LAB REPORT

NAME: JAYANTI R L

SUBJECT: DATA STRUCTURES

USN: 1BM19CS067

ACADEMIC YEAR 2020-21

LAB PROGRAM-1

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

CODE:

```
 #include<stdio.h>
#include<stdlib.h>
#define STACK_SIZE 5
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)
return -1;
return s[top--];
}
```

```
void display()
{
int i;
if(top==-1)
{
printf("Stack is empty\n");
return;
}
printf("Contents of the stack:\n");
for(i=0;i<=top;i++)
{
printf("%d\n",s[i]);
}
}
void main()
{
int item_deleted;
int choice;
for(;;)
{
printf("\n1.Push\n2.Pop\n3.Display\n4.Exit\n");
printf("Enter the choice\n");
scanf("%d",&choice);
switch(choice)
{
case 1:printf("Enter the item to be inserted\n");
scanf("%d",&item);
push();
break;
case 2:item_deleted=pop();
if(item_deleted==-1)
printf("Stack is empty\n");
else
```

```
printf("Item deleted is %d\n",item_deleted);
break;
case 3:display();
break;
default:exit(0);
}
}
}
```

OUTPUT:

```
1.Push
2.Pop
3.Display
4.Exit
Enter the choice
1
Enter the item to be inserted
2

1.Push
2.Pop
3.Display
4.Exit
Enter the choice
1
Enter the item to be inserted
6

1.Push
2.Pop
3.Display
4.Exit
Enter the choice
2
Item deleted is 6

1.Push
2.Pop
3.Display
4.Exit
Enter the choice
2
Item deleted is 2
```

```
1.Push
2.Pop
3.Display
4.Exit
Enter the choice
2
Stack is empty

1.Push
2.Pop
3.Display
4.Exit
Enter the choice
4

Process returned 0 (0x0)   execution time : 27.132 s
Press any key to continue.
```

LAB PROGRAM-2

2) 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).

CODE:

```
#include<stdio.h>
#include<stdlib.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[]){
int top,j,i;
char s[30],postfix[30];
char 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';
printf("Postfix expression is:\n");
puts(postfix);
}
```

```
int main()
{
char exp[30];
printf("Enter an expression:\n");
gets(exp);
infix_postfix(exp);
return 0;
}
```

OUTPUT :

```
Enter an expression:
(a+b)*(d-f)
Postfix expression is:
ab+df-*

Process returned 0 (0x0)   execution time : 19.456 s
Press any key to continue.
```

```
Enter an expression:
(a+(b-c)*d)
Postfix expression is:
abc-d*+

Process returned 0 (0x0)   execution time : 29.297 s
Press any key to continue.
```

LAB PROGRAM-3

3) WAP to simulate the working of a queue of integers using an array. Provide the following operations a) Insert b) Delete c) Display The program should print appropriate messages for queue empty and queue overflow conditions.

CODE:

```
#include<stdio.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 item 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);
}
}
}
```

OUTPUT :

```
1:Insert rear
2:Delete front
3:Display
4:exit
Enter the choice
1
Enter the item to be inserted
5

1:Insert rear
2:Delete front
3:Display
4:exit
Enter the choice
1
Enter the item to be inserted
8

1:Insert rear
2:Delete front
3:Display
4:exit
Enter the choice
1
Enter the item to be inserted
3

1:Insert rear
2:Delete front
3:Display
4:exit
Enter the choice
1
Enter the item to be inserted
2
Queue overflow

1:Insert rear
2:Delete front
3:Display
4:exit
Enter the choice
3
Contents of queue
5
8
3
```

```
1:Insert rear
2:Delete front
3:Display
4:exit
Enter the choice
2
Item deleted=5

1:Insert rear
2:Delete front
3:Display
4:exit
Enter the choice
2
Item deleted=8

1:Insert rear
2:Delete front
3:Display
4:exit
Enter the choice
2
Item deleted=3

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

Process returned 0 (0x0)   execution time : 388.983 s
Press any key to continue.
```

LAB PROGRAM-4

4) WAP to simulate the working of a Circular queue of integers using an array. Provide the following operations a) Insert b) Delete c) Display The program should print appropriate messages for queue empty and queue overflow conditions

