

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



DATA STRUCTURE LAB RECORD

Submitted by

HEMANG SINGH (1BM19CS061)

Under the Guidance of

Prof. SHEETAL VA
Assistant Professor, BMSCE

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

Sep-2020 to Jan-2021

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering



CERTIFICATE

This is to certify that the LAB RECORD carried out by **HEMANG SINGH (IBM19CS061)** who is the bonafide students of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visveswaraiiah Technological University, Belgaum during the year 2020-2021. The lab report has been approved as it satisfies the academic requirements in respect of **DATA STRUCTURE LAB RECORD (19CS3PCDST)** work prescribed for the said degree.

Signature of the Guide
Prof. Prof. Sheelal VA
Assistant Professor
BMSCE, Bengaluru

Signature of the HOD
Dr. Umadevi V
Associate Prof.& Head, Dept. of CSE
BMSCE, Bengaluru

External Viva

Name of the Examiner

Signature with date

1. _____

2. _____

LAB-1

Write a program to simulate the working of stack using an array with the following : (a) Push (b) Pop (c) Display. The program should print appropriate messages for stack overflow, stack underflow.

```
#include <stdio.h>

#include<conio.h>

#define STACK_SIZE 3

int top=-1; int s[10];

int item; void push(){
    if(top==STACK_SIZE-1){
        printf("Stack overflow\n");
        return;
    }
    top=top+1;
    s[top]=item;
}

int pop(){
    if(top== -1){
        printf("stack is empty\n");
    }
    else
        return s[top--];
}
```

```

void display(){
    int i;

    if(top== -1){
        printf("stack is empty\n");
        return;
    }

    printf("items in the stack are: ");
    for(i=top; i>=0; i--){
        printf("%d\t", s[i]);
    }
}

main(){
    int choice;
    int item_deleted;

    for(;;){
        printf("\n1:push 2:pop 3:display 4:exit\n");
        printf("Enter your choice: ");
        scanf("%d",&choice);
        switch(choice){
            case 1: printf("Enter the item to be
inserted:");
                scanf("%d",&item);
                push();
                break;
            case 2: item_deleted=pop();

```

[illegible]

LAB-2

1- WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide).

```
#include<stdio.h>
#include<string.h>
int F(char symbol)
{
    switch(symbol)
    {
        case '+':
        case '-': return 2;
        case '*':
        case '/': return 4;
        case '^':
        case '$': return 5;
        case '(': return 0;
        case '#': return -1;
        default: return 8;
    }
}
int G(char symbol)
{
    switch(symbol)
    {
        case '+':
        case '-': return 1;
        case '*':
        case '/': return 3;
        case '^':
        case '$': return 6;
        case '(': return 9;
        case ')': return 0;
        default: return 7;
    }
}
void infix_postfix(char infix[],char postfix[])
{
    int top,i,j;
```

```

char s[30],symbol;
top=-1;
s[++top]='#';
j=0;
for(i=0;i<strlen(infix);i++)
{
    symbol = infix[i];
    while(F(s[top])>G(symbol))
    {
        postfix[j]=s[top--];
        j++;
    }
    if(F(s[top])!=G(symbol))
        s[++top]=symbol;
    else
        top--;
}
while(s[top]!='#')
{
    postfix[j++]=s[top--];
}
postfix[j]='\0';
}

int main()
{
    char infix[20];
    char postfix[20];
    printf("Enter the valid infix expression\n");
    scanf("%s",infix);
    infix_postfix(infix,postfix);
    printf("The postfix expression is:\n");
    printf("%s\n",postfix);
}

```

```
"D:\coding files\CodeBlocks\c-programming\DS lab\nikita.exe"
Enter the valid infix expression
x^y^z-M+N+p/Q
The postfix expression is:
xyz^M-N+pQ/+
Process returned 0 (0x0)   execution time : 27.318 s
Press any key to continue.
```

```
"D:\coding files\CodeBlocks\c-programming\DS lab\nikita.exe"
Enter the valid infix expression
((a+b)*c-(d-e))^(f+g)
The postfix expression is:
ab+c*de--fg+^
Process returned 0 (0x0)   execution time : 28.070 s
Press any key to continue.
```

```
"D:\coding files\CodeBlocks\c-programming\DS lab\nikita.exe"
Enter the valid infix expression
(A+(B-C)*D)
The postfix expression is:
ABC-D*+
Process returned 0 (0x0)   execution time : 23.282 s
Press any key to continue.
```


LAB - 3

/*Write a Program to simulate the working of queue of integers using an array. Provide the following operations.

a) Insert Rear

b) Delete Front

c) Display the contents of queue

The program should print the appropriate messages for a queue empty and queue full condition.

*/

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#define QUE_SIZE 3
int item,front=0,rear=-1,q[10];
void insertrear(){
    if(rear == QUE_SIZE - 1 ){
        printf("QUEUE OVERFLOW\n");
        return;
    }
    rear = rear+1;
    q[rear] = item;
}
int deletefront(){
    if(front > rear){
        front =0;
        rear =-1;
        return -1;
    }
    return q[front ++];
}
void displayQ(){
    int i;
    if(front>rear){
        printf("QUEUE IS EMPTY\n");
        return;
    }
    printf("***Contents of Queue** \n");
    for(i=front;i<=rear;i++){
        printf(" %d\n",q[i]);
    }
}
void main(){
```

