1. Infix to Postfix

```
#include <stdlib.h>
#include <ctype.h>
#define SIZE 20
struct stack
{
    int top;
    char data[SIZE];
};
typedef struct stack STACK;
void push(STACK *s,char item)
{
    s->data[++(s->top)]=item;
}

char pop(STACK *s)
{
    return s->data[(s->top)--];
}

int preced(char symbol)
{
    switch(symbol)
    {
        case '^':return 5;
        case '*':
           case '/':return 3;
        case '+':
           case '-':return 1;
    }
}

void infixtopostfix(STACK *s,char infix[SIZE])
{
    int i,j=0;
    char postfix[SIZE],temp,symbol;
    for(i=0;infix[i]!='\0';i++)
    {
        symbol=infix[i];
        if(isalnum(symbol))
            postfix[j++]=symbol;
        else
        {
            switch(symbol)
            {
                case '(':push(s,symbol);
                          break;
                case ')':temp=pop(s);
                         while(temp!='(')
                         {
                             postfix[j++]=temp;
                             temp=pop(s);
                         }
                         break;
```

```
                case '+':
                case '-':
                case '*':
                case '/':
                case '^': if (s->top ==-1 || s->data[s->top]=='(')
                                push(s,symbol);
                        else
                        {
                            while(preced(s->data[s->top])>=
preced(symbol) && s->top!=-1 &&s->data[s->top]!='(')
                                    postfix[j++]=pop(s);
                            push(s,symbol);
                        }
                        break;
                default :printf("\n Invalid!!!!!");
                         exit(0);

            }
        }
    }
    while(s->top!=-1)
        postfix[j++]=pop(s);
    postfix[j]='\0';
    printf("\n The postfix expression is %s\n",postfix);
}
int main()
{
    STACK s;
    s.top=-1;
    char infix[SIZE];
    printf("\n Read Infix expression\n");
    scanf("%s",infix);
    infixtopostfix(&s,infix);
    return 0;
}
```

2. Evaluation of Prefix

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>
#include <string.h>
#define SIZE 20
struct stack
{
    int top;
    float data[SIZE];
};
typedef struct stack STACK;
void push(STACK *s,float item)
{
    s->data[++(s->top)]=item;
}

float pop(STACK *s)
{
    return s->data[(s->top)--];
}

float operate(float op1,float op2,char symbol)
{
    switch(symbol)
    {
        case '+':return op1+op2;
        case '-':return op1-op2;
        case '*':return op1*op2;
        case '/':return op1/op2;
        case '^':return pow(op1,op2);
    }
}

float eval(STACK *s,char prefix[SIZE])
{
    int i;
    char symbol;
    float res,op1,op2;
    for(i=strlen(prefix)-1;i>=0;i--)
    {
        symbol=prefix[i];
        if(isdigit(symbol))
           push(s,symbol-'0');
        else
        {
            op1=pop(s);
            op2=pop(s);
            res=operate(op1,op2,symbol);
            push(s,res);
        }
    }
    return pop(s);
```