CODE:

```
#include<stdio.h>
#include<conio.h>
#define QUE_SIZE 3
int item,front=0,rear=-1,q[QUE_SIZE],count=0;
void insertrear()
{
if(count==QUE_SIZE)
{
printf("queue overflow\n");
return;
}
rear=(rear+1)%QUE_SIZE;
q[rear]=item;
count++;
}
int deletefront()
{
if(count==0) return -1;
item=q[front];
front=(front+1)%QUE_SIZE;
count=count-1;
return item;
```

```
}
void displayQ()
{
int i,f;
if(count==0)
{
printf("queue is empty\n");
return;
}
f=front;
printf("Contents of queue \n");
for(i=1;i<=count;i++)
{
printf("%d\n",q[f]);
f=(f+1)%QUE_SIZE;
}
}
void main()
{
int choice;
for(;;)
{
printf("\n1:insertrear\n2:deletefront\n3:display\n4:exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
{
case 1:printf("enter the item 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);
}
}
}
```

OUTPUT:

```
1:insertrear
2:deletefront
3:display
4:exit
enter the choice
1
enter the item to be inserted
10

1:insertrear
2:deletefront
3:display
4:exit
enter the choice
1
enter the item to be inserted
20

1:insertrear
2:deletefront
3:display
4:exit
enter the choice
1
enter the item to be inserted
30

1:insertrear
2:deletefront
3:display
4:exit
enter the choice
1
enter the item to be inserted
40
queue overflow

1:insertrear
2:deletefront
3:display
4:exit
enter the choice
3
Contents of queue
10
20
30
```

```
1:insertrear
2:deletefront
3:display
4:exit
enter the choice
2
item deleted =10

1:insertrear
2:deletefront
3:display
4:exit
enter the choice
1
enter the item to be inserted
40

1:insertrear
2:deletefront
3:display
4:exit
enter the choice
3
Contents of queue
20
30
40

1:insertrear
2:deletefront
3:display
4:exit
enter the choice
2
item deleted =20
```

```
1:insertrear
2:deletefront
3:display
4:exit
enter the choice
2
item deleted =30

1:insertrear
2:deletefront
3:display
4:exit
enter the choice
2
item deleted =40

1:insertrear
2:deletefront
3:display
4:exit
enter the choice
2
queue is empty

1:insertrear
2:deletefront
3:display
4:exit
enter the choice
4

Process returned 0 (0x0)   execution time : 68.977 s
Press any key to continue.
```

LAB PROGRAM-5

5) WAP to Implement Singly Linked List with following operations a) Create a linked list. b) Insertion of a node at first position, at any position and at end of list. c) Display the contents of the linked list.

CODE:

```
#include <stdio.h>
#include <conio.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_front(NODE first, int item)
{
NODE temp;
temp = getnode();
temp->info = item;
temp->link = NULL;
if (first == NULL)
return temp;
temp->link = first;
first = temp;
return first;
}
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 insert_pos(int item, int pos, NODE first)
{
NODE temp;
NODE prev, cur;
int count;
temp = getnode();
temp->info = item;
```

```
temp->link = NULL;
if (first == NULL && pos == 1)
return temp;
if (first == NULL)
{
printf("invalid pos\n");
return first;
}
if (pos == 1)
{
temp->link = first;
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("IP\n");
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, pos;
NODE first = NULL;
for (;;)
{
printf("\n1:Insert_front\n2:Insert_rear\n3:insert_pos\n4:display_li
st\n5: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);
first = insert_front(first, item);
break;
case 2:
printf("enter the item at rear-end\n");
scanf("%d", &item);
first = insert_rear(first, item);
break;
case 3:
printf("enter the position and item:\n");
scanf("%d", &pos);
```

```
scanf("%d",&item);
first = insert_pos(item, pos, first);
break;
case 4:
display(first);
break;
default:
exit(0);
}
}
}
```

