LAB PROGRAM 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 <process.h>
#include <conio.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; printf("Item has been inserted\n");
}
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 ;
clrscr();
for(;;)
{
printf ("\n 1. PUSH\n 2: POP \n 3: DISPLAY \n 4:EXIT\n");
printf ("Enter your choice \n");
scanf ("%d", &choice);
switch (choice)
{
case 1: printf("Enter the item to be inserted \n");
scanf("%d", &item );
push();
printf(" item has been inserted \n");
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 : printf("Invalid choice\n");
    }
  }
}
```

OUTPUT:

```
1. PUSH
2. POP
3. DISPLAY
Enter your choice
1
Enter the item to be inserted
4
Item has been inserted

1. PUSH
2. POP
3. DISPLAY
Enter your choice
1
Enter the item to be inserted
6
Item has been inserted
1. PUSH
2. POP
3. DISPLAY
Enter your choice
```

1
Enter the item to be inserted
98
Item has been inserted
1. PUSH
2. POP
3. DISPLAY
Enter your choice
1
Enter the item to be inserted
33
Item has been inserted
1. PUSH
2. POP
3. DISPLAY
Enter your choice
1.
Enter the item to be inserted
45
Item has been inserted
1. PUSH
2. POP
3. DISPLAY
Enter your choice
1
Enter item to be inserted
2
Stack overflow
1. PUSH
2. POP
3. DISPLAY
Enter your choice

3
Contents of the stack are:
4
6
98
33
45

1. PUSH
2. POP
3. DISPLAY
Enter your choice
2
Item deleted is 45
1. PUSH
2. POP
3. DISPLAY
Enter your choice
2
Item deleted is 33
1. PUSH
2. POP
3. DISPLAY
Enter your choice
2
Item deleted is 98
1. PUSH
2. POP
3. DISPLAY
Enter your choice
2
Item deleted is 6
1. PUSH
2. POP
3. DISPLAY
Enter your choice
2
Item deleted is 4
1. PUSH
2. POP
3. DISPLAY
Enter your choice
4
stack is empty
1. PUSH
2. POP
3. DISPLAY
Enter your choice.

```
 1. PUSH
 2. POP
 3. DISPLAY
 Enter your choice
1
Enter the item to be inserted
4
Item has been inserted

 1. PUSH
 2. POP
 3. DISPLAY
 Enter your choice
1
Enter the item to be inserted
6
Item has been inserted

 1. PUSH
 2. POP
 3. DISPLAY
 Enter your choice
1
Enter the item to be inserted
98
Item has been inserted

 1. PUSH
 2. POP
 3. DISPLAY
 Enter your choice
1
Enter the item to be inserted
```

```
33
Item has been inserted

 1. PUSH
 2. POP
 3. DISPLAY
 Enter your choice
1
Enter the item to be inserted
45
Item has been inserted

 1. PUSH
 2. POP
 3. DISPLAY
 Enter your choice
1
Enter the item to be inserted
2
Stack overflow
Item has been inserted

 1. PUSH
 2. POP
 3. DISPLAY
 Enter your choice
3
Contents of the stack are:
4
6
98
33
45
```

```
 1. PUSH
 2. POP
 3. DISPLAY
 Enter your choice
2
Item deleted is 98

 1. PUSH
 2. POP
 3. DISPLAY
 Enter your choice
2
Item deleted is 6

 1. PUSH
 2. POP
 3. DISPLAY
 Enter your choice
2
Item deleted is 4

 1. PUSH
 2. POP
 3. DISPLAY
 Enter your choice
2
Stack is empty

 1. PUSH
 2. POP
 3. DISPLAY
 Enter your choice
```

```
 1. PUSH
 2. POP
 3. DISPLAY
 Enter your choice
1
Enter the item to be inserted
4
Item has been inserted

 1. PUSH
 2. POP
 3. DISPLAY
 Enter your choice
1
Enter the item to be inserted
6
Item has been inserted

 1. PUSH
 2. POP
 3. DISPLAY
 Enter your choice
1
Enter the item to be inserted
98
Item has been inserted

 1. PUSH
 2. POP
 3. DISPLAY
 Enter your choice
1
Enter the item to be inserted
```

LAB PROGRAM 2: Write a program 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>
#include <process.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];
 top = -1;
  S [++top] = '#';
   j=0;
  for (i=0; i<strlen( infix) ;i++ )
  {
char 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';
    }
   void main()
   {
   char infix[20];
   char postfix[20];
    clrscr;
   printf("Enter the valid infix expression\n");
    scanf("%s", infix);
    infix_postfix(infix, postfix);
    Printf("the postfix expression is \n");
    printf("%s\n", postfix);
     getch();
     }
```