```

int choice;
for(;;){
    printf("\n1.Insert Rear \n2.Delete front \n3.Display \n4.Exit\n");
    printf("Enter the choice\n");
    scanf("%d",&choice);
    switch(choice){
        case 1: printf("Enter the items to be inserted\n");
                scanf("%d",&item);
                insertrear();
                break;
        case 2: item = deletefront();
                if(item == -1)
                    printf("QUEUE IS EMPTY\n");
                else
                    printf("Item Deleted = %d\n",item);
                break;
        case 3: displayQ();
                break;
        default: exit(0);
    }
}
}
}

```

Command Prompt

D:\coding files\CodeBlocks\c-programming\DS lab>lab3

1.Insert Rear

2.Delete front

3.Display

4.Exit

Enter the choice

1

Enter the items to be inserted

10

1.Insert Rear

2.Delete front

3.Display

4.Exit

Enter the choice

1

Enter the items to be inserted

20

1.Insert Rear

2.Delete front

3.Display

4.Exit

Enter the choice

1

Enter the items to be inserted

30

1.Insert Rear

2.Delete front

3.Display

4.Exit

Enter the choice

3

Contents of Queue

10

20

30

1.Insert Rear

2.Delete front

3.Display

Command Prompt

1.Insert Rear

2.Delete front

3.Display

4.Exit

Enter the choice

1

Enter the items to be inserted

40

QUEUE OVERFLOW

1.Insert Rear

2.Delete front

3.Display

4.Exit

Enter the choice

3

Contents of Queue

10

20

30

1.Insert Rear

2.Delete front

3.Display

4.Exit

Enter the choice

2

Item Deleted = 10

1.Insert Rear

2.Delete front

3.Display

4.Exit

Enter the choice

2

Item Deleted = 20

1.Insert Rear

2.Delete front

3.Display

4.Exit

Enter the choice

Command Prompt

Item Deleted = 20

1.Insert Rear
2.Delete front
3.Display
4.Exit
Enter the choice

2

Item Deleted = 30

1.Insert Rear
2.Delete front
3.Display
4.Exit
Enter the choice

2

QUEUE IS EMPTY

1.Insert Rear
2.Delete front
3.Display
4.Exit
Enter the choice

3

QUEUE IS EMPTY

1.Insert Rear
2.Delete front
3.Display
4.Exit
Enter the choice

4

D:\coding files\CodeBlocks\c-programming\DS lab>

LAB-4

1.Double Ended Queue

```
#include<stdio.h>
#include<stdlib.h>
#define qsize 5
int f=0,r=-1,ch;
int item,q[10];

int isfull()
{
    return(r==qsize-1)?1:0;
}
int isempty()
{
    return(f>r)?1:0;
}
void insert_rear()
{
    if(isfull())
    {
        printf("queue overflow\n");
        return;
    }
    r=r+1;
    q[r]=item;
}
void delete_front()
{
    if(isempty())
    {
        printf("queue empty\n");
        return;
    }
    printf("item deleted is %d\n",q[(f)++]);
    if(f>r)
    {
        f=0;
    }
}
```

```

        r=-1;
    }
}
void insert_front()
{
    if(f!=0)
    {
        f=f-1;
        q[f]=item;
        return;
    }
    else if((f==0)&&(r== -1))
    {
        q[++(r)]=item;
        return;
    }
    else
        printf("insertion not possible\n");
}
void delete_rear()
{
    if(isempty())
    {
        printf("queue is empty\n");
        return;
    }
    printf("item deleted is %d\n",q[(r)--]);
    if(f>r)
    {
        f=0;
        r=-1;
    }
}
void display()
{
    int i;
    if(isempty())
    {
        printf("queue empty\n");
        return;
    }
    for(i=f;i<=r;i++)
        printf("%d\n",q[i]);
}

```

```
void main()
{
    for(;;)
    {
        printf("1.insert_rear\n2.insert_front\n3.delete_rear\n4.delete_front\n5.display\n6.exit\n");
        printf("enter choice\n");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:printf("enter the item\n");
                    scanf("%d",&item);
                    insert_rear();
                    break;
            case 2:printf("enter the item\n");
                    scanf("%d",&item);
                    insert_front();
                    break;
            case 3:delete_rear();
                    break;
            case 4:delete_front();
                    break;
            case 5:display();
                    break;
            default:exit(0);
        }
    }
}
```


Command Prompt - queue

```
D:\coding files\DS lab>gcc -o queue lab4-1.c
```

```
D:\coding files\DS lab>queue
```

```
1.insert_rear
```

```
2.insert_front
```

```
3.delete_rear
```

```
4.delete_front
```

```
5.display
```

```
6.exit
```

```
enter choice
```

```
1
```

```
enter the item
```

```
23
```

```
1.insert_rear
```

```
2.insert_front
```

```
3.delete_rear
```

```
4.delete_front
```

```
5.display
```

```
6.exit
```

```
enter choice
```

```
1
```

```
enter the item
```

```
25
```

```
1.insert_rear
```

```
2.insert_front
```

```
3.delete_rear
```

```
4.delete_front
```

```
5.display
```

```
6.exit
```

```
enter choice
```

```
1
```

```
enter the item
```

```
28
```

```
1.insert_rear
```

```
2.insert_front
```

```
3.delete_rear
```

```
4.delete_front
```

```
5.display
```

```
6.exit
```

```
enter choice
```

Command Prompt - queue

```
1
enter the item
34
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
1
enter the item
54
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
1
enter the item
67
queue overflow
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
5
23
25
28
34
54
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
```

CA. Command Prompt - queue

```
6.exit
enter choice
3
item deleted is 54
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
5
23
25
28
34
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
4
item deleted is 23
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
5
25
28
34
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
```

Git Command Prompt - queue

```
enter choice
2
enter the item
77
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
5
77
25
28
34
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
3
item deleted is 34
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
3
item deleted is 28
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
3
item deleted is 25
```

C:\ Command Prompt - queue

```
3
item deleted is 25
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
4
item deleted is 77
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
4
queue empty
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
5
queue empty
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
```

2.Input and Output restricted queue

```
# include<stdio.h>
# define Size 5
int deque_arr[Size];
int front = -1, rear = -1;

void insert_rear()
{
    int added_item;
    if((front == 0 && rear == Size-1) || (front == rear+1))
    { printf("***Queue Overflow***\n");
      return;
    }
    if (front == -1)
    {
        front = 0;
        rear = 0;
    }
    else
    if(rear == Size-1)
        rear = 0;
    else
        rear = rear+1;

    printf("Enter the element for adding in queue : ");
    scanf("%d",&added_item);
    deque_arr[rear] = added_item ;
}
```

```
void insert_front()
{ int added_item;
  if((front == 0 && rear == Size-1) || (front == rear+1))
  { printf("Queue Overflow \n");
    return;
  }
  if (front == -1)
  { front = 0;
    rear = 0;
  }
```

```

    }
else
if(front== 0)
    front=Size-1;
else
    front=front-1;
printf("Enter the element for adding in queue : ");
scanf("%d", &added_item);
deque_arr[front] = added_item ;
}

```