OUTPUT :

```
1:Insert_front
2:Insert_rear
3:insert_pos
4:display_list
5:Exit
enter the choice
1
enter the item at front-end
20

1:Insert_front
2:Insert_rear
3:insert_pos
4:display_list
5:Exit
enter the choice
1
enter the item at front-end
10

1:Insert_front
2:Insert_rear
3:insert_pos
4:display_list
5:Exit
enter the choice
4
10
20
```

```
1:Insert_front
2:Insert_rear
3:insert_pos
4:display_list
5:Exit
enter the choice
2
enter the item at rear-end
40

1:Insert_front
2:Insert_rear
3:insert_pos
4:display_list
5:Exit
enter the choice
2
enter the item at rear-end
50

1:Insert_front
2:Insert_rear
3:insert_pos
4:display_list
5:Exit
enter the choice
4
10
20
40
50
```

```
1:Insert_front
2:Insert_rear
3:insert_pos
4:display_list
5:Exit
enter the choice
3
enter the position and item:
3 30

1:Insert_front
2:Insert_rear
3:insert_pos
4:display_list
5:Exit
enter the choice
4
10
20
30
40
50

1:Insert_front
2:Insert_rear
3:insert_pos
4:display_list
5:Exit
enter the choice
3
enter the position and item:
6 60

1:Insert_front
2:Insert_rear
3:insert_pos
4:display_list
5:Exit
enter the choice
4
10
20
30
40
50
60
```

```
1:Insert_front
2:Insert_rear
3:insert_pos
4:display_list
5:Exit
enter the choice
5

Process returned 0 (0x0)   execution time : 54.830 s
Press any key to continue.
```

LAB PROGRAM-6

6) WAP to Implement Singly Linked List with following operations a) Create a linked list. b) Deletion of first element, specified element and last element in the list. c) Display the contents of the linked list.

CODE:

```
#include <stdio.h>
#include <conio.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_front(NODE first, int item)
{
NODE temp;
temp = getnode();
temp->info = item;
temp->link = NULL;
if (first == NULL)
return temp;
temp->link = first;
first = temp;
return first;
}
NODE delete_front(NODE first)
{
NODE temp;
if (first == NULL)
{
printf("list is empty cannot delete\n");
return first;
}
temp = first;
temp = temp->link;
printf("item deleted at front-end is=%d\n", first->info);
free(first);
return temp;
}
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("iten deleted at rear-end is %d", cur->info);
free(cur);
prev->link = NULL;
return first;
}
```

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

```
return first;
}
prev->link = cur->link;
printf("iten deleted is %d", cur->info);
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, pos;
NODE first = NULL;
for (;;)
{
printf("\n 1:Insert_front\n 2:Delete_front\n 3:Insert_rear\n
4:Delete_rear\n 5:delete_pos\n 6:display_list\n 7: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);
first = insert_front(first, item);
```

```
break;
case 2:
first = delete_front(first);
break;
case 3:
printf("Enter the item at rear-end\n");
scanf("%d", &item);
first = insert_rear(first, item);
break;
case 4:
first = delete_rear(first);
break;
case 5:
printf("Enter the position:\n");
scanf("%d", &pos);
first = delete_pos(pos, first);
break;
case 6:
display(first);
break;
default:
exit(0);
break;
}
}
}
```

OUTPUT :

```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:delete_pos
 6:display_list
 7:Exit
Enter the choice
1
Enter the item at front-end
10

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:delete_pos
 6:display_list
 7:Exit
Enter the choice
1
Enter the item at front-end
20

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:delete_pos
 6:display_list
 7:Exit
Enter the choice
3
Enter the item at rear-end
30

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:delete_pos
 6:display_list
 7:Exit
Enter the choice
3
Enter the item at rear-end
40
```

```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:delete_pos
 6:display_list
 7:Exit
Enter the choice
6
20
10
30
40

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:delete_pos
 6:display_list
 7:Exit
Enter the choice
2
item deleted at front-end is=20

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:delete_pos
 6:display_list
 7:Exit
Enter the choice
4
iten deleted at rear-end is 40
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:delete_pos
 6:display_list
 7:Exit
Enter the choice
6
10
30
```

```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:delete_pos
 6:display_list
 7:Exit
Enter the choice
1
Enter the item at front-end
50

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:delete_pos
 6:display_list
 7:Exit
Enter the choice
6
50
10
30

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:delete_pos
 6:display_list
 7:Exit
Enter the choice
5
Enter the position:
2
iten deleted is 10
```

```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:delete_pos
 6:display_list
 7:Exit
Enter the choice
6
50
30

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:delete_pos
 6:display_list
 7:Exit
Enter the choice
5
Enter the position:
1
iten deleted is 50
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:delete_pos
 6:display_list
 7:Exit
Enter the choice
2
item deleted at front-end is=30
```

```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:delete_pos
 6:display_list
 7:Exit
Enter the choice
2
list is empty cannot delete

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:delete_pos
 6:display_list
 7:Exit
Enter the choice
7

Process returned 0 (0x0)   execution time : 58.990 s
Press any key to continue.
```