OUTPUT :-

Enter the valid infix expression

(1 + 2)* 3/4 - 5

The postfix expression is : 12+3*4/5-

```
Enter the valid infix expression
(1+2)*3/4-5
the postfix expression is :12+3*4/5-

[Program finished]
```

→ Circular Queue

```
#include <stdio.h>
#include <stdlib.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=0; i<= count; i++)
  {
   printf("%d \n", q[f]);
   f=(f+1) % que_size;
  }
}
int main()
 {
  int choice;
  for(;;)
  {
    printf("\n1. Insert Rear \n 2. Delete front \n 3.
Display \n 4. Exit \n");
printf("Enter your choice : ");
scanf("%d", &choice);
switch(choice)
 {

case 1: printf("Enter the item to be inserted : ");
        scanf("%d", &item);
        insertrear();
        break;
```

```
case 2: item = deletefront();
        if (item == -1)
          printf("Queue is empty\n");
        else
          printf("Item deleted is %d\n", item);
        break;
case 3: displayq();
        break;
default: exit(0);
    }
  }
}
```

OUTPUT:-

1. Insert rear
2. Delete front
3. Display
4. Exit

Enter your choice : 1
Enter the item to be inserted : 3

1. Insert rear
2. Delete front
3. Display
4. Exit

Enter your choice : 1
Enter the item to be inserted : 5

1. Insert rear

2. Delete front
3. Display
4. Exit

Enter your choice : 1
Enter the item to be inserted : 7

1. Insert rear
2. Delete front
3. Display
4. Exit

Enter your choice : 1
Enter the item to be inserted : 9
queue overflow

1. Insert rear
2. Delete front
3. Display
4. Exit

Enter your choice : 3
Contents of queue
3
5
7
3

1. Insert rear
2. Delete front
3. Display
4. Exit

Enter your choice : 2
Item deleted is 3

1. Insert rear
2. Delete front
3. Display
4. Exit

Enter your choice : 2

Item deleted is 5

1. Insert rear
2. Delete front
3. Display
4. Exit

Enter your choice : 2

Item deleted is 7.

```
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 :5

1.Insert rear
2.Delete front
3.Display
4.exit
 Enter the choice : 1
Enter the item to be inserted :7

1.Insert rear
2.Delete front
3.Display
4.exit
 Enter the choice : 1
Enter the item to be inserted :9
queue overflow
1.Insert rear
2.Delete front
3.Display
4.exit
 Enter the choice : 3
contents of queue
3
5
7
3

1.Insert rear
2.Delete front
3.Display
4.exit
 Enter the choice : 2
item deleted is 3

1.Insert rear
2.Delete front
3.Display
4.exit
 Enter the choice : 2
item deleted is 5

1.Insert rear
2.Delete front
3.Display
4.exit
 Enter the choice : 2
item deleted is 7
```

→ Dequeue:-

```
#include<stdio.h>
#include<conio.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 is empty\n");
   return;
  }
for (i=f; i<=r; i++)
printf("%d\n", q[i]);
}
int main()
{
 clrscr();
 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);
    }
  }
}
```

OUTPUT:-

1. Insert_rear
2. insert_front
3. delete_rear
4. delete_front
5. Display
6. Exit

Enter choice

1

Enter the item

1

1. Insert_rear
2. ~~Delete_f~~ Insert_front
3. Delete_rear
4. Delete_front
5. Display
6. Exit

Enter ~~the item~~ choice

1

Enter the item

2

1. Insert_rear
2. insert_front
3. delete_rear
4. delete_front
5. display
6. Exit

5

1

2

```
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
1
enter the item
1
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
1
enter the item
2
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
1
enter the item
3
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
1
enter the item
4
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
1
enter the item
5
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
5
1
2
3
```

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

→ Queue :-

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

```
return ;
}
printf("Contents of queue \n");
for (i = front; i <= rear; i++)
 {
 printf("%d\n", q[i]);
}}
int main()
{
 int choice;
 for (;;)
 {
  printf("1. Insert rear 2. Delete front 3. Display
          4. 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: display Q();
break;
default: exit(0);
}
}
}
```