```
}

int main()
{
    char prefix[SIZE];
    STACK s;
    float ans;
    s.top=-1;
    printf("\n Read prefix expr\n");
    scanf("%s",prefix);
    ans=eval(&s,prefix);
    printf("\n The final answer is %f\n",ans);
    return 0;
}
```

3. Message Queueing System

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define SIZE 5
struct queue
{
    int front,rear;
    char data[SIZE][20];
};
typedef struct queue QUEUE;

void send(QUEUE *q,char item[20])
{
    if(q->front==(q->rear+1) % SIZE )
        printf("\n Queue full");
    else
    {
        q->rear=(q->rear+1)%SIZE;
        strcpy(q->data[q->rear],item);
        if(q->front==-1)
            q->front=0;
    }
}

char *receive(QUEUE *q)
{
    char *del;
    if(q->front==-1)
    {
        printf("\n Queue empty");
        return -1;
    }
    else
    {
        del=q->data[q->front];
        if(q->front==q->rear)
        {
            q->front=-1;
            q->rear=-1;
        }
        else
            q->front=(q->front+1)% SIZE;
        return del;
    }
}

void display(QUEUE q)
{
    int i;
    if(q.front==-1)
        printf("\n Queue Empty");
    else
    {
```

```
        printf("\n Queue content are\n");
        for(i=q.front;i!=q.rear;i=(i+1)%SIZE)
            printf("%s\n",q.data[i]);
        printf("%s\n",q.data[i]);
    }

}

int main()
{
    int ch;
    char *del;
    char item[20];
    QUEUE q;
    q.front=-1;
    q.rear=-1;
    for(;;)
    {
    printf("\n1. Send\n2. Receive\n3. Display\n4. Exit");
    printf("\nRead Choice :");
    scanf("%d",&ch);
    getchar();
    switch(ch)
    {
        case 1:printf("\n Read msg to be send :");
               gets(item);
               send(&q,item);
               break;
        case 2:del=receive(&q);
                if(del!=NULL)
                    printf("\n Element deleted is %s\n",del);
                break;
        case 3:display(q);
                break;
        default:exit(0);
    }
    }
    return 0;

}
```

4. Multiplication of Polynomials using Singly Linked List

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
int count;
struct node
{
    int co,po;
    struct node *addr;
};
typedef struct node *NODE;
NODE insertend(NODE start,int co,int po)
{
    NODE temp,cur;
    temp=(NODE)malloc(sizeof(struct node));
    temp->co=co;
    temp->po=po;
    temp->addr=NULL;
    if(start==NULL)
        return temp;
    cur=start;
    while(cur->addr!=NULL)
        cur=cur->addr;
    cur->addr=temp;
    return start;
}

void display(NODE start)
{
    NODE temp;
    if(start==NULL)
        printf("\n Polynomial Empty");
    else
    {
        temp=start;
        while(temp->addr!=NULL)
        {
            printf("%dx^%d+",temp->co,temp->po);
            temp=temp->addr;
        }
            printf("%dx^%d\n",temp->co,temp->po);
    }
}

NODE addterm(NODE res,int co,int po)
{
    NODE temp,cur;
    temp=(NODE)malloc(sizeof(struct node));
    temp->co=co;
    temp->po=po;
    temp->addr=NULL;
    if(res==NULL)
```

```
        return temp;
    cur=res;
    while(cur!=NULL)
    {
        if(cur->po==po)
        {
            cur->co=cur->co+co;
            return res;
        }
        cur=cur->addr;
    }
    if(cur==NULL)
        res=insertend(res,co,po);
    return res;
}
NODE multiply(NODE poly1,NODE poly2)
{
    NODE p1,p2,res=NULL;
    for(p1=poly1;p1!=NULL;p1=p1->addr)
        for(p2=poly2;p2!=NULL;p2=p2->addr)
             res=addterm(res,p1->co*p2->co,p1->po+p2->po);
    return res;
}
int main()
{
    NODE poly1=NULL,poly2=NULL,poly;
    int co,po;
    int i,n,m;
    printf("\nRead no of terms of first polynomial:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        printf("\n Read CO and PO of %d term : ",i);
        scanf("%d%d",&co,&po);
        poly1=insertend(poly1,co,po);
    }
    printf("\n First polynomial is\n");
    display(poly1);
    printf("\nRead no of terms of second polynomial:");
    scanf("%d",&m);
    for(i=1;i<=m;i++)
    {
        printf("\n Read CO and PO of %d term : ",i);
        scanf("%d%d",&co,&po);
        poly2=insertend(poly2,co,po);
    }
    printf("\n Second polynomial is\n");
    display(poly2);
    poly=multiply(poly1,poly2);
    printf("\n Resultant polynomial is\n");
    display(poly);
    return 0;