LAB PROGRAM-7 AND 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 d) Stack and Queue Implementation.

CODE:

```
#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_front(NODE first,int item)
{
NODE temp;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL)
return temp;
temp->link=first;
first=temp;
return first;
}
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 insert_pos(int item,int pos,NODE first)
{
NODE temp;
NODE prev,cur;
int count;
temp=getnode();
temp->info=item;
```

```
temp->link=NULL;
if(first==NULL && pos==1)
return temp;
if(first==NULL)
{
printf("invalid pos\n");
return first;
}
if(pos==1)
{
temp->link=first;
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("IP\n");
return first;
}
NODE delete_front(NODE first)
{
```

```
NODE temp;
if(first==NULL)
{
printf("list is empty cannot delete\n");
return first;
}
temp=first;
temp=temp->link;
printf("item deleted at front-end is=%d\n",first->info);
free(first);
return temp;
}
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("iten deleted at rear-end is %d",cur->info);
free(cur);
prev->link=NULL;
return first;
}
NODE delete_pos(int pos,NODE first)
{
NODE prev,cur;
int count;
if (first==NULL || pos<=0)
{
printf("Invalid position\n");
return NULL;
}
if (pos==1)
{
cur=first;
first=first->link;
printf("Item deleted at position %d is %d",pos,cur->info);
freenode(cur);
return first;
}
prev=NULL;
cur=first;
count=1;
while (cur!=NULL)
{
if (count==pos)
{
break;
}
prev=cur;
cur=cur->link;count++;
```

```
}
if (count!=pos)
{
printf("Invalid position\n");
return first;
}
prev->link=cur->link;
printf("Item deleted at position %d is %d",pos,cur->info);
freenode(cur);
return first;
}
NODE order_list(int item,NODE first)
{
NODE temp,prev,cur;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL) return temp;
if(item<first->info)
{
temp->link=first;
return temp;
}
prev=NULL;
cur=first;
while(cur!=NULL&&item>cur->info)
{
prev=cur;
cur=cur->link;
}
prev->link=temp;
temp->link=cur;
return first;
```

```
}
NODE sort(NODE first)
{
int swapped;
NODE ptr1;
NODE lptr = NULL;
if (first == NULL)
return NULL;
do
{
swapped = 0;
ptr1 = first;
while (ptr1->link != lptr)
{
if (ptr1->info > ptr1->link->info)
{
int tem = ptr1->info;
ptr1->info = ptr1->link->info;
ptr1->link->info = tem;
swapped = 1;
}
ptr1 = ptr1->link;
}
lptr = ptr1;
} while (swapped);
}
NODE concat(NODE first,NODE second)
{
NODE cur;
if(first==NULL)
return second;
if(second==NULL)
return first;
```

```
cur=first;
while(cur->link!=NULL)
cur=cur->link;
cur->link=second;
return first;
}
NODE reverse(NODE first)
{
NODE cur,temp;
cur=NULL;
while(first!=NULL)
{
temp=first;
first=first->link;
temp->link=cur;
cur=temp;
}
return cur;
}
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);
}
}
int length(NODE first)
{
NODE cur;
int count=0;
```

```
if(first==NULL) return 0;
cur=first;
while(cur!=NULL)
{
count++;
cur=cur->link;
}
return count;
}
void search(int key,NODE first)
{
NODE cur;
int count1=0;
if(first==NULL)
{
printf("List is empty\n");
return;
}
cur=first;
while(cur!=NULL)
{
count1++;
if(key==cur->info)
break;
cur=cur->link;
}
if(cur==NULL)
{
printf("Search is unsuccessful\n");
return;
}
printf("Search is successfull\n");
printf("Item present at the position number %d\n",count1);
```

```
}
void main()
{
int item,choice,pos,i,n,count,key;
NODE first=NULL,a,b;
for(;;)
{
printf("\n 1:Insert_front\n 2:Insert_rear\n 3:Insert_pos\n
4:Delete_front\n 5:Delete_rear\n 6:Delete_pos\n 7:Sort_list\n
8:Order_list\n 9:Concat\n 10:Reverse List\n 11:Display_list\n
12:Stack\n 13:Queue\n 14:Length of the list\n 15:Search item\n
16: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);
first=insert_front(first,item);
break;
case 2:printf("enter the item at rear-end\n");
scanf("%d",&item);
first=insert_rear(first,item);
break;
case 3:printf("enter the position\n");
scanf("%d",&pos);
printf("Enter the item\n");
scanf("%d",&item);
first=insert_pos(item,pos,first);
break;
case 4:first=delete_front(first);
break;
case 5:first=delete_rear(first);
```

```
break;
case 6:printf("Enter the position:\n");
scanf("%d",&pos);
first=delete_pos(pos,first);
break;
case 7:sort(first);
break;
case 8:printf("Enter the item to be inserted in ordered_list\n");
scanf("%d",&item);
first=order_list(item,first);
break;
case 9:printf("Enter the no of nodes in 1\n");
scanf("%d",&n);
a=NULL;
for(i=0;i<n;i++)
{
printf("Enter the item\n");
scanf("%d",&item);
a=insert_rear(a,item);
}
printf("Enter the no of nodes in 2\n");
scanf("%d",&n);
b=NULL;
for(i=0;i<n;i++)
{
printf("Enter the item\n");
scanf("%d",&item);
b=insert_rear(b,item);
}
a=concat(a,b);
printf("\n");
printf("Items are :\n");
display(a);
```

```
break;
case 10:first=reverse(first);
printf("Items of the reverse list are :\n");
display(first);
break;
case 11:display(first);
break;
case 12:printf("Stack\n");
for(;;)
{
printf("\n 1:Insert_rear\n 2:Delete_rear\n 3:Display_list\n
4: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:display(first);
break;
default:exit(0);
break;
}
}
case 13:printf("QUEUE\n");
for(;;)
{
printf("\n 1:Insert_rear\n 2:Delete_front\n 3:Display_list\n
4: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_front(first);
break;
case 3:display(first);
break;
default:exit(0);
break;
}
}
case 14:count=length(first);
printf("Length(items) in the list is %d\n",count);
break;
case 15:printf("Enter the item to be searched\n");
scanf("%d",&key);
search(key,first);
break;
default:exit(0);
break;
}
}
getch();
}
```

