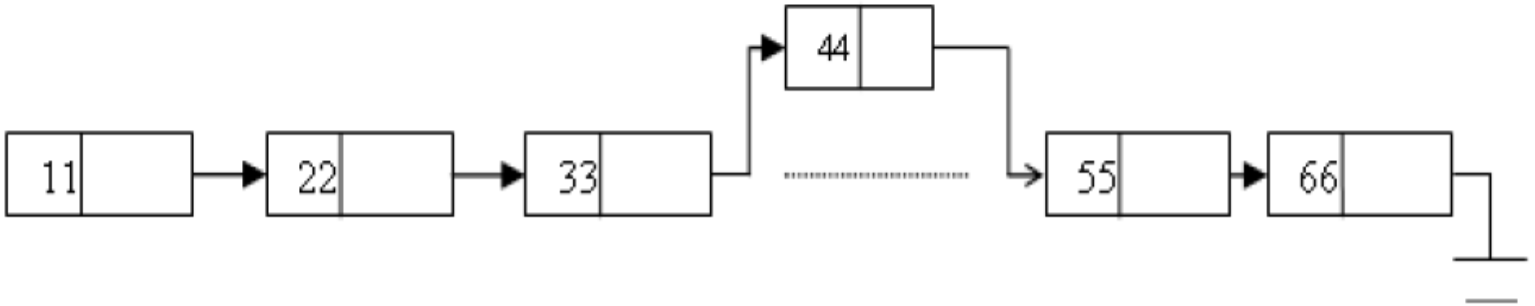


fig:- Inserting the new node with 44 between 33 and 55.

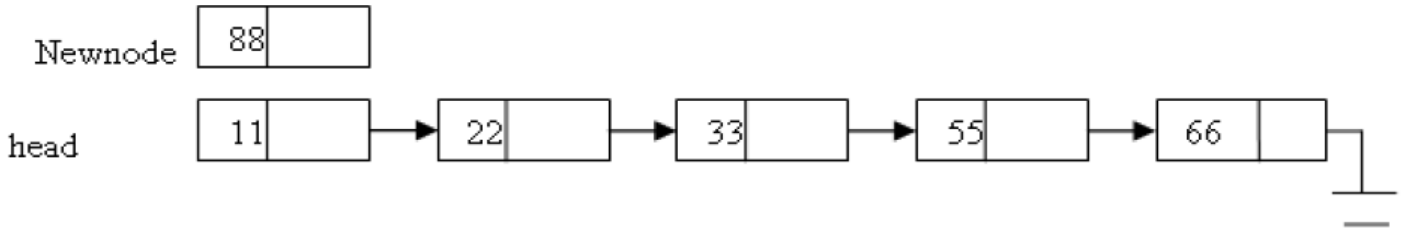
An algorithm to insert a node at the beginning of the singly linked list:

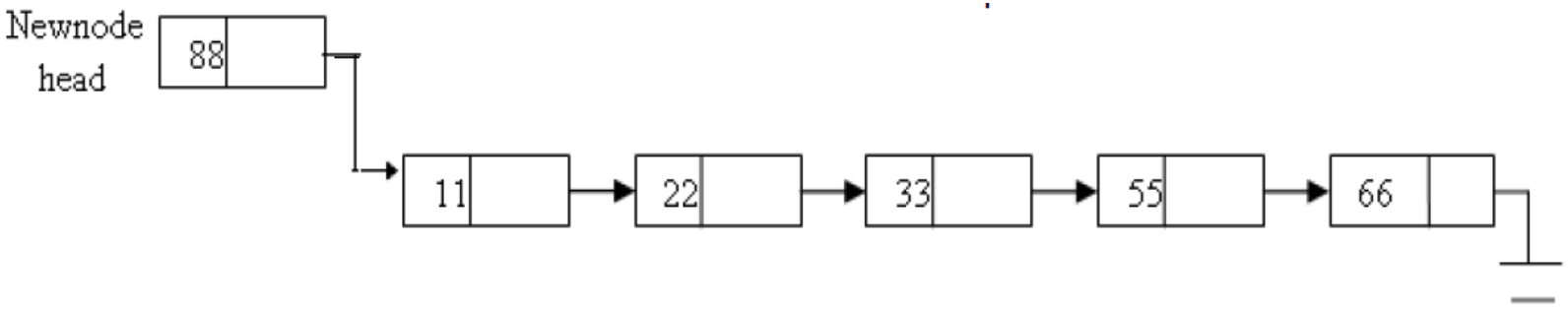
let *head be the pointer to first node in the current list