}
```

5. Queue of Integers using Circular List

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
int count;
struct node
{
    int data;
    struct node *addr;
};
typedef struct node *NODE;

NODE insertend(NODE last,int item)
{
    NODE temp;
    if(count>=SIZE)
        {
            printf("\ Queue full");
            return last;
        }
    count=count+1;
    temp=(NODE)malloc(sizeof(struct node));
    temp->data=item;
    if(last==NULL)
    {
        temp->addr=temp;
        return temp;
    }
    else
    {
        temp->addr=last->addr;
        last->addr=temp;
        return temp;
    }
}

NODE deletebegin(NODE last)
{
    NODE temp;
    if(last==NULL)
    {
        printf("\n Queue empty");
        return NULL;
    }
    if(last->addr==last)
    {
        printf("\n Element deleted is %d\n",last->data);
        free(last);
        return NULL;
    }
    else
    {
    temp=last->addr;
    last->addr=temp->addr;
```

```
    printf("\n Element deleted is %d\n",temp->data);
    free(temp);
    return last;
    }
}
void display(NODE last)
{
    NODE temp;
    if(last==NULL)
        printf("\n Queue is empty");
    else
    {
        printf("\n Queue Content are\n");
        temp=last->addr;
        while(temp!=last)
        {
            printf("%d\t",temp->data);
            temp=temp->addr;
        }
        printf("%d\t",temp->data);
    }
}
int main()
{
    NODE last=NULL;
    int item,ch;
    for(;;)
    {
        printf("\n1.Insert\n2.Delete\n3.Display\n4.Exit");
        printf("\nRead Choice :");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:printf("\n Read data to be inserted:");
                   scanf("%d",&item);
                   last=insertend(last,item);
                   break;
            case 2:last=deletebegin(last);
                   break;
            case 3:display(last);
                   break;
            default:exit(0);
        }
    }
    return 0;
}
```

6. Hashing

7. Priority Queue using Heap

```
#include <stdio.h>
#include <stdlib.h>

void heapify(int a[10],int n)
{
    int i,k,v,j,flag=0;
    for(i=n/2;i>=1;i--)
    {
        k=i;
        v=a[k];
        while(!flag && 2*k <= n)
        {
            j=2*k;
            if(j<n)
            {
                if(a[j]<a[j+1])
                    j=j+1;
            }
            if(v>=a[j])
                flag=1;
            else
            {
                a[k]=a[j];
                k=j;
            }
        }
        a[k]=v;
        flag=0;
    }
}

int main()
{
    int n,i,a[10],ch;
    for(;;)
    {
        printf("\n 1. Create Heap");
        printf("\n 2. Extractmax");
        printf("\n 3. Exit");
        printf("\n Read Choice :");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:printf("\n Read no of elements :");
                   scanf("%d",&n);
                   printf("\n Read Elements\n");
                   for(i=1;i<=n;i++)
                    scanf("%d",&a[i]);
                   heapify(a,n);
                   printf("\n Elements after heap\n");
                   for(i=1;i<=n;i++)
                    printf("%d\t",a[i]);
```

```
                    break;
            case 2:if(n>=1)
                    {
                        printf("\n Element deleted is %d\n",a[1]);
                        a[1]=a[n];
                        n=n-1;
                        heapify(a,n);
                        if(n!=0)
                        {
                            printf("\n Elements after reconstructing
heap\n");
                            for(i=1;i<=n;i++)
                               printf("%d\t",a[i]);
                        }
                    }
                    else
                        printf("\n No element to delete");
                    break;
            default:exit(0);

        }
    }
    return 0;
}
```