OUTPUT :

```
 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
1
enter the item at front-end
10

 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
1
enter the item at front-end
20
```

```
 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
2
enter the item at rear-end
30

 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
2
enter the item at rear-end
40
```

```
 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
11
20
10
30
40

 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
3
enter the position
3
Enter the item
50
```

```
 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
11
20
10
50
30
40

 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
4
item deleted at front-end is=20
```

```
 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
5
iten deleted at rear-end is 40
 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
11
10
50
30
```

```
 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
6
Enter the position:
2
Item deleted at position 2 is 50
 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
11
10
30
```

```
 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
1
enter the item at front-end
20

 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
1
enter the item at front-end
40
```

```
 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
11
40
20
10
30
```

```
 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
7

 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
11
10
20
30
40
```

```
 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
8
Enter the item to be inserted in ordered_list
25

 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
11
10
20
25
30
40
```

```
 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
10
Items of the reverse list are :
40
30
25
20
10
```

```
 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
14
Length(items) in the list is 5
```

```
 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
15
Enter the item to be searched
25
Search is successfull
Item present at the position number 3
```

```
 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
15
Enter the item to be searched
60
Search is unsuccessful
```

```
 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
9
Enter the no of nodes in 1
2
Enter the item
10
Enter the item
20
Enter the no of nodes in 2
3
Enter the item
30
Enter the item
40
Enter the item
50

Items are :
10
20
30
40
50
```

```
 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
12
Stack

 1:Insert_rear
 2:Delete_rear
 3:Display_list
 4:Exit
Enter the choice
1
Enter the item at rear-end
50

 1:Insert_rear
 2:Delete_rear
 3:Display_list
 4:Exit
Enter the choice
3
40
30
25
20
10
50
```

```
 1:Insert_rear
 2:Delete_rear
 3:Display_list
 4:Exit
Enter the choice
1
Enter the item at rear-end
60

 1:Insert_rear
 2:Delete_rear
 3:Display_list
 4:Exit
Enter the choice
2
iten deleted at rear-end is 60
 1:Insert_rear
 2:Delete_rear
 3:Display_list
 4:Exit
Enter the choice
3
40
30
25
20
10
50

 1:Insert_rear
 2:Delete_rear
 3:Display_list
 4:Exit
Enter the choice
4
```

```
 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
11
40
30
25
20
10
50
```

```
 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Delete_front
 5:Delete_rear
 6:Delete_pos
 7:Sort_list
 8:Order_list
 9:Concat
 10:Reverse List
 11:Display_list
 12:Stack
 13:Queue
 14:Length of the list
 15:Search item
 16:Exit
Enter the choice
13
QUEUE

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
Enter the choice
1
Enter the item at rear-end
60

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
Enter the choice
3
40
30
25
20
10
50
60
```

```
 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
Enter the choice
2
item deleted at front-end is=40

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
Enter the choice
3
30
25
20
10
50
60

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
Enter the choice
4

Process returned 0 (0x0)   execution time : 102.247 s
Press any key to continue.
```