```

void delete_front()
{
    if (front == -1)
    {
        printf("Queue Underflow\n");
        return ;
    }
    printf("Element deleted from queue is : %d\n",deque_arr[front]);
    if(front == rear)
    {
        front = -1;
        rear=-1;
    }
    else
        if(front == Size-1)
            front = 0;
        else
            front = front+1;
}

```

```

void delete_rear()
{
    if (front == -1)
    {
        printf("Queue Underflow\n");
        return ;
    }
    printf("Element deleted from queue is : %d\n",deque_arr[rear]);
    if(front == rear)
    {
        front = -1;
        rear=-1;
    }
}

```

```

}
else
    if(rear == 0)
        rear=Size-1;
    else
        rear=rear-1;
    }

```

```

void display_queue()
{
    int front_pos = front, rear_pos = rear;
    if(front == -1)
    { printf("Queue is empty\n");
      return;
    }
    printf("Queue elements :\n");
    if( front_pos <= rear_pos )
    {
        while(front_pos <= rear_pos)
        {
            printf("%d \n",deque_arr[front_pos]);
            front_pos++;
        }
    }
    else
    {
        while(front_pos <= Size-1)
        { printf("%d \n",deque_arr[front_pos]);
          front_pos++;
        }
        front_pos = 0;
        while(front_pos <= rear_pos)
        {
            printf("%d \n",deque_arr[front_pos]);
            front_pos++;
        }
    }
    printf("\n");
}

```

/*Input Queue*/


```

void input_que()
{ int choice;
  do
  { printf("1.Insert at rear\n2.Delete from front\n3.Delete from rear\n4.Display\n5.Quit\n");
    printf("Enter your choice :");
    scanf("%d",&choice);
    switch(choice)
    { case 1:
      insert_rear();
      break;
      case 2:
      delete_front();
      break;
      case 3:
      delete_rear();
      break;
      case 4:
      display_queue();
      break;
      case 5:
      break;
      default:
      printf("Wrong choice\n");
    }
  }
  while(choice!=5);
}

```

```

/*Output Queue*/
void output_que()
{ int choice;
  do
  { printf("1.Insert at rear\n2.Insert at front\n3.Delete from front\n4.Display\n5.Quit\n");
    printf("Enter your choice : ");
    scanf("%d",&choice);
    switch(choice)
    {
      case 1:
      insert_rear();
      break;
      case 2:
      insert_front();
      break;

```

```

        case 3:
            delete_front();
            break;
        case 4:
            display_queue();
            break;
        case 5:
            break;
        default:
            printf("Wrong choice\n");
    }
}while(choice!=5);
}

```

```

main()
{ int choice;
  printf("1.Input restricted dequeue\n2.Output restricted dequeue\n");
  printf("Enter your choice : ");
  scanf("%d",&choice);
  switch(choice)
  {
    case 1 :
        input_que();
        break;
    case 2:
        output_que();
        break;
    default:
        printf("Wrong choice\n");
  }
}

```

```
D:\coding files\DS lab>gcc -o restricted lab4-2.c
```

```
D:\coding files\DS lab>restricted
```

```
1.Input restricted dequeue
```

```
2.Output restricted dequeue
```

```
Enter your choice : 1
```

```
1.Insert at rear
```

```
2.Delete from front
```

```
3.Delete from rear
```

```
4.Display
```

```
5.Quit
```

```
Enter your choice :1
```

```
Enter the element for adding in queue : 23
```

```
1.Insert at rear
```

```
2.Delete from front
```

```
3.Delete from rear
```

```
4.Display
```

```
5.Quit
```

```
Enter your choice :1
```

```
Enter the element for adding in queue : 34
```

```
1.Insert at rear
```

```
2.Delete from front
```

```
3.Delete from rear
```

```
4.Display
```

```
5.Quit
```

```
Enter your choice :1
```

```
Enter the element for adding in queue : 45
```

```
1.Insert at rear
```

```
2.Delete from front
```

```
3.Delete from rear
```

```
4.Display
```

```
5.Quit
```

```
Enter your choice :1
```

```
Enter the element for adding in queue : 67
```

```
1.Insert at rear
```

```
2.Delete from front
```

```
3.Delete from rear
```

```
4.Display
```

```
5.Quit
```

```
Enter your choice :2
```

```
Element deleted from queue is : 23
```

```
1.Insert at rear
```

C:\ Command Prompt - 1

Enter the element for adding in queue : 67

- 1.Insert at rear
- 2.Delete from front
- 3.Delete from rear
- 4.Display
- 5.Quit

Enter your choice :2

Element deleted from queue is : 23

- 1.Insert at rear
- 2.Delete from front
- 3.Delete from rear
- 4.Display
- 5.Quit

Enter your choice :4

Queue elements :

34
45
67

- 1.Insert at rear
- 2.Delete from front
- 3.Delete from rear
- 4.Display
- 5.Quit

Enter your choice :3

Element deleted from queue is : 67

- 1.Insert at rear
- 2.Delete from front
- 3.Delete from rear
- 4.Display
- 5.Quit

Enter your choice :5

D:\coding files\DS lab>1

Enter the total no of students

Command Prompt - 3

```
D:\coding files\DS lab>
D:\coding files\DS lab>restricted
1.Input restricted dequeue
2.Output restricted dequeue
Enter your choice : 2
1.Insert at rear
2.Insert at front
3.Delete from front
4.Display
5.Quit
Enter your choice : 1
Enter the element for adding in queue : 23
1.Insert at rear
2.Insert at front
3.Delete from front
4.Display
5.Quit
Enter your choice : 1
Enter the element for adding in queue : 45
1.Insert at rear
2.Insert at front
3.Delete from front
4.Display
5.Quit
Enter your choice : 2
Enter the element for adding in queue : 34
1.Insert at rear
2.Insert at front
3.Delete from front
4.Display
5.Quit
Enter your choice : 2
Enter the element for adding in queue : 67
1.Insert at rear
2.Insert at front
3.Delete from front
4.Display
5.Quit
Enter your choice : 2
Enter the element for adding in queue : 78
1.Insert at rear
2.Insert at front
3.Delete from front
```

Command Prompt - 3