8. Expression Tree

```
#include <stdio.h>
#include <stdlib.h>
#include<ctype.h>
struct node
{
    char data;
    struct node *left;
    struct node *right;
};
typedef struct node *NODE;

struct stack
{
    int top;
    NODE data[10];
};
typedef struct stack STACK;

void push(STACK *s,NODE item)
{
    s->data[++(s->top)]=item;
}

NODE pop(STACK *s)
{
    return s->data[(s->top)--];
}

int preced(char symbol)
{
  switch(symbol)
  {
      case '$':return 5;
      case '*':
      case '/':return 3;
      case '+':
      case '-':return 1;
  }
}

NODE createnode(char item)
{
    NODE temp;
    temp=(NODE)malloc(sizeof(struct node));
    temp->data=item;
    temp->left=NULL;
    temp->right=NULL;
    return temp;
}

void preorder(NODE root)
{
```

```
    if(root!=NULL)
    {
        printf("%c",root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

void inorder(NODE root)
{
    if(root!=NULL)
    {
        inorder(root->left);
        printf("%c",root->data);
        inorder(root->right);
    }
}

void postorder(NODE root)
{
    if(root!=NULL)
    {
        postorder(root->left);
        postorder(root->right);
        printf("%c",root->data);
    }
}

NODE create_expr_tree(NODE root,char infix[10])
{
    STACK TS,OS;
    TS.top=-1;
    OS.top=-1;
    int i;
    char symbol;
    NODE temp,t;
    for(i=0;infix[i]!='\0';i++)
    {
        symbol=infix[i];
        temp=createnode(symbol);
        if(isalnum(symbol))
            push(&TS,temp);
        else
        {
            if(OS.top==-1)
                push(&OS,temp);
            else
            {
                while(OS.top!=-1 && preced(OS.data[OS.top]->data)>=
preced(symbol))
                {
                    t=pop(&OS);
                    t->right=pop(&TS);
                    t->left=pop(&TS);
                    push(&TS,t);
                }
```

```
                push(&OS,temp);
            }
        }

    }
    while(OS.top!=-1)
    {
         t=pop(&OS);
         t->right=pop(&TS);
         t->left=pop(&TS);
         push(&TS,t);
    }
    return pop(&TS);
}
int main()
{
    char infix[10];
    NODE root=NULL;
    printf("\n Read the infix expression :");
    scanf("%s",infix);
    root=create_expr_tree(root,infix);
    printf("\n The preorder traversal is\n");
    preorder(root);
    printf("\n The inorder traversal is\n");
    inorder(root);
    printf("\n The postorder traversal is\n");
    postorder(root);
    return 0;
}
```

9. Binary Tree

```
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int data;
    struct node *left;
    struct node *right;
};
typedef struct node *NODE;

NODE create_node(int item)
{
    NODE temp;
    temp=(NODE)malloc(sizeof(struct node));
    temp->data=item;
    temp->left=NULL;
    temp->right=NULL;
    return temp;
}

NODE insertleft(NODE root,int item)
{
    root->left=create_node(item);
    return root->left;
}

NODE insertright(NODE root,int item)
{
    root->right=create_node(item);
    return root->right;
}

void display(NODE root)
{
    if(root!=NULL)
    {
        display(root->left);
        printf("%d\t",root->data);
        display(root->right);
    }
}

int count_nodes(NODE root)
{
   if (root == NULL)
        return 0;
   else
    return (count_nodes(root->left) + count_nodes(root->right) + 1);
}

int height(NODE root)
{
```

```
    int leftht,rightht;
    if(root == NULL)
        return -1;
    else
    {
        leftht = height(root->left);
        rightht = height(root->right);
        if(leftht > rightht)
         return leftht + 1;
        else
         return rightht + 1;
    }
}

int leaf_nodes(NODE root)
{
    if(root==NULL)
        return 0;
    else if(root->left == NULL && root->right == NULL)
        return 1;
    else
        return leaf_nodes(root->left) + leaf_nodes(root->right);
}

int nonleaf_nodes(NODE root)
{
    if(root==NULL || (root->left == NULL && root->right == NULL))
        return 0;
    else
        return nonleaf_nodes(root->left) + nonleaf_nodes(root-
>right) + 1;
}
int main()
{
    NODE root=NULL;
    root=create_node(45);
    insertleft(root,39);
    insertright(root,78);
    insertleft(root->right,54);
    insertright(root->right,79);
    insertright(root->right->left,55);
    insertright(root->right->right,80);
    printf("\n The tree(inorder) is\n");
    display(root);
    printf("\n");
    printf("\n The total number of nodes is
%d\n",count_nodes(root));
    printf("\n The height of the tree is %d\n",height(root));
    printf("\n The total number of leaf nodes is
%d\n",leaf_nodes(root));
    printf("\n The total number of non-leaf nodes is
%d\n",nonleaf_nodes(root));
    return 0;
}
```