LAB PROGRAM-9

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.

CODE:

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

```
}
NODE dinsert_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 unsuccessfull\n");
}
else
{
printf("Key element found at the position %d\n",count);
}
```

```
}
NODE dinsert_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("All the duplicate elements of the given item are removed
successfully\n");
}
return head;
}
NODE delete_all_key(int item,NODE head)
{
NODE prev,cur,next;
int count;
if(head->rlink==head)
{
printf("LE");
return head;
}
count=0;
cur=head->rlink;
while(cur!=head)
{
if(item!=cur->info)
cur=cur->rlink;
```

```
else
{
count++;
prev=cur->llink;
next=cur->rlink;
prev->rlink=next;
next->llink=prev;
freenode(cur);
cur=next;
}
}
if(count==0)
printf("Key not found");
else
printf("Key found at %d positions and are deleted\n", count);
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
lestpos\n8:dinsert rightpos\n9:ddelete
duplicates\n10:ddelete_based on specified value\n11: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;
case 10:printf("Enter the key value\n");
```

```
scanf("%d",&item);
delete_all_key(item,head);
break;
case 11:exit(0);
default:printf("Invalid choice\n");
}
}
return 0;
}
```

OUTPUT :

```
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:ddelete_based on specified value
11:exit
Enter the choice
1
Enter the item at front end:
1

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:ddelete_based on specified value
11:exit
Enter the choice
1
Enter the item at front end:
2
```

```
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:ddelete_based on specified value
11:exit
Enter the choice
1
Enter the item at front end:
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:ddelete_based on specified value
11:exit
Enter the choice
5
The contents of the list are:
3
2
1
```

```
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:ddelete_based on specified value
11:exit
Enter the choice
7
Enter the key element:
2
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:ddelete_based on specified value
11:exit
Enter the choice
5
The contents of the list are:
3
4
2
1
```

```
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:ddelete_based on specified value
11:exit
Enter the choice
8
Enter the key element:
4
Enter the item to be inserted at the right of the given item:
5

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:ddelete_based on specified value
11:exit
Enter the choice
5
The contents of the list are:
3
4
5
2
1
```

```
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:ddelete_based on specified value
11:exit
Enter the choice
6
Enter the key element to be searched:
2
Key element found at the position 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:ddelete_based on specified value
11:exit
Enter the choice
6
Enter the key element to be searched:
9
Search unsuccessfull
```

```
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:ddelete_based on specified value
11:exit
Enter the choice
2
Enter the item at rear end:
2

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:ddelete_based on specified value
11:exit
Enter the choice
5
The contents of the list are:
3
4
5
2
1
2
```

```
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:ddelete_based on specified value
11:exit
Enter the choice
9
Enter the key element whose duplicates should be removed:
2
All the duplicate elements of the given item are removed 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:ddelete_based on specified value
11:exit
Enter the choice
5
The contents of the list are:
3
4
5
2
1
```

```
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:ddelete_based on specified value
11:exit
Enter the choice
10
Enter the key value
5
Key found at 1 positions and are deleted

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:ddelete_based on specified value
11:exit
Enter the choice
5
The contents of the list are:
3
4
2
1
```

```
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:ddelete_based on specified value
11:exit
Enter the choice
3
Item deleted at the front end is: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:ddelete_based on specified value
11:exit
Enter the choice
4
Item deleted at the rear end is:1
```

```
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:ddelete_based on specified value
11:exit
Enter the choice
3
Item deleted at the front end is: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:ddelete_based on specified value
11:exit
Enter the choice
4
Item deleted at the rear end is:2
```

```
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:ddelete_based on specified value
11:exit
Enter the choice
3
List is empty
```

```
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:ddelete_based on specified value
11:exit
Enter the choice
30
Invalid choice

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:ddelete_based on specified value
11:exit
Enter the choice
11

Process returned 0 (0x0)   execution time : 616.916 s
Press any key to continue.
```