OUTPUT:

```
Insert rear 2. deletefront 3. Display 4. Exit
enter the choice
1
Enter the item to be inserted
1
1. Insert rear 2. deletefront 3. Display 4. Exit
Enter the choice
1
Enter the item to be inseeted
2
1. Insert rear 2. delete front 3. Display 4. Exit
Enter the choice
3
Contents of queue
1
2
1. Insert rear 2. delete front 3. Display 4. Exit
Enter the choice
2
item deleted = 1
1. Insert rear 2. deletefront 3. Display 4. Exit
Enter the choice
2
item deleted = 2
```

```
1:insertrear 2:deletefront 3:display 4:exit
enter the choice
1
enter the item to be inserted
1
1:insertrear 2:deletefront 3:display 4:exit
enter the choice
1
enter the item to be inserted
2
1:insertrear 2:deletefront 3:display 4:exit
enter the choice
1
enter the item to be inserted
3
1:insertrear 2:deletefront 3:display 4:exit
enter the choice
3
contents of queue
1
2
3
1:insertrear 2:deletefront 3:display 4:exit
enter the choice
2
item deleted=1
1:insertrear 2:deletefront 3:display 4:exit
enter the choice
2
item deleted=2
1:insertrear 2:deletefront 3:display 4:exit
enter the choice
2
item deleted=3
1:insertrear 2:deletefront 3:display 4:exit
enter the choice
3
queue is empty
1:insertrear 2:deletefront 3:display 4:exit
enter the choice
```