```
1.Insert at rear
2.Insert at front
3.Delete from front
4.Display
5.Quit
Enter your choice : 1
***Queue Overflow***
1.Insert at rear
2.Insert at front
3.Delete from front
4.Display
5.Quit
Enter your choice : 4
Queue elements :
78
67
34
23
45

1.Insert at rear
2.Insert at front
3.Delete from front
4.Display
5.Quit
Enter your choice : 3
Element deleted from queue is : 78
1.Insert at rear
2.Insert at front
3.Delete from front
4.Display
5.Quit
Enter your choice : 3
Element deleted from queue is : 67
1.Insert at rear
2.Insert at front
3.Delete from front
4.Display
5.Quit
Enter your choice : 3
Element deleted from queue is : 34
1.Insert at rear
2.Insert at front
```


Command Prompt - 3

```
4.Display
5.Quit
Enter your choice : 3
Element deleted from queue is : 67
1.Insert at rear
2.Insert at front
3.Delete from front
4.Display
5.Quit
Enter your choice : 3
Element deleted from queue is : 34
1.Insert at rear
2.Insert at front
3.Delete from front
4.Display
5.Quit
Enter your choice : 3
Element deleted from queue is : 23
1.Insert at rear
2.Insert at front
3.Delete from front
4.Display
5.Quit
Enter your choice : 5

D:\coding files\DS lab>3
Enter number of employees:
3
```

LAB - 5 & 6

/*C-Program to implement LINKED LIST with functions insertion & deletion with specified position

LAB 5 & 6

*/

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int info;
    struct node *link;
};
typedef struct node *NODE;
NODE getnode(){
    NODE x;
    x=(NODE)malloc(sizeof(struct node));
    if(x==NULL){
        printf("mem full\n");
        exit(0);
    }
    return x;
}
void freenode(NODE x){
    free(x);
}
NODE insert_rear(NODE first,int item){
    NODE temp,cur;
    temp=getnode();
    temp->info=item;
    temp->link=NULL;
    if(first==NULL)
        return temp;
    cur=first;
    while(cur->link!=NULL)
        cur=cur->link;
    cur->link=temp;
    return first;
}
NODE delete_rear(NODE first){
    NODE cur,prev;
    if(first==NULL){
```



```

    printf("list is empty cannot delete\n");
    return first;
}
if(first->link==NULL){
    printf("item deleted is %d\n",first->info);
    free(first);
    return NULL;
}
prev=NULL;
cur=first;
while(cur->link!=NULL){
    prev=cur;
    cur=cur->link;
}
printf("item deleted at rear-end is %d",cur->info);
free(cur);
prev->link=NULL;
return first;
}
NODE insert_pos(int item,int pos,NODE first){
    NODE temp,cur,prev;
    int count;
    temp=getnode();
    temp->info=item;
    temp->link=NULL;
    if(first==NULL&&pos==1){
        return temp;
    }
    if(first==NULL){
        printf("invalid position\n");
        return first;
    }
    if(pos==1){
        temp->link=first;
        first=temp;
        return temp;
    }
    count=1;
    prev=NULL;
    cur=first;
    while(cur!=NULL&&count!=pos){
        prev=cur;
        cur=cur->link;
        count++;
    }

```

```

}
if(
    count==pos){
    prev->link=temp;
    temp->link=cur;
    return first;
}
printf("invalid position\n");
return first;
}
NODE delete_pos(int pos,NODE first){
    NODE cur;
    NODE prev;
    int count,flag=0;
    if(first==NULL || pos<0){
        printf("invalid position\n");
        return NULL;
    }
    if(pos==1){
        cur=first;
        first=first->link;
        freenode(cur);
        return first;
    }
    prev=NULL;
    cur=first;
    count=1;
    while(cur!=NULL){
        if(count==pos){
            flag=1;
            break;
        }
        count++;
        prev=cur;
        cur=cur->link;
    }
    if(flag==0){
        printf("invalid position\n");
        return first;
    }
    printf("item deleted at given position is %d\n",cur->info);
    prev->link=cur->link;
    freenode(cur);
    return first;
}

```

```

}
void display(NODE first){
    NODE temp;
    if(first==NULL)
        printf("list empty cannot display items\n");
    for(temp=first;temp!=NULL;temp=temp->link){
        printf("%d\n",temp->info);
    }
}
void main()
{
    int item,choice,key,pos;
    int count=0;
    NODE first=NULL;
    for(;;){

        printf("\n1:Insert_rear\n2:Delete_rear\n3:insert_info_position\n4:Delete_info_position\n5:Display
        _list\n6:Exit\n");
        printf("enter the choice\n");
        scanf("%d",&choice);
        switch(choice){
            case 1:printf("enter the item at rear-end\n");
                scanf("%d",&item);
                first=insert_rear(first,item);
                break;
            case 2:first=delete_rear(first);
                break;
            case 3:printf("enter the item to be inserted at given position\n");
                scanf("%d",&item);
                printf("enter the position\n");
                scanf("%d",&pos);
                first=insert_pos(item,pos,first);
                break;
            case 4:printf("enter the position\n");
                scanf("%d",&pos);
                first=delete_pos(pos,first);
                break;
            case 5:display(first);
                break;
            default:exit(0);
                break;
        }
    }
}

```

Command Prompt - linked-list1

D:\coding files\DS lab>linked-list1

```
1:Insert_rear
2:Delete_rear
3:insert_info_position
4:Delete_info_position
5:Display_list
6:Exit
enter the choice
1
enter the item at rear-end
12
```

```
1:Insert_rear
2:Delete_rear
3:insert_info_position
4:Delete_info_position
5:Display_list
6:Exit
enter the choice
1
enter the item at rear-end
13
```

```
1:Insert_rear
2:Delete_rear
3:insert_info_position
4:Delete_info_position
5:Display_list
6:Exit
enter the choice
1
enter the item at rear-end
45
```

```
1:Insert_rear
2:Delete_rear
3:insert_info_position
4:Delete_info_position
5:Display_list
6:Exit
enter the choice
1
```