LAB PROGRAM-10

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.

CODE:

```
#include<stdio.h>
#include<conio.h>
#include<process.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("mem 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);
}
}
NODE delete(NODE root,int item)
{
NODE cur,parent,q,suc;
if(root==NULL)
{
printf("empty\n");
return root;
}
parent=NULL;
cur=root;
while(cur!=NULL&&item!=cur->info)
{
parent=cur;
cur=(item<cur->info)?cur->llink:cur->rlink;
}
if(cur==NULL)
{
printf("not found\n");
return root;
}
if(cur->llink==NULL)
q=cur->rlink;
else if(cur->rlink==NULL)
q=cur->llink;
else
{
suc=cur->rlink;
while(suc->llink!=NULL)
suc=suc->llink;
suc->llink=cur->llink;
```

```
q=cur->rlink;
}
if(parent==NULL)
return q;
if(cur==parent->llink)
parent->llink=q;
else
parent->rlink=q;
freenode(cur);
return root;
}
void preorder(NODE root)
{
if(root!=NULL)
{
printf("%d\n",root->info);
preorder(root->llink);
preorder(root->rlink);
}
}
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);
}
}
void main()
{
int item,choice;
NODE root=NULL;
for(;;)
{
printf("\n1.Insert\n2.Display\n3.Pre Order\n4.Post Order\n5.In
Order\n6.Delete\n7.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;
case 6:printf("enter the item\n");
scanf("%d",&item);
root=delete(root,item);
break;
```

```
default:exit(0);
break;
}
}
}
```

OUTPUT :

```
1.Insert
2.Display
3.Pre Order
4.Post Order
5.In Order
6.Delete
7.Exit
enter the choice
1
enter the item
10

1.Insert
2.Display
3.Pre Order
4.Post Order
5.In Order
6.Delete
7.Exit
enter the choice
2
10
```

```
1.Insert
2.Display
3.Pre Order
4.Post Order
5.In Order
6.Delete
7.Exit
enter the choice
1
enter the item
20

1.Insert
2.Display
3.Pre Order
4.Post Order
5.In Order
6.Delete
7.Exit
enter the choice
1
enter the item
5

1.Insert
2.Display
3.Pre Order
4.Post Order
5.In Order
6.Delete
7.Exit
enter the choice
2
  20
10
  5
```

```
1.Insert
2.Display
3.Pre Order
4.Post Order
5.In Order
6.Delete
7.Exit
enter the choice
1
enter the item
15

1.Insert
2.Display
3.Pre Order
4.Post Order
5.In Order
6.Delete
7.Exit
enter the choice
2
   20
      15
10
   5
```

```
1.Insert
2.Display
3.Pre Order
4.Post Order
5.In Order
6.Delete
7.Exit
enter the choice
1
enter the item
1

1.Insert
2.Display
3.Pre Order
4.Post Order
5.In Order
6.Delete
7.Exit
enter the choice
2
   20
     15
10
   5
     1
```

```
1.Insert
2.Display
3.Pre Order
4.Post Order
5.In Order
6.Delete
7.Exit
enter the choice
1
enter the item
30

1.Insert
2.Display
3.Pre Order
4.Post Order
5.In Order
6.Delete
7.Exit
enter the choice
2
     30
  20
     15
10
  5
     1
```