Q.) Singly linked list with sort, reverse and concatenation.

```
#include <stdio.h>
# include <conio.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("memory 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 IF (NODE second, int item)
{
 NODE temp;
 temp = getnode();
 temp -> info = item;
 temp -> link = NULL;
  if (second == NULL)
   return temp;
   temp -> link = second;
   second = temp;
   return second;
   }
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 IR (NODE second, int item)
{
  NODE temp, cur;
  temp = getnode();
  temp -> info = item;
  temp -> link = NULL;
  if (second == NULL)
  return temp;
  cur = second;
  while (cur -> link != NULL)
  cur = cur -> link;
  cur -> link = temp;
  return second;
  }
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 %od ", cur -> info);
free(cur);
prev -> link = NULL;
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("Invalid Position\n");
return first;
}
NODE delete_pos(int pos, NODE first)
{
NODE cur;
NODE prev;
int count;
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)
break; // if found
prev = cur;
cur = cur -> link;
count++;
}
if(count != pos)
{
printf("Invalid Position\n");
return first;
```

```
}
if (count != pos)
{
  printf("Invalid position specified\n");
  return first;
}
prev->link = cur->link;
freenode(cur);
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;
}
NODE asc(NODE first)
{
  NODE prev = first;
  NODE cur = NULL;
  int temp;
if (first == NULL) {
  return 0;
}
else {
  while (prev != NULL) {
  cur = prev->link;
  while (cur != NULL) {
  if (prev->info > cur->info) {
  temp = prev->info;
  prev->info = cur->info;
```

```
cur->info = temp;
}
cur = cur->link;
}
prev = prev->link;
}}
return first;
}
NODE des(NODE first)
{
  NODE prev = first;
  NODE cur = NULL;
    int temp;
  if (first == NULL) {
   return 0;
  } else {
  while (prev != NULL) {
  cur = prev->link;
  while (cur != NULL) {
  if (prev->info < cur->info) {
   temp = prev->info;
   prev->info = cur->info;
   cur->info = temp;
   }
   cur = cur->link;
   }
   prev = prev->link;
    }}
    return first;
  }
NODE concate (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;
}
void display (NODE first)
{
 NODE temp;
 if (first==NULL)
 printf("List is empty. Can't display items\n");
 for(temp = first ;temp!= NULL;temp= temp-> link)
 {
  printf("%d\n", temp-> info);
 }}
 void main()
 {
  int item, choice, pos, element, option, choice2, item1, num;
  NODE first = NULL;
  NODE second = NULL;
  for(;;)
  {
  printf("\n. 1. Insert_front \n 2. Delete_front \n 3. Insert_rea
 \n 4. Delete_rear\n 5. random-position \n 6. Reverse \n 7. Sort\n
 8. Concatenate \n 9. Display_list\n 10. Exit \n");
 printf("Enter the choice\n");
  scanf("%d", &choice);
  switch(choice)
  {
  case 1: printf("Enter the item at the 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 the rear end\n");
```

```
scanf("%d", &item);
first = insert_rear(first, item);
break;
case 4: first = delete_rear(first);
        break;
case 5: printf("press 1 to insert & 2 to delete at any desired
         position \n");
        scanf("%d", &element);
        if (element == 1) {
        printf("enter the position to insert \n");
         scanf("%d", &pos);
         printf("enter the item to insert \n");
         scanf("%d", &item);
         first = insert_pos(item, pos, first);
        }
        if (element == 2) {
          printf("enter the position to delete \n");
          scanf("%d", &pos);
          first = delete_pos(pos, first);
        }
        break;
case 6: first = reverse(first);
        break;
case 7: printf("press 1 for ascending sort & 2 for descending
          sort \n");
        scanf("%d", &option);
        if (option == 1);
        first = asc(first);
        if (option == 2)
        first = des(first);
          break;
case 8: printf("create a second list \n");
        printf("Enter the number of elements in 2nd list \n");
        scanf("%d", &num);
```

```
for (int i=1;i<=num;i++) {
printf("\n press 1 to insert front & 2 to insert rear \n");
 scanf("%d", &choice2);
 if (choice2==1) {
printf("enter the item at front end \n");
 scanf("%d", &item1);
 second= IF (second,item1);
 }
if (choice2==2) {
printf("Enter the item at rear end \n");
 scanf("%d", &item1);
 second= IR(second,item1);
 }}
first = concate(first, second);
 break;
case 9: display(first);
 break;
default: exit(0);
 break;
}}
 getch();
}
```

```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
1
enter the item at front-end
2

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
3
enter the item at rear-end
3

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
9
2
3

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
2
item deleted at front-end is=2

 1:Insert_front
```

```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
1
enter the item at front-end
6

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
4
iten deleted at rear-end is 3
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
3
enter the item at rear-end
8

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
1
enter the item at front-end
2

 1:Insert_front
 2:Delete_front
```

```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
5
press 1 to insert or 2 to delete at any desired position
1
enter the position to insert
3
enter the item to insert
4

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
9
2
6
4
8

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
6

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
```

```
enter the choice
9
8
4
6
2

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
7
press 1 for ascending sort and 2 for descending sort:
1

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
9
2
4
6
8

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
8
create a second list
enter the number of elements in second list
3

 press 1 to insert front and 2 to insert rear
1
enter the item at front-end
1

 press 1 to insert front and 2 to insert rear
```

```
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
8
create a second list
enter the number of elements in second list
3

 press 1 to insert front and 2 to insert rear
1
enter the item at front-end
1

 press 1 to insert front and 2 to insert rear
1
enter the item at front-end
2

 press 1 to insert front and 2 to insert rear
1
enter the item at front-end
3

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
9
2
4
6
8
3
2
1

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:random_position
 6:reverse
 7:sort
 8.concate
 9:display_list
 10:Exit
enter the choice
```