1. Create a new node using malloc function
   NewNode=(NodeType)malloc(sizeof(NodeType));*
2. Assign data to the info field of new node
   NewNode->info=newItem;
3. Set next of new node to head
   NewNode->next=head;
4. Set the head pointer to the new node
   head=NewNode;
5. End

The C function to insert a node at the beginning of the singly linked list:

```
void InsertAtBeg(int newItem)
{
        NodeType *NewNode;
        NewNode=getNode();
        NewNode->info=newItem;
        NewNode->next=head;
        head=NewNode;
}
```





An algorithm to insert a node at the end of the singly linked list:

let *head be the pointer to first node in the current list

1. Create a new node using malloc function
   NewNode=(NodeType)malloc(sizeof(NodeType));*
2. Assign data to the info field of new node
   NewNode->info=newItem;
3. Set next of new node to NULL
   NewNode->next=NULL;
4. if (head ==NULL)then
   Set head =NewNode.and exit.
5. Set temp=head;
6. while(temp->next!=NULL)
   temp=temp->next; //increment temp
7. *Set temp->next=NewNode;*
8. End

The C function to insert a node at the end of the linked list:

```
void InsertAtEnd(int newItem)
{
        NodeType *NewNode;
        NewNode=getNode();
        NewNode->info=newItem;
        NewNode->next=NULL;
        if(head==NULL)
        {
                head=NewNode;
        }
        else
        {
                temp=head;
                while(temp->next!=NULL)
                {
                        temp=temp->next;
                 }
                temp->next=NewNode;
        }
}
```

An algorithm to insert a node after the given node in singly linked list:

let *head be the pointer to first node in the current list and *p be the pointer to the node after which we want to insert a new node.

1. Create a new node using malloc function
   NewNode=(NodeType)malloc(sizeof(NodeType));*
2. Assign data to the info field of new node
   NewNode->info=newItem;
3. Set next of new node to next of p
   NewNode->next=p->next;
4. Set next of p to NewNode
   p->next =NewNode..
5. End

The C function to insert a node after the given node in singly linked list:

```
void InsertAfterNode(NodeType *p int newItem)
{
        NodeType *NewNode;
        NewNode=getNode();
        NewNode->info=newItem;
        if(p==NULL)
        {
                printf("Void insertion");
                exit(1);
        }
        else
        {
                NewNode->next=p->next;
                 p->next =NewNode..
        }
}
```

An algorithm to insert a node at the specified position in a singly linked list:

let *head be the pointer to first node in the current list

1. Create a new node using malloc function
   NewNode=(NodeType)malloc(sizeof(NodeType));*
2. Assign data to the info field of new node
   NewNode->info=newItem;
3. Enter position of a node at which you want to insert a new node. Let this position is pos.
4. Set temp=head;
5. if (head ==NULL)then
   printf("void insertion"); and exit(1).
6. for(i=1; i<pos-1; i++)
   temp=temp->next;
7. Set *NewNode->next=temp->next;*
   set temp->next =NewNode..
8. End

The C function to insert a node at the specified position in a singly linked list:

```
void InsertAtPos(int newItem)
{
        NodeType *NewNode;
        int pos , i ;
        printf(" Enter position of a node at which you want to insert a new node");
        scanf("%d",&pos);
        if(head==NULL)
        {
                printf("void insertion");
                exit(1).
        }
        else
        {
                temp=head;
```

```
            for(i=1; i<pos-1; i++)
            {
                temp=temp->next;
            }
            NewNode=getNode();
            NewNode->info=newItem;
            NewNode->next=temp->next;
            temp->next =NewNode;
        }
}
```