10. Binary Search Tree

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *left;
    struct node *right;
};
typedef struct node *NODE;

NODE create_node(int item)
{
    NODE temp;
    temp=(NODE)malloc(sizeof(struct node));
    temp->data=item;
    temp->left=NULL;
    temp->right=NULL;
    return temp;
}

NODE Insertbst(NODE root,int item)
{
    NODE temp;
    temp=create_node(item);
    if(root==NULL)
        return temp;
    else
    {
       if(item < root->data)
            root->left=Insertbst(root->left,item);
        else
            root->right=Insertbst(root->right,item);
    }
    return root;

}

void preorder(NODE root)
{
    if(root!=NULL)
    {
        printf("%d\t",root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

void inorder(NODE root)
{
    if(root!=NULL)
    {
        inorder(root->left);
        printf("%d\t",root->data);
```

```
        inorder(root->right);
    }
}

void postorder(NODE root)
{
    if(root!=NULL)
    {
        postorder(root->left);
        postorder(root->right);
        printf("%d\t",root->data);
    }
}

NODE inordersuccessor(NODE root)
{
    NODE cur=root;
    while(cur->left != NULL)
        cur = cur->left;
    return cur;
}

NODE deletenode(NODE root,int key)
{
    NODE temp;
    if(root == NULL)
        return NULL;
    if(key<root->data)
        root->left = deletenode(root->left,key);
    else if(key > root->data)
        root->right = deletenode(root->right,key);
    else
    {
        if(root->left == NULL)
        {
            temp=root->right;
            free(root);
            return temp;
        }
        if(root->right == NULL)
        {
            temp=root->left;
            free(root);
            return temp;
        }
        temp=inordersuccessor(root->right);
        root->data=temp->data;
        root->right=deletenode(root->right,temp->data);
    }
    return root;
}
int main()
{
    NODE root = NULL;
    int ch,item,key;
    for(;;)
```