→ linked list program to count, sort and search.

```
#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("memory is full\n");
exit(0);
}
return x;
}
void freenode(NODE x)
{
free(x);
NODE insertfront(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 deleterear(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;
}
void display(NODE first)
{
NODE temp;
if(first ==NULL)
printf("List is empty.\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;
if(first==NULL)
{
printf("List is empty\n");
return;
}
cur = first;
while(cur != NULL)
{
if(key == cur->info)
break;
cur = cur->link;
}
if(cur==NULL)
{
printf("Search is unsuccesful \n");
return;
}
printf("Search is successful. Element is found in the list at %d",
                                                    cur->link);
}
NODE asc(NODE first)
{
  NODE prev = first;
  NODE cur = NULL;
  int temp;
if(first==NULL) { return 0; }
else { while(prev != NULL) {
cur = prev->link;
while(cur != NULL) {
if (prev->info > cur->info) {
temp = prev->info;
```

```
    prev -> info = cur -> info;
    cur -> info = temp;
    }
    cur = cur -> link;
    }
    prev = prev -> link;
  }} return first; }
NODE des(NODE first)
 {
  NODE prev = first;
  NODE cur = NULL;
  int temp;
 if (first == NULL) {
  return 0;
 } else {
while(prev != NULL) {
 cur = prev -> link;
 while (cur != NULL) {
if (prev -> info < cur -> info) {
temp = prev -> info;
 prev -> info = cur -> info;
 cur -> info = temp;
  }
 cur = cur -> link;
} prev = prev -> link;
}} return first; }
int main()
{
int item, choice, count, key, option;
NODE first = NULL;
for(;;)
{
printf("\n1. Insert front \n2. Delete rear \n3. Display list \n
4. Count items \n5. Search item \n6. Order list \n7. Exit\n");
printf("Enter your choice \n");
scanf("%d", &choice);
switch(choice) {
```

```
case 1: printf("Enter the item at front end \n");
        scanf("%d", &item);
        first = insertfront(first, item);
        break;
case 2: first = deletefront(first);
        break;
case 3: display(first);
        break;
case 4: count = length(first);
        printf("Length of items in the list is %d \n", count);
        break;
case 5: printf("Enter the item to be searched \n");
        scanf("%d", &key);
        search(key, first);
        break;
case 6: printf("1. Ascending order \n 2. Descending order \n");
        scanf("%d", &option);
        if (option == 1)
        {
          first=asc(first);
          display(first);
        }
        else {
          first = desc(first);
          display(first);
        }
        break;
        default: exit(0);
        }}
    return 0;
}
```

```
main.c:119:77: warning: format '%d' expects argument of type 'int', but argument 2 has type 'struct node *' [-Wformat=
]

 1:Insert front
 2:Delete rear
 3:Display list
 4:Count items
 5:Search items
 6:Order list
 7:Exit
Enter the choice
1
Enter the item at front-end
21

 1:Insert front
 2:Delete rear
 3:Display list
 4:Count items
 5:Search items
 6:Order list
 7:Exit
Enter the choice
1
Enter the item at front-end
32
```

```
 1:Insert front
 2:Delete rear
 3:Display list
 4:Count items
 5:Search items
 6:Order list
 7:Exit
Enter the choice
1
Enter the item at front-end
22

 1:Insert front
 2:Delete rear
 3:Display list
 4:Count items
 5:Search items
 6:Order list
 7:Exit
Enter the choice
1
Enter the item at front-end
54

 1:Insert front
 2:Delete rear
```

```
 1:Insert front
 2:Delete rear
 3:Display list
 4:Count items
 5:Search items
 6:Order list
 7:Exit
Enter the choice
6

1:Ascending ordered list
2:Descending ordered list
1
21
22
32
54

 1:Insert front
 2:Delete rear
 3:Display list
 4:Count items
 5:Search items
 6:Order list
 7:Exit
Enter the choice
```

```
 1:Insert front
 2:Delete rear
 3:Display list
 4:Count items
 5:Search items
 6:Order list
 7:Exit
Enter the choice
6

1:Ascending ordered list
2:Descending ordered list
2
54
32
22
21

 1:Insert front
 2:Delete rear
 3:Display list
 4:Count items
 5:Search items
 6:Order list
 7:Exit
Enter the choice
```

```
 1:Insert front
 2:Delete rear
 3:Display list
 4:Count items
 5:Search items
 6:Order list
 7:Exit
Enter the choice
3
54
22
32
21

 1:Insert front
 2:Delete rear
 3:Display list
 4:Count items
 5:Search items
 6:Order list
 7:Exit
Enter the choice
4
Length of items in the list is 4

 1:Insert front
```

```
 1:Insert front
 2:Delete rear
 3:Display list
 4:Count items
 5:Search items
 6:Order list
 7:Exit
Enter the choice
5
Enter the item to be searched
55
Search is unsuccessful

 1:Insert front
 2:Delete rear
 3:Display list
 4:Count items
 5:Search items
 6:Order list
 7:Exit
Enter the choice
7


...Program finished with exit code 0
Press ENTER to exit console.
```

→ WAP to implement doubly linked list with following operations: a) Create a doubly linked list.
b) Insert front & rear.
c) Delete the node both front & rear.
d) Display the contents of the list.
e) Simple search
f) Inserting the node before and after key node.
g) Deleting repeating occurances.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
  int info;
  struct node *llink;
  struct node *rlink;
};
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)
{
```