Deleting Nodes:

A node may be deleted:

- From the beginning of the linked list
- from the end of the linked list
- from the specified position in a linked list

Deleting first node of the linked list:

An algorithm to deleting the first node of the singly linked list:

let *head be the pointer to first node in the current list

1. If(head==NULL) then
   print "Void deletion" and exit
2. Store the address of first node in a temporary variable **temp.**
   temp=head;
3. Set head to next of head.
   head=head->next;
4. Free the memory reserved by temp variable.
   free(temp);
5. End

The C function to deleting the first node of the singly linked list:

```
void deleteBeg()
        {
                NodeType *temp;
                if(head==NULL)
                {
                        printf("Empty list");
                        exit(1).
                }
                else
                {
                        temp=head;
                        printf("Deleted item is %d" , head->info);
                        head=head->next;
                        free(temp);
                  }
        }
```

Deleting the last node of the linked list:

An algorithm to deleting the last node of the singly linked list:

let *head be the pointer to first node in the current list

```
1.  If(head==NULL) then            //if list is empty
        print "Void deletion" and exit
2.  else if(head->next==NULL) then      //if list has only one node
      Set temp=head;
      print deleted item as,
      printf("%d" ,head->info);
      head=NULL;
      free(temp);
3.  else
     set temp=head;
     while(temp->next->next!=NULL)
        set temp=temp->next;
      End of while
       free(temp->next);
      Set temp->next=NULL;
4.  End
```

The C function to deleting the last node of the singly linked list:

let *head be the pointer to first node in the current list

```
void deleteEnd()
{
    NodeType *temp;
    if(head==NULL)
    {
        printf("Empty list");
        return;
    }
    else if(head->next==NULL)
    {
            temp=head;
            head=NULL;
            printf("Deleted item is %d", temp->info);
            free(temp);
    }
    else
    {
            temp=head;
            while(temp->next->next!=NULL)
            {
                    temp=temp->next;
            }
            printf("deleted item is %d"' , temp->next->info):
            free(temp->next);
            temp->next=NULL;
    }
}
```

An algorithm to delete a node after the given node in singly linked list:

let *head be the pointer to first node in the current list and *p be the pointer to the node after which we want to delete a new node.

1. ***if****(p==NULL or p->next==NULL) then*
   print "deletion not possible and exit
2. set q=p->next
3. *Set p->next=q->next;*
4. **free**(q)
5. End

The C function to delete a node after the given node in singly linked list:

let *p be the pointer to the node after which we want to delete a new node.

```
void deleteAfterNode(NodeType *p)
{
        NodeType *q;
        if(p==NULL || p->next==NULL )
        {
                printf("Void insertion");
                exit(1);
        }
        else
        {
                q=p->next;
                p->next=q->next;
                free(q);
        }
}
```

An algorithm to delete a node at the specified position in a singly linked list:

let *head be the pointer to first node in the current list

1. *Read position of a node which to be deleted, let it be pos.*
2. if head==NULL
   print "void deletion" and exit
3. Enter position of a node at which you want to delete a new node. Let this position is pos.
4. Set temp=head
   declare a pointer of a structure let it be *p
5. if (head ==NULL)then
   print "void ideletion" and exit
   otherwise;.
6. for(i=1; i<pos-1; i++)
   temp=temp->next;
7. *print deleted item is temp->next->info*
8. *Set p=temp->next;*
9. Set temp->next =temp->next->next;
10. free(p);
11. End

The C function to delete a node at the specified position in a singly linked list

```
void deleteAtSpecificPos()
{
    NodeType *temp *p;
    int pos, i;
    if(head==NULL)
    {
        printf("Empty list");
        return;
    }
    else
    {
            printf("Enter position of a node which you wand to delete");
            scanf("%d" , &pos);
            temp=head;
            for(i=1; i<pos-1; i++)
            {
                    temp=temp->next;
            }
            p=temp->next;
            printf("Deleted item is %d", p->info);
            temp->next =p->next;
            free(p);
     }
}
```

Searching an item in a linked list:

To search an item from a given linked list we need to find the node that contain this data item. If we find such a node then searching is successful otherwise searching unsuccessful.