```
    {
        printf("\n 1. Insert");
        printf("\n 2. Preorder");
        printf("\n 3. Inorder");
        printf("\n 4. Postorder");
        printf("\n 5. Delete");
        printf("\n 6. Exit");
        printf("\n Read ur choice:");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:printf("\n Read element to be inserted :");
                scanf("%d",&item);
                root=Insertbst(root,item);
                break;
            case 2:printf("\n The Preorder traversal is\n");
                 preorder(root);
                 break;
            case 3:printf("\n The Inorder traversal is\n");
                 inorder(root);
                 break;
            case 4:printf("\n The Postorder traversal is\n");
                 postorder(root);
                 break;
            case 5:printf("\n Read node to be deleted : ");
                  scanf("%d",&key);
                 root=deletenode(root,key);
                 break;
            default :exit(0);
        }
    }
    return 0;
}
```

PART B – PRACTICE PROGRAMS

1. Infix to Prefix

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#define SIZE 20
struct stack
{
    int top;
    char data[SIZE];
};
typedef struct stack STACK;
void push(STACK *s,char item)
{
    s->data[++(s->top)]=item;
}

char pop(STACK *s)
{
    return s->data[(s->top)--];
}

int preced(char symbol)
{
    switch(symbol)
    {
        case '^':return 5;
        case '*':
           case '/':return 3;
        case '+':
           case '-':return 1;
    }
}

void infixtoprefix(STACK *s,char infix[SIZE])
{
    int i,j=0;
    char *str1;
    char prefix[SIZE],temp,symbol;
    for(i=strlen(infix)-1;i>=0;i--)
    {
        symbol=infix[i];
        if(isalnum(symbol))
            prefix[j++]=symbol;
        else
        {
            switch(symbol)
            {
                case ')':push(s,symbol);
                         break;
                case '(':temp=pop(s);
```

```
                        while(temp!=')')
                        {
                            prefix[j++]=temp;
                            temp=pop(s);
                        }
                        break;
                case '+':
                case '-':
                case '*':
                case '/':
                case '^': if (s->top ==-1 || s->data[s->top]==')')
                                push(s,symbol);
                          else
                          {
                              while(preced(s->data[s->top])>
preced(symbol) && s->top!=-1 &&s->data[s->top]!=')')
                                        prefix[j++]=pop(s);
                              push(s,symbol);
                          }
                          break;
                default :printf("\n Invalid!!!!!");
                         exit(0);

            }
        }
    }
    while(s->top!=-1)
        prefix[j++]=pop(s);
    prefix[j]='\0';
    str1=strrev(prefix);
    printf("\n The prefix expression is %s\n",str1);
}
int main()
{
    STACK s;
    s.top=-1;
    char infix[SIZE];
    printf("\n Read Infix expression\n");
    scanf("%s",infix);
    infixtoprefix(&s,infix);
    return 0;
}
```

2. Evaluation of Postfix

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>
#define SIZE 20
struct stack
{
    int top;
    float data[SIZE];
};
typedef struct stack STACK;
void push(STACK *s,float item)
{
    s->data[++(s->top)]=item;
}

float pop(STACK *s)
{
    return s->data[(s->top)--];
}

float operate(float op1,float op2,char symbol)
{
    switch(symbol)
    {
        case '+':return op1+op2;
        case '-':return op1-op2;
        case '*':return op1*op2;
        case '/':return op1/op2;
        case '^':return pow(op1,op2);
    }
}

float eval(STACK *s,char postfix[SIZE])
{
    int i;
    char symbol;
    float res,op1,op2;
    for(i=0;postfix[i]!='\0';i++)
    {
        symbol=postfix[i];
        if(isdigit(symbol))
           push(s,symbol-'0');
        else
        {
            op2=pop(s);
            op1=pop(s);
            res=operate(op1,op2,symbol);
            push(s,res);
        }
    }
    return pop(s);
```

```
}

int main()
{
    char postfix[SIZE];
    STACK s;
    float ans;
    s.top=-1;
    printf("\n Read postfix expr\n");
    scanf("%s",postfix);
    ans=eval(&s,postfix);
    printf("\n The final answer is %f\n",ans);
    return 0;
}
```

3. Circular Queue of Integers

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
struct queue
{
    int front,rear;
    int data[SIZE];
};
typedef struct queue QUEUE;

void enqueue(QUEUE *q,int item)
{
    if(q->front==(q->rear+1) % SIZE )
        printf("\n Queue full");
    else
    {
        q->rear=(q->rear+1)%SIZE;
        q->data[q->rear]=item;
        if(q->front==-1)
            q->front=0;
    }
}

int dequeue(QUEUE *q)
{
    int del;
    if(q->front==-1)
    {
        printf("\n Queue empty");
        return -1;
    }
    else
    {
        del=q->data[q->front];
        if(q->front==q->rear)
        {
            q->front=-1;
            q->rear=-1;
        }
        else
            q->front=(q->front+1)% SIZE;
        return del;
    }
}

void display(QUEUE q)
{
    int i;
    if(q.front==-1)
        printf("\n Queue Empty");
    else
    {
```

```
        printf("\n Queue content are\n");
        for(i=q.front;i!=q.rear;i=(i+1)%SIZE)
            printf("%d\t",q.data[i]);
        printf("%d\t",q.data[i]);
    }

}

int main()
{
    int item,ch,del;
    QUEUE q;
    q.front=-1;
    q.rear=-1;
    for(;;)
    {
    printf("\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit");
    printf("\nRead Choice :");
    scanf("%d",&ch);
    switch(ch)
    {
        case 1:printf("\n Read element to be inserted :");
               scanf("%d",&item);
               enqueue(&q,item);
               break;
        case 2:del=dequeue(&q);
                if(del!=-1)
                    printf("\n Element deleted is %d\n",del);
                break;
        case 3:display(q);
                break;
        default:exit(0);
    }
    }
    return 0;

}
```