Command Prompt - linked-list1

6:Exit

enter the choice

1

enter the item at rear-end

234

1:Insert_rear

2:Delete_rear

3:insert_info_position

4:Delete_info_position

5:Display_list

6:Exit

enter the choice

1

enter the item at rear-end

22

1:Insert_rear

2:Delete_rear

3:insert_info_position

4:Delete_info_position

5:Display_list

6:Exit

enter the choice

5

12

13

45

234

22

1:Insert_rear

2:Delete_rear

3:insert_info_position

4:Delete_info_position

5:Display_list

6:Exit

enter the choice

2

item deleted at rear-end is 22

1:Insert_rear

2:Delete_rear

3:insert_info_position

C:\ Command Prompt - linked-list1

2:Delete_rear

3:insert_info_position

4:Delete_info_position

5:Display_list

6:Exit

enter the choice

5

12

13

45

234

1:Insert_rear

2:Delete_rear

3:insert_info_position

4:Delete_info_position

5:Display_list

6:Exit

enter the choice

3

enter the item to be inserted at given position

99

enter the position

4

1:Insert_rear

2:Delete_rear

3:insert_info_position

4:Delete_info_position

5:Display_list

6:Exit

enter the choice

5

12

13

45

99

234

1:Insert_rear

2:Delete_rear

3:insert_info_position

4:Delete_info_position

Command Prompt - linked-list1

5
12
13
45
99
234

1:Insert_rear
2:Delete_rear
3:insert_info_position
4:Delete_info_position
5:Display_list
6:Exit

enter the choice

4

enter the position

2

item deleted at given position is 13

1:Insert_rear
2:Delete_rear
3:insert_info_position
4:Delete_info_position
5:Display_list
6:Exit

enter the choice

5

12

45

99

234

1:Insert_rear
2:Delete_rear
3:insert_info_position
4:Delete_info_position
5:Display_list
6:Exit

enter the choice

Lab-7,8

/*

WAP Implement Single Link List with following operations

- a) Sort the linked list.
- b) Reverse the linked list.
- c) Concatenation of two linked lists

WAP to implement Stack & Queues using Linked Representation

*/

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

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

```
struct node
```

```
{
```

```
    int data;
```

```
    struct node* next;
```

```
};
```

```
struct node *rear=NULL, *front =NULL, *top=NULL;
```

```
struct node* getnode(int item)
```

```
{
```

```
    struct node* newn = (struct node*)malloc(sizeof(struct node));
```

```
    newn->data = item;
```

```
    newn->next = NULL;
```

```
    return newn;
```

```
}
```

```
void display(struct node* head)
```

```
{
```

```
    if(head == NULL)
```

```
    {
```

```

        printf("List is empty.\n");
        return;
    }
    struct node* ptr = head;
    while(ptr)
    {
        printf("%d->", ptr->data);
        ptr = ptr->next;
    }
    printf("\b \b\b \n");
}

```

```

struct node* insertfront(struct node* head, int item)
{
    struct node* newn = getnode(item);
    newn->next = head;
    head = newn;
    return head;
}

```

```

void swap(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}

```

```

struct node* sort (struct node* head)
{
    int sorted;
    if(head == NULL) return head;
    struct node* ptr = head;
    do

```

```

{
    ptr = head;
    sorted = 0;
    while(ptr->next)
    {
        if(ptr->data > ptr->next->data)
        {
            swap(&ptr->data, &ptr->next->data);
            sorted = 1;
        }
        ptr = ptr->next;
    }
} while(sorted == 1);
return head;
}

```

```

void reverse(struct node** head)
{
    struct node* prev = NULL;
    struct node* current = *head;
    struct node* next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head = prev;
}

```

```

struct node* concatenate(struct node* head1, struct node* head2)
{
    struct node* ptr = head1;
    while(ptr->next)
    {

```

```

        ptr = ptr->next;
    }
    ptr->next = head2;
    return head1;
}

```

```

void qinsert()
{
    struct node *newnode;
    newnode=(struct node *) malloc(sizeof(struct node));
    printf("Enter the element:\n");
    scanf("%d",&newnode->data);
    newnode->next=NULL;

    if(rear==NULL)
    {
        rear=newnode;
        front=newnode;
    }
    else
    {
        rear->next=newnode;
        rear=newnode;
    }
}

```

```

void qdel()
{
    if(front==NULL)
    {
        printf("Queue is empty\n");return;
    }
}

```

```

else
{
    printf("Deleted ele is %d",front->data);
    if(front==rear)
    {
        printf("Queue is empty\n");
        front=NULL; rear=NULL;
    }
    else
        front=front->next;
}
}

```

```

void qdisplay()
{
    struct node *temp;
    if(front ==NULL)
    {
        printf("Queue is empty");
        return;
    }
    temp=front;
    while (temp !=NULL)
    {
        printf("%d ",temp->data);
        temp=temp->next;
    }
}

```

```

}
void spush()
{
    int item;
    struct node *newnode;
    printf("Enter the element\n");
}

```

```

scanf("%d",&item);

newnode=(struct node*)malloc(sizeof(struct node));
newnode->data=item;
newnode->next=NULL;
if(top==NULL)
    top=newnode;
else
    newnode->next=top;
    top=newnode;
}
void spop()
{
    if(top==NULL)
        printf("stack is empty");
    else
    {

        printf("element removed is %d:", top->data);

        top=top->next;

    }

}

void sdisplay()
{
    struct node *temp;
    temp=top;
    if(top==NULL)
        printf("Stack is empty");
    while(temp!=NULL)
    {

```

```

    printf("%d",temp->data);
    printf("\n");
    temp=temp->next;
}

}

```

```

int main()
{
    printf("Linked list program containing sort, reverse, and concatenate
functions.\n");
    int n1, n2, n, ch, flag = 0;
    int choice;
    struct node* head1 = NULL; struct node* head2 = NULL;
    do
    {
        printf("Enter the choice\n1.Stack\n2.Queue\n3: Linked list 1\n4:
Linked list 2\n5: Exit\n");
        scanf("%d", &n1);
        switch(n1)
        {

            case 1:
            {

                do
                { printf("\n1. Push \n2. Display \n3. Pop\n");
                printf("\nEnter your choice : ");
                scanf("%d",&choice);
                switch(choice)
                {
                    case 1: spush(); break;
                    case 2: sdisplay();break;
                    case 3: spop(); break;