```
inorder(root);
}
break;
case 4: if (root==NULL)
        {
        printf("Tree is empty");
        }
       else
        {
        printf("Given tree is \n");
        display(root, 1);
        printf("The postorder traversal is \n");
        postorder(root);
        }
      break;
case 5: display(root,1);
        break;
default: exit(0);
}}}
```

```
NODE temp, cur;
temp = getnode();
temp->info = item;
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;
cur = head->llink;
head->llink = temp;
temp->rlink = head;
temp->llink = cur;
cur->rlink = temp;
return head;
}
NODE ddelete_front(NODE head)
{
NODE cur, next;
if(head->rlink == head)
{
printf("linked list is empty\n");
return head;
}
cur = head->rlink;
next = cur->rlink;
head->rlink = next;
next->llink = head;
printf("The node deleted is %d", cur->info);
freenode(cur);
return head;
}
NODE ddelete_rear(NODE head)
{
NODE cur, prev;
```

```
if (head -> rlink == head)
{
printf("linked list is empty\n");
return head;
}
cur = head -> llink;
prev = cur -> llink;
head -> llink = prev;
prev -> rlink = head;
printf("The node deleted is %d", cur -> info);
freenode (cur);
return head;
}
NODE insert_leftpos (int item, NODE head)
{
NODE temp, cur, prev;
if (head -> rlink == head)
{
printf("list is empty\n"));
return head;
}
cur = head -> rlink;
while (cur != head)
{
if (item == cur -> info) break;
cur = cur -> rlink;
}
if (cur == head)
{
printf("Key not found\n");
return head;
}
prev = cur -> llink;
printf("Enter towards left of %d: ", item);
temp = getnode();
scanf("%d", &temp -> info);
prev -> rlink = temp;
```

```
temp -> llink = prev;
cur -> llink = temp;
temp -> rlink = cur;
return head;
}
NODE insert_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 (item == cur -> info) break;
cur = cur -> rlink;
}
if (cur == head)
{
printf ("Key not found\n");
return head;
}
next = cur -> rlink;
printf ("Enter towards right of %d :", item);
temp = getnode();
scanf ("%d", &temp -> info);
cur -> rlink = temp;
temp -> llink = cur;
next -> llink = temp;
temp -> rlink = next;
return head;
}
NODE search (NODE head, int item)
{
NODE temp, cur;
int flag = 0;
if (head -> rlink == head)
{
printf ("linked list empty\n");
```

```
return head;
}
cur = head -> rlink;
while (cur != head)
{
if (item == cur -> info)
  flag = 1;
  break;
}
cur = cur -> rlink;
}
if (cur == head)
printf("search unsuccessful\n");
if (flag == 1)
printf("search successful\n");
}
NODE delete_all_key(int item, NODE head)
{
NODE prev, cur, next;
int count;
if (head -> rlink == head)
{
printf("List is empty\n");
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("Element is not found\n");
else { printf("found at %d positions and are deleted",
```

```
return head; }}
void display (NODE head)
{
NODE temp;
if (head -> rlink == head)
{
printf (" Linked list is empty \n");
return ;}
printf (" Contents of doubly linked list :\n");
temp = head -> rlink;
while (temp != head)
{
printf("%d ", temp -> info );
temp= temp -> rlink;
}}
void main()
{
NODE head, last;
int item, choice;
head = getnode ();
head -> rlink = head;
head -> llink = head;
for (;;)
{
printf ("\n1.Insert front \n 2. Insert Rear \n 3. Delete front \n 4. Delete
rear \n 5. Insert to the left of the key element \n 6. Insert to the
right of the key element \n 7. Search \n 8. Delete duplicate items
\n 9. Display \n 10. Exit \n Enter your choice:");
scanf ("%d", &choice);
switch (choice)
{
case 1: printf (" enter the item at front end \n");
scanf("%d", &item);
last = dinsert_front (item, head)
break;
case 2: printf (" enter the item at rear end \n");
scanf(" %d", & item);
```