*let *head be the pointer to first node in the current list*

```
void searchItem()
{
    NodeType *temp;
    int key;
    if(head==NULL)
    {
            printf("empty list");
            exit(1);
    }
    else
    {
            printf("Enter searched item");
            scanf('%d" ,&key);
            temp=head;
            while(temp!=NULL)
            {
                    if(temp->info==key)
                    {
                            printf("Search successful");
                            break;
                    }
                    temp=temp->next;
            }
            if(temp==NULL)
            printf("Unsuccessful search");
    }
}
```

Complete program:

/***Various operations on singly linked list**********/***

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>    //for malloc function
#include<process.h>   //fpr exit function
struct node
{
        int info;
        struct node *next;
};
typedef struct node NodeType;
NodeType *head;
head=NULL;
void insert_atfirst(int);
void insert_givenposition(int);
void insert_atend(int);
void delet_first();
void delet_last();
void delet_nthnode();
void info_sum();
void count_nodes();
void main()
{
        int choice;
        int item;
        clrscr();
        do
        {
```

```
printf("\n manu for program:\n");
printf("1. insert first \n2.insert at given position \n3 insert at last \n 4:Delete first
node\n 5:delete last node\n6:delete nth node\n7:count nodes\n8Display items\n10:exit\n");
printf("enter your choice\n");
scanf("%d",&choice);
switch(choice)
{
        case 1:
                printf("Enter item to be inserted");
                scanf("%d", &item)
                insert_atfirst(item);
                break;
        case 2:
                printf("Enter item to be inserted");
                scanf("%d", &item)
                insert_givenposition(item);
                break;
        case 3:
                printf("Enter item to be inserted");
                scanf("%d", &item)
                insert_atend();
                break;
        case 4:
                delet_first();
                break;
        case 5:
                delet_last();
                break;
        case 6:
                delet_nthnode();
                break;
        case 7:
                info_sum();
                break;
        case 8:
```

```
                count_nodes();
                break;
            case 9:
                exit(1);
                break;
            default:
                printf("invalid choice\n");
                break;
        }
    }while(choice<10);
    getch();
}

/***********function definitions**************/
void insert_atfirst(int item)
{
    NodeType *nnode;
    nnode=(NodeType*)malloc(sizeof(NodeType));
    nnode->info=item;
    nnode->next=head;
    head=nnode;
}
```

```
void insert_givenposition(int item)
{
        NodeType *nnode;
        NodeType *temp;
        temp=head;
        int p,i;
        nnode=( NodeType *)malloc(sizeof(NodeType));
        nnode->info=item;
        if (head==NULL)
        {
           nnode->next=NULL;
           head=nnode;
        }
        else
        {
             printf("Enter Position of a node at which  you want to insert an new node\n");
            scanf("%d",&p);
                for(i=1;i<p-1;i++)
                {
                        temp=temp->next;
                }
                nnode->next=temp->next;
                temp->next=nnode;
        }
}
```

```
void insert_atend(int item)
  {
        NodeType *nnode;
        NodeType *temp;
        temp=head;
        nnode=( NodeType *)malloc(sizeof(NodeType));
        nnode->info=item;

            if(head==NULL)
            {
               nnode->next=NULL;
               head=nnode;
            }
            else
            {
              while(temp->next!=NULL)
              {
                      temp=temp->next;
              }
              nnode->next=NULL;
              temp->next=nnode;
            }
   }
```

```
void delet_first()
 {
        NodeType *temp;
        if(head==NULL)
        {
           printf("Void deletion|n");
           return;
        }
        else
        {
            temp=head;
            head=head->next;
            free(temp);
        }
}

 void delet_last()
 {
        NodeType *hold,*temp;
        if(head==NULL)
        {
           printf("Void deletion|n");
           return;
        }
        else if(head->next==NULL)
        {
            hold=head;
            head=NULL;
            free(hold);
        }
        else
        {
           temp=head;
          while(temp->next->next!=NULL)
          {
            temp=temp->next;
          }
          hold=temp->next;
          temp->next=NULL;
          free(hold);
        }
      }
```