```

```

        ;
    }
    }while(choice!=10);

}

case 2:
    {
        do
        { printf("\nQueue implementation using linked list\n");
          printf("\n1. Create \n2. Display \n3. Delete \n4. Exit
\n");

          printf("\nEnter your choice : ");
          scanf("%d",&choice);
          switch(choice)
          { case 1: qinsert(); break;
            case 2: qdisplay();break;
            case 3: qdel(); break;

          }
        }while(choice!=10);
    }

case 3:
    {
        do
        {
            printf("3: Insert\n4: Sort\n5: Reverse\n6:
Concatenate with list 1\n7: Display list\n8: Go back to main menu\n9:
Exit\n");

```


inserted: ");

insertfront(head1, n);

concatenate(head1, head2);

```
scanf("%d", &n2);
switch(n2)
{
    case 3: {
        printf("Enter item to be

        scanf("%d", &n);
        head1 =

        break;
    }
    case 4: {
        head1 = sort(head1);
        break;
    }
    case 5: {
        reverse(&head1);
        break;
    }
    case 6: {
        head1 =

        break;
    }
    case 7: {
        display(head1);
        break;
    }
    case 8: {
        flag = 1;
        break;
    }
    case 9: {
        exit(0);
    }
}
```

```

        default: printf("Invalid input.\n");
    }
    if(flag == 1)
    {
        break;
    }
}while(1);
break;
}
case 4: {
    flag = 0;
    do
    {
        printf("3: Insert\n4: Sort\n5: Reverse\n6:
Concatenate with list 1\n7: Display list\n8: Go back to main menu\n9:
Exit\n");

        scanf("%d", &n2);
        switch(n2)
        {
            case 3: {
                printf("Enter item to be
inserted: ");

                scanf("%d", &n);
                head2 =
insertfront(head2, n);

                break;
            }
            case 4: {
                head2 = sort(head2);
                break;
            }
            case 5: {
                reverse(&head2);
                break;
            }
        }
    }
}

```

```

        case 6: {
            head2 =
concatenate(head2, head1);
            break;
        }
        case 7: {
            display(head2);
            break;
        }
        case 8: {
            flag = 1;
            break;
        }
        case 9: {
            exit(0);
        }
        default: printf("Invalid input.\n");
    }
    if(flag == 1)
    {
        flag = 0; break;
    }
}while(1);
break;
}
case 9: exit(0);
default: printf("Invalid input.\n");
}
}while(1);
return 0;
}

```

LAB-9

WAP Implement doubly link list with primitive operations

- a) Create a doubly linked list.**
- b) Insert a new node to the left of the node.**
- c) Delete the node based on a specific value**
- d) Display the contents of the list**

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int info;
    struct node *rlink;
    struct node *llink;
};
typedef struct node *NODE;
NODE getnode()
{
    NODE x;
    x=(NODE)malloc(sizeof(struct node));
    if (x==NULL)
    {
        printf("Memory full\n");
        exit(0);
    }
    return x;
}
NODE dinser_front(int item,NODE head)
{
    NODE temp,cur;
    temp=getnode();
    temp->info=item;
    temp->llink=NULL;
    temp->rlink=NULL;
    cur=head->rlink;
    head->rlink=temp;
    temp->llink=head;
    temp->rlink=cur;
    cur->llink=temp;
```

```

return head;
}
NODE dinsert_rear(int item,NODE head)
{
    NODE temp,cur;
    temp=getnode();
    temp->info=item;
    temp->llink=NULL;
    temp->rlink=NULL;
    cur=head->llink;
    head->llink=temp;
    temp->rlink=head;
    cur->rlink=temp;
    temp->llink=cur;
    return head;
}
NODE ddelete_front(NODE head)
{
    NODE cur,next;
    if (head->rlink==head)
    {
        printf("List is empty\n");
        return head;
    }
    cur=head->rlink;
    next=cur->rlink;
    head->rlink=next;
    next->llink=head;
    printf("Item deleted at the front end is:%d\n",cur->info);
    free(cur);
    return head;
}
NODE ddelete_rear(NODE head)
{
    NODE cur,prev;
    if (head->rlink==head)
    {
        printf("List is empty\n");
        return head;
    }
    cur=head->llink;
    prev=cur->llink;
    prev->rlink=head;
    head->llink=prev;
}

```

```

printf("Item deleted at the rear end is:%d\n",cur->info);
free(cur);
return head;
}
void ddisplay(NODE head)
{
NODE temp;
if (head->rlink==head)
{
printf("List is empty\n");
}
printf("The contents of the list are:\n");
temp=head->rlink;
while (temp!=head)
{
printf("%d\n",temp->info);
temp=temp->rlink;
}
}
void dsearch(int key,NODE head)
{
NODE cur;
int count;
if (head->rlink==head)
{
printf("List is empty\n");
}
cur=head->rlink;
count=1;
while (cur!=head && cur->info!=key)
{
cur=cur->rlink;
count++;
}
if (cur==head)
{
printf("Search unsuccessful\n");
}
else
{
printf("Key element found at the position %d\n",count);
}
}
NODE dinser_leftpos(int item,NODE head)

```

```

{
NODE cur,prev,temp;
if (head->rlink==head)
{
printf("List is empty\n");
return head;
}
cur=head->rlink;
while (cur!=head)
{
if (cur->info==item)
{
break;
}
cur=cur->rlink;
}
if (cur==head)
{
printf("No such item found in the list\n");
return head;
}
prev=cur->llink;
temp=getnode();
temp->llink=NULL;
temp->rlink=NULL;
printf("Enter the item to be inserted at the left of the given item:\n");
scanf("%d",&temp->info);
prev->rlink=temp;
temp->llink=prev;
temp->rlink=cur;
cur->llink=temp;
return head;
}
NODE dinsert_rightpos(int item,NODE head)
{
NODE temp,cur,next;
if (head->rlink==head)
{
printf("List is empty\n");
return head;
}
cur=head->rlink;
while (cur!=head)
{

```