4. Stack using Singly Linked List

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
int count;
struct node
{
    int data;
    struct node *addr;
};
typedef struct node *NODE;
NODE push(NODE start,int item)
{
    NODE temp;
    if(count>=SIZE)
    {
       printf("\n Stack Overflow");
       return start;
    }
    else
    {
    temp=(NODE)malloc(sizeof(struct node));
    count=count+1;
    temp->data=item;
    temp->addr=NULL;
    if(start==NULL)
        return temp;
    temp->addr=start;
    return temp;
    }
}
NODE pop(NODE start)
{
    NODE temp;
    if(start==NULL)
    {
        printf("\n stack Underflow");
        return start;
    }
    else
    {
        temp=start;
        start=start->addr;
        printf("\n Element popped is %d\n",temp->data);
        count=count-1;
        return start;
    }
}

void display(NODE start)
{
    NODE temp;
    if(start==NULL)
        printf("\n Stack is empty");
```

```
    else
    {
        printf("\n Stack Content are\n");
        temp=start;
        while(temp!=NULL)
        {
            printf("%d\n",temp->data);
            temp=temp->addr;
        }
    }
}
int main()
{
    NODE start=NULL;
    int item,ch;
    for(;;)
    {
        printf("\n1.Push\n2.Pop\n3.Display\n4.Exit");
        printf("\n Read choice :");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:printf("\n Read data to be pushed:");
                   scanf("%d",&item);
                   start=push(start,item);
                   break;
            case 2:start=pop(start);
                    break;
            case 3:display(start);
                    break;
            default:exit(0);
        }
    }
    return 0;
}
```

5. Sparse Matrix using Doubly Linked List

```
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int row,col,data;
    struct node *next;
    struct node *prev;
};
typedef struct node *NODE;

NODE insertend(NODE start,int row,int col,int item)
{
    NODE temp,cur;
    temp=(NODE)malloc(sizeof(struct node));
    temp->row=row;
    temp->col=col;
    temp->data=item;
    temp->next=NULL;
    temp->prev=NULL;
    if(start == NULL)
        return temp;
    cur=start;
    while(cur->next!=NULL)
        cur = cur->next;
    cur->next=temp;
    temp->prev=cur;
    return start;
}

void display(NODE start)
{
    NODE temp;
    if(start==NULL)
        printf("\n list is empty");
    else
    {
        printf("\nROW\tCOL\tDATA\n");
        temp=start;
        while(temp!=NULL)
        {
            printf("%d\t%d\t%d\n",temp->row,temp->col,temp->data);
            temp=temp->next;
        }
    }
}

void displaymatrix(NODE start,int m,int n)
{
    NODE temp;
    int i,j;
    temp=start;
    printf("\n The Sparse matrix is\n");
```

```
    for(i=1;i<=m;i++)
        {
            for(j=1;j<=n;j++)
            {
                if(temp!=NULL && temp->row == i && temp->col == j)
                {
                    printf("%d\t",temp->data);
                    temp=temp->next;
                }
                else
                    printf("0\t");
       }
     printf("\n");
    }