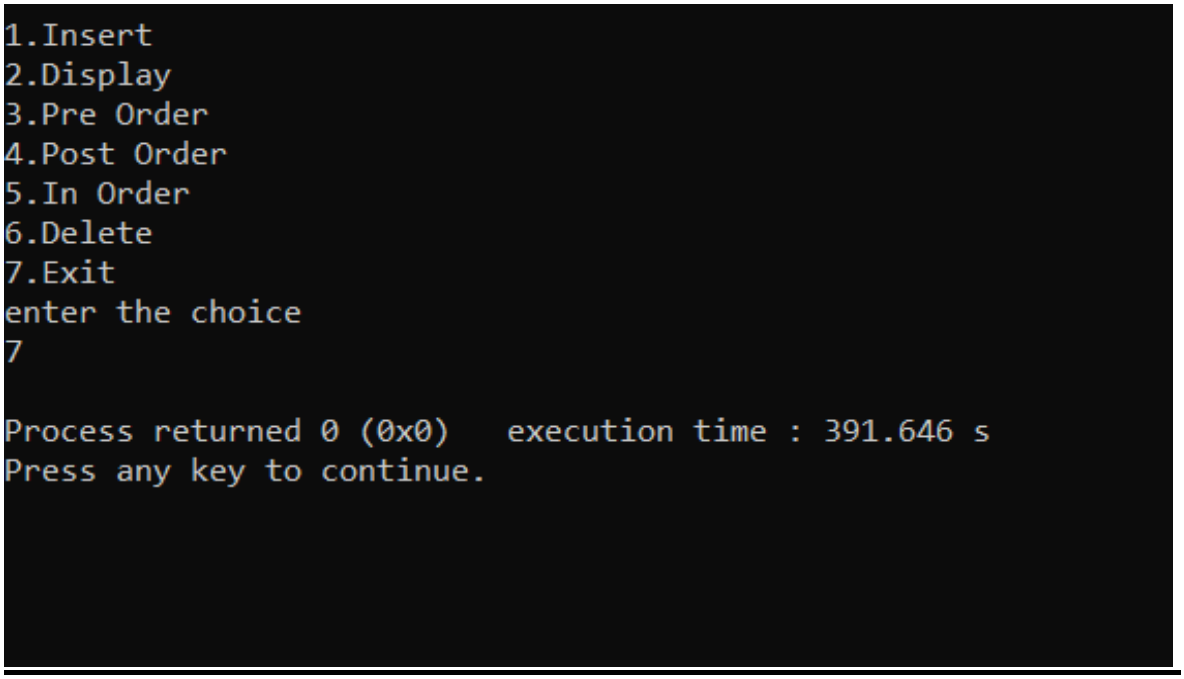
```
1.Insert
2.Display
3.Pre Order
4.Post Order
5.In Order
6.Delete
7.Exit
enter the choice
1
enter the item
7

1.Insert
2.Display
3.Pre Order
4.Post Order
5.In Order
6.Delete
7.Exit
enter the choice
2
     30
  20
     15
10
     7
  5
     1
```

```
1.Insert
2.Display
3.Pre Order
4.Post Order
5.In Order
6.Delete
7.Exit
enter the choice
3
10
5
1
7
20
15
30

1.Insert
2.Display
3.Pre Order
4.Post Order
5.In Order
6.Delete
7.Exit
enter the choice
4
1
7
5
15
30
20
10

1.Insert
2.Display
3.Pre Order
4.Post Order
5.In Order
6.Delete
7.Exit
enter the choice
5
1
5
7
10
15
20
30
```


1.Insert
2.Display
3.Pre Order
4.Post Order
5.In Order
6.Delete
7.Exit
enter the choice
7
Process returned 0 (0x0)   execution time : 391.646 s
Press any key to continue.