```

if (cur->info==item)
{
break;
}
cur=cur->rlink;
}
if (cur==head)
{
printf("No such item found in the list\n");
return head;
}
next=cur->rlink;
temp=getnode();
temp->llink=NULL;
temp->rlink=NULL;
printf("Enter the item to be inserted at the right of the given item:\n");
scanf("%d",&temp->info);
cur->rlink=temp;
temp->llink=cur;
next->llink=temp;
temp->rlink=next;
return head;
}
NODE ddelete_duplicates(int item,NODE head)
{
NODE prev,cur,next;
int count=0;
if (head->rlink==head)
{
printf("List is empty\n");
return head;
}
cur=head->rlink;
while (cur!=head)
{
if (cur->info!=item)
{
cur=cur->rlink;
}
else
{
count++;
if (count==1)
{

```



```

cur=cur->rlink;
continue;
}
else
{
prev=cur->llink;
next=cur->rlink;
prev->rlink=next;
next->llink=prev;
free(cur);
cur=next;
}
}
}
if (count==0)
{
printf("No such item found in the list\n");
}
else
{
printf("Removed all the duplicate elements of the given item successfully\n");
}
return head;
}
int main()
{
NODE head;
int item, choice, key;
head=getnode();
head->llink=head;
head->rlink=head;
for(;;)
{
printf("\n1:dinsert front\n2:dinsert rear\n3:ddelete front\n4:ddelete
rear\n5:ddisplay\n6:dsearch\n7:dinsert lestop\n8:dinsert rightpos\n9:ddelete
duplicates\n10:exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
{
case 1: printf("Enter the item at front end:\n");
scanf("%d",&item);
head=dinsert_front(item,head);
break;

```

```
case 2: printf("Enter the item at rear end:\n");
scanf("%d",&item);
head=dinsert_rear(item,head);
break;
case 3:head=ddelete_front(head);
break;
case 4:head=ddelete_rear(head);
break;
case 5:ddisplay(head);
break;
case 6:printf("Enter the key element to be searched:\n");
scanf("%d",&key);
dsearch(key,head);
break;
case 7:printf("Enter the key element:\n");
scanf("%d",&key);
head=dinsert_leftpos(key,head);
break;
case 8:printf("Enter the key element:\n");
scanf("%d",&key);
head=dinsert_rightpos(key,head);
break;
case 9:printf("Enter the key element whose duplicates should be removed:\n");
scanf("%d",&key);
head=ddelete_duplicates(key,head);
break;
default:exit(0);
}
}
return 0;
}
```

C:\ Command Prompt

```
D:\coding files\DS lab>gcc -o lab9 lab9.c
```

```
D:\coding files\DS lab>lab9
```

```
1:dinsert front
```

```
2:dinsert rear
```

```
3:ddelete front
```

```
4:ddelete rear
```

```
5:ddisplay
```

```
6:dsearch
```

```
7:dinsert lestpos
```

```
8:dinsert rightpos
```

```
9:ddelete duplicates
```

```
10:exit
```

```
enter the choice
```

```
1
```

```
Enter the item at front end:
```

```
12
```

```
1:dinsert front
```

```
2:dinsert rear
```

```
3:ddelete front
```

```
4:ddelete rear
```

```
5:ddisplay
```

```
6:dsearch
```

```
7:dinsert lestpos
```

```
8:dinsert rightpos
```

```
9:ddelete duplicates
```

```
10:exit
```

```
enter the choice
```

```
1
```

```
Enter the item at front end:
```

```
13
```

```
1:dinsert front
```

```
2:dinsert rear
```

```
3:ddelete front
```

```
4:ddelete rear
```

```
5:ddisplay
```

```
6:dsearch
```

```
7:dinsert lestpos
```

```
8:dinsert rightpos
```

C:\. Command Prompt

```
1:dinsert front
2:dinsert rear
3:ddelete front
4:ddelete rear
5:ddisplay
6:dsearch
7:dinsert lestpos
8:dinsert rightpos
9:ddelete duplicates
10:exit
enter the choice
1
Enter the item at front end:
14
```

```
1:dinsert front
2:dinsert rear
3:ddelete front
4:ddelete rear
5:ddisplay
6:dsearch
7:dinsert lestpos
8:dinsert rightpos
9:ddelete duplicates
10:exit
enter the choice
1
Enter the item at front end:
15
```

```
1:dinsert front
2:dinsert rear
3:ddelete front
4:ddelete rear
5:ddisplay
6:dsearch
7:dinsert lestpos
8:dinsert rightpos
9:ddelete duplicates
10:exit
enter the choice
1
Enter the item at front end:
```


Git Command Prompt

12

```
1:dinsert front
2:dinsert rear
3:ddelete front
4:ddelete rear
5:ddisplay
6:dsearch
7:dinsert lestpos
8:dinsert rightpos
9:ddelete duplicates
10:exit
```

enter the choice

2

Enter the item at rear end:

18

```
1:dinsert front
2:dinsert rear
3:ddelete front
4:ddelete rear
5:ddisplay
6:dsearch
7:dinsert lestpos
8:dinsert rightpos
9:ddelete duplicates
10:exit
```

enter the choice

5

The contents of the list are:

16

12

15

14

13

12

18

```
1:dinsert front
2:dinsert rear
3:ddelete front
4:ddelete rear
5:ddisplay
```

Command Prompt

18

1:dinsert front

2:dinsert rear

3:ddelete front

4:ddelete rear

5:ddisplay

6:dsearch

7:dinsert lestpos

8:dinsert rightpos

9:ddelete duplicates

10:exit

enter the choice

3

Item deleted at the front end is:16

1:dinsert front

2:dinsert rear

3:ddelete front

4:ddelete rear

5:ddisplay

6:dsearch

7:dinsert lestpos

8:dinsert rightpos

9:ddelete duplicates

10:exit

enter the choice

5

The contents of the list are:

12

15

14

13

12

18

1:dinsert front

2:dinsert rear

3:ddelete front

4:ddelete rear

5:ddisplay

6:dsearch

7:dinsert lestpos

C:\ Command Prompt

```
2:dinsert rear
3:ddelete front
4:ddelete rear
5:ddisplay
6:dsearch
7:dinsert lestpos
8:dinsert rightpos
9:ddelete duplicates
10:exit
enter the choice
6
Enter the key element to be searched:
14
Key element found at the position 3