}


int main()
{
    NODE start = NULL;
    int i,j,m,n,item;
    printf("\n Read the order of the matrix\n");
    scanf("%d%d",&m,&n);
    printf("\n Read the matrix\n");
    for(i=1;i<=m;i++)
    {
        for(j=1;j<=n;j++)
        {
            scanf("%d",&item);
            if(item!=0)
                start=insertend(start,i,j,item);
         }
    }
    display(start);
    displaymatrix(start,m,n);
    return 0;
}
```

6. Addition of 2 Long integers using Header node

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *addr;
};
typedef struct node *NODE;

NODE insertend(NODE head,int item)
{
    NODE temp,cur;
    temp=(NODE)malloc(sizeof(struct node));
    temp->data=item;
    temp->addr=NULL;
    if(head->addr == NULL)
    {
        head->addr=temp;
        return head;
    }
    else
    {
        cur=head->addr;
        while(cur->addr!=NULL)
            cur=cur->addr;
        cur->addr=temp;
        return head;
    }
}

NODE insertbegin(NODE head,int item)
{
    NODE temp,cur;
    temp=(NODE)malloc(sizeof(struct node));
    temp->data=item;
    temp->addr=NULL;
    if(head->addr == NULL)
    {
        head->addr=temp;
        return head;
    }
    else
    {
        temp->addr=head->addr;
        head->addr=temp;
        return head;
    }
}
void display(NODE head)
{
    NODE temp;
```

```
    if(head->addr == NULL)
        printf("\n List is Empty");
    else
    {
        temp=head->addr;
        while(temp!=NULL)
        {
            printf("%d",temp->data);
            temp=temp->addr;
        }
    }
}

void appendzero(NODE head1,NODE head2)
{
    int ct1=0,ct2=0,i;
    NODE cur;
    cur=head1->addr;
    while(cur!=NULL)
    {
        ct1=ct1+1;
        cur=cur->addr;
    }
    cur=head2->addr;
    while(cur!=NULL)
    {
        ct2=ct2+1;
        cur=cur->addr;
    }
    if(ct1>ct2)
    {
        for(i=0;i<ct1-ct2;i++)
            head2=insertbegin(head2,0);
    }
    else
    {
        for(i=0;i<ct2-ct1;i++)
            head1=insertbegin(head1,0);
    }
}

NODE reverse(NODE head)
{
    NODE next,prev,cur;
    cur=head->addr;
    prev=NULL;
    while(cur!=NULL)
    {
        next=cur->addr;
        cur->addr=prev;
        prev=cur;
        cur=next;
    }
    head->addr=prev;
    return head;
}
```

```
void add(NODE head1,NODE head2)
{
    NODE t1,t2,head;
    int sum,carry=0;
    head=(NODE)malloc(sizeof(struct node));
    head->addr=NULL;
    head1=reverse(head1);
    head2=reverse(head2);
    t1=head1->addr;
    t2=head2->addr;
    while(t1!=NULL)
    {
        sum=t1->data+t2->data+carry;
        carry=sum/10;
        sum=sum%10;
        head=insertbegin(head,sum);
        t1=t1->addr;
        t2=t2->addr;
    }
    if(carry!=0)
        head=insertbegin(head,carry);
    printf("\n The added number is\n");
    display(head);
}
int main()
{
    NODE head1,head2;
    char first[20],second[20];
    int i,j;
    printf("\n Read first number :");
    scanf("%s",first);
    head1=(NODE)malloc(sizeof(struct node));
    head1->addr=NULL;
    for(i=0;first[i]!='\0';i++)
        head1=insertend(head1,first[i]-'0');
    printf("\n First Number is\n");
    display(head1);
    printf("\n Read second number :");
    scanf("%s",second);
    head2=(NODE)malloc(sizeof(struct node));
    head2->addr=NULL;
    for(i=0;second[i]!='\0';i++)
        head2=insertend(head2,second[i]-'0');
    printf("\n Second Number is\n");
    display(head2);
    appendzero(head1,head2);
    printf("\n First Number is\n");
    display(head1);
    printf("\n Second Number is\n");
    display(head2);
    add(head1,head2);
    return 0;
}
```