```
void delet_nthnode()
 {
        NodeType *hold,*temp;
        int pos, i;
        if(head==NULL)
        {
           printf("Void deletion|n");
           return;
        }
        else
        {
          temp=head;
          printf("Enter position of node which node is to be deleted\n");
          scanf("%d",&pos);
          for(i=1;i<pos-1;i++)
          {
            temp=temp->next;
          }
          hold=temp->next;
          temp->next=hold->next;
          free(hold);
        }
 }
```

```
void info_sum()
 {
    NodeType *temp;
    temp=head;
    while(temp!=NULL)
    {
          printf("%d\t",temp->info);
          temp=temp->next;
    }
 }
void count_nodes()
 {
    int cnt=0;
    NodeType *temp;
    temp=head;
    while(temp!=NULL)
    {
          cnt++;
          temp=temp->next;
    }
    printf("total nodes=%d",cnt);
 }
}
```

Linked list implementation of Stack:

Push function:

let *top be the top of the stack or pointer to the first node of the list.

```
void push(item)
{
        NodeType *nnode;
        int data;
      nnode=( NodeType *)malloc(sizeof(NodeType));
      if(top==0)
      {
        nnode->info=item;
        nnode->next=NULL;
        top=nnode;
      }
      else
      {
       nnode->info=item;
       nnode->next=top;
       top=nnode;
      }
}
```

Pop function:

let *top be the top of the stack or pointer to the first node of the list.

```
void pop()
{
        NodeType *temp;
        if(top==0)
        {
                printf("Stack contain no elements:\n");
                return;
        }
        else
        {
                temp=top;
                top=top->next;
                printf("\ndeleted item is %d\t",temp->info);
                free(temp);
        }
}
```

A Complete C program for linked list implementation of stack:

/***********Linked list implementation of stack**************/***

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
#include<process.h>
struct node
{
        int info;
        struct node *next;
};
typedef struct node NodeType;
NodeType *top;
top=0;
void push(int);
void pop();
void display();
void main()
{
        int choice, item;
        clrscr();
```

```
        do
        {
           printf("\n1.Push \n2.Pop \n3.Display\n4:Exit\n");
           printf("enter ur choice\n");
           scanf("%d",&choice);
           switch(choice)
           {
                  case 1:
                         printf("\nEnter the data:\n");
                         scanf("%d",&item);
                         push(item);
                         break;
                  case 2:
                          pop();
                          break;
                  case 3:
                         display();
                         break;
                  case 4:
                         exit(1);
                         break;
                  default:
                         printf("invalid choice\n");
                         break;
           }
        }while(choice<5);
        getch();

}
```

```
/**************push function*******************/
        void push(int item)
        {
                NodeType *nnode;
                int data;
                nnode=( NodeType *)malloc(sizeof(NodeType));
                if(top==0)
                {
                        nnode->info=item;
                        nnode->next=NULL;
                        top=nnode;
                }
                else
                {
                         nnode->info=item;
                        nnode->next=top;
                        top=nnode;
                }
        }
/******************pop function*********************/
        void pop()
        {
                NodeType *temp;
                if(top==0)
                {
                        printf("Stack contain no elements:\n");
                        return;
```

```
        }
        else
        {
                temp=top;
                top=top->next;
                printf("\ndeleted item is %d\t",temp->info);
                free(temp);
        }
}

/**************display function************************/
void display()
{
        NodeType *temp;
        if(top==0)
        {
                printf("Stack is empty\n");
                return;
        }
        else
        {
                temp=top;
                printf("Stack items are:\n");
                while(temp!=0)
                {
                        printf("%d\t",temp->info);
                        temp=temp->next;
                }
        }
}
```

Thank You

Any Queries?