1:dinsert front
2:dinsert rear
3:ddelete front
4:ddelete rear
5:ddisplay
6:dsearch
7:dinsert lestpos
8:dinsert rightpos
9:ddelete duplicates
10:exit
enter the choice
7
Enter the key element:
12
Enter the item to be inserted at the left of the given item:
4

1:dinsert front
2:dinsert rear
3:ddelete front
4:ddelete rear
5:ddisplay
6:dsearch
7:dinsert lestpos
8:dinsert rightpos
9:ddelete duplicates
10:exit
enter the choice
```


0% Command Prompt

Removed all the duplicate elements of the given item successfully

```
1:dinsert front
2:dinsert rear
3:ddelete front
4:ddelete rear
5:ddisplay
6:dsearch
7:dinsert lestpos
8:dinsert rightpos
9:ddelete duplicates
10:exit
```

enter the choice

5

The contents of the list are:

4
12
15
14
13

```
1:dinsert front
2:dinsert rear
3:ddelete front
4:ddelete rear
5:ddisplay
6:dsearch
7:dinsert lestpos
8:dinsert rightpos
9:ddelete duplicates
10:exit
```

enter the choice

10

D:\coding files\DS lab>

LAB-10

Write a program

- a) To construct a binary Search tree.**
- b) To traverse the tree using all the methods i.e., in-order, preorder and post order**
- c) To display the elements in the tree**

```
#include<stdio.h>
#include<stdlib.h>
struct node{
    int info;
    struct node *rlink,*llink;
};
typedef struct node* NODE;
NODE getnode(){
    NODE x;
    x = (NODE)malloc(sizeof(struct node));
    if(x == NULL){
        printf("Memory full\n");
        exit(0);
    }
    return x;
}

void freenode(NODE x){
    free(x);
}

NODE insert(NODE root, int item){
    NODE temp,cur,prev;
    temp = getnode();
    temp -> rlink = NULL;
    temp -> llink = NULL;
    temp -> info = item;
    if(root == NULL)
        return temp;
    prev = NULL;
    cur = root;
    while(cur != NULL){
        prev = cur;
```

```

    cur =(item<cur->info)?cur->llink:cur->rlink;
}
if(item<prev->info)
    prev -> llink = temp;
else
    prev ->rlink = temp;
return root;
}
void display(NODE root,int i){
    int j;
    if(root != NULL){
        display(root->rlink,i+1);
        for(j=0;j<i;j++)
            printf(" ");
        printf("%d\n",root->info);
        display(root->llink,i+1);
    }
}
void preorder(NODE root){
    if(root!=NULL){
        printf("%d\n",root->info);
        preorder(root->rlink);
        preorder(root->llink);
    }
}
void postorder(NODE root){
    if(root!=NULL){
        postorder(root->llink);
        postorder(root->rlink);
        printf("%d\n",root->info);
    }
}
void inorder(NODE root){
    if(root != NULL){
        inorder(root->llink);
        printf("%d\n",root->info);
        inorder(root->rlink);
    }
}

int main(){
    int item,choice;
    NODE root = NULL;
    for(;;){

```

```
printf("\n1.Insert\n2.Display\n3.Preorder\n4.Postorder\n5.Inorder\n6.Exit\n");
printf("Enter the choice: \n");
scanf("%d:",&choice);
switch(choice){
    case 1: printf("Enter the item \n");
            scanf("%d",&item);
            root = insert(root,item);
            break;
    case 2: display(root,0);
            break;
    case 3: preorder(root);
            break;
    case 4: postorder(root);
            break;
    case 5: inorder(root);
            break;
    default: exit(0);
            break;
}
}
}
```

C:\ Command Prompt - lab10

```
D:\coding files\DS lab>gcc -o lab10 lab10.c
```

```
D:\coding files\DS lab>lab10
```

```
1.Insert  
2.Display  
3.Preorder  
4.Postorder  
5.Inorder  
6.Exit  
Enter the choice:
```

```
1
```

```
Enter the item
```

```
12
```

```
1.Insert  
2.Display  
3.Preorder  
4.Postorder  
5.Inorder  
6.Exit  
Enter the choice:
```

```
1
```

```
Enter the item
```

```
13
```

```
1.Insert  
2.Display  
3.Preorder  
4.Postorder  
5.Inorder  
6.Exit  
Enter the choice:
```

```
1
```

```
Enter the item
```

```
14
```

```
1.Insert  
2.Display  
3.Preorder  
4.Postorder  
5.Inorder
```

CA. Command Prompt - lab10

```
1.Insert
2.Display
3.Preorder
4.Postorder
5.Inorder
6.Exit
```

Enter the choice:

2

14

13

12

```
1.Insert
2.Display
3.Preorder
4.Postorder
5.Inorder
6.Exit
```

Enter the choice:

1

Enter the item

11

```
1.Insert
2.Display
3.Preorder
4.Postorder
5.Inorder
6.Exit
```

Enter the choice:

2

14

13

12

11

```
1.Insert
2.Display
3.Preorder
4.Postorder
5.Inorder
6.Exit
```

Enter the choice:

C:\ Command Prompt - lab10

```
1.Insert
2.Display
3.Preorder
4.Postorder
5.Inorder
6.Exit
```

Enter the choice:

1

Enter the item

18

```
1.Insert
2.Display
3.Preorder
4.Postorder
5.Inorder
6.Exit
```

Enter the choice:

2

18

14

13

12

11

```
1.Insert
2.Display
3.Preorder
4.Postorder
5.Inorder
6.Exit
```

Enter the choice:

3

12

13

14

18

11

```
1.Insert
2.Display
3.Preorder
4.Postorder
```

C:\> Command Prompt

```
14
13
12
11

1.Insert
2.Display
3.Preorder
4.Postorder
5.Inorder
6.Exit
Enter the choice:
3
12
13
14
18
11

1.Insert
2.Display
3.Preorder
4.Postorder
5.Inorder
6.Exit
Enter the choice:
4
11
18
14
13
12

1.Insert
2.Display
3.Preorder
4.Postorder
5.Inorder
6.Exit
Enter the choice:
6

D:\coding files\DS lab>
```