```
last: dinsert_rear(item,head);
break;
case 3: last= ddelete_front(head);
break;
case 4: last= ddelete_rear(head);
break;
case 5: printf("Enter the key element\n");
        scanf("%d", &item);
        last= insert_leftpos(item, head);
        break;
case 6: printf("Enter the key element\n");
        scanf("%d", &item);
        last= insert_rightpos(item,head);
        break;
case 7: printf("enter the search element:\n");
        scanf("%d", &item);
        search(head,item);
        break;
case 8: printf("Enter the element to be deleted\n");
        scanf("%d", &item);
        last= delete_all_key(item, head);
        break;
case 9: display(head);
        break;
default: exit(0);
}}}
```

```
1: Insert front
2: Insert rear
3: Delete front
4: Delete rear
5: Insert to the left of the key element
6:Insert to the right of the key element
7: Search
8: Delete duplicate terms
9: Display
10: Exit
Enter the choice
1
enter the item at front end
12

1: Insert front
2: Insert rear
3: Delete front
4: Delete rear
5: Insert to the left of the key element
6:Insert to the right of the key element
7: Search
8: Delete duplicate terms
9: Display
10: Exit
Enter the choice
2
enter the item at rear end
13

1: Insert front
2: Insert rear
3: Delete front
4: Delete rear
5: Insert to the left of the key element
6:Insert to the right of the key element
7: Search
8: Delete duplicate terms
9: Display
10: Exit
Enter the choice
1
enter the item at front end
14

1: Insert front
2: Insert rear
3: Delete front
4: Delete rear
5: Insert to the left of the key element
6:Insert to the right of the key element
7: Search
8: Delete duplicate terms
9: Display
10: Exit
Enter the choice
2
enter the item at rear end
15
```

```
1: Insert front
2: Insert rear
3: Delete front
4: Delete rear
5: Insert to the left of the key element
6:Insert to the right of the key element
7: Search
8: Delete duplicate terms
9: Display
10: Exit
Enter the choice
1
enter the item at front end
15

1: Insert front
2: Insert rear
3: Delete front
4: Delete rear
5: Insert to the left of the key element
6:Insert to the right of the key element
7: Search
8: Delete duplicate terms
9: Display
10: Exit
Enter the choice
9
contents of doubly linked list:
15  14  12  13  15

1: Insert front
2: Insert rear
3: Delete front
4: Delete rear
5: Insert to the left of the key element
6:Insert to the right of the key element
7: Search
8: Delete duplicate terms
9: Display
10: Exit
Enter the choice
3
the node deleted is 15
1: Insert front
2: Insert rear
3: Delete front
4: Delete rear
5: Insert to the left of the key element
6:Insert to the right of the key element
7: Search
8: Delete duplicate terms
9: Display
10: Exit
Enter the choice
1
enter the item at front end
15

1: Insert front
2: Insert rear
```

```
1: Insert front
2: Insert rear
3: Delete front
4: Delete rear
5: Insert to the left of the key element
6:Insert to the right of the key element
7: Search
8: Delete duplicate terms
9: Display
10: Exit
Enter the choice
5
enter the key element
13
enter towards left of 13=11

1: Insert front
2: Insert rear
3: Delete front
4: Delete rear
5: Insert to the left of the key element
6:Insert to the right of the key element
7: Search
8: Delete duplicate terms
9: Display
10: Exit
Enter the choice
9
contents of doubly linked list:
15  14  12  11  13  15

1: Insert front
2: Insert rear
3: Delete front
4: Delete rear
5: Insert to the left of the key element
6:Insert to the right of the key element
7: Search
8: Delete duplicate terms
9: Display
10: Exit
Enter the choice
7
enter the search element
14
search successful

1: Insert front
2: Insert rear
3: Delete front
4: Delete rear
5: Insert to the left of the key element
6:Insert to the right of the key element
7: Search
8: Delete duplicate terms
9: Display
10: Exit
Enter the choice
8
enter element to be deleted
```

```
1: Insert front
2: Insert rear
3: Delete front
4: Delete rear
5: Insert to the left of the key element
6:Insert to the right of the key element
7: Search
8: Delete duplicate terms
9: Display
10: Exit
Enter the choice
9
contents of doubly linked list:
15  14  12  11  13  15

1: Insert front
2: Insert rear
3: Delete front
4: Delete rear
5: Insert to the left of the key element
6:Insert to the right of the key element
7: Search
8: Delete duplicate terms
9: Display
10: Exit
Enter the choice
7
enter the search element
14
search successful

1: Insert front
2: Insert rear
3: Delete front
4: Delete rear
5: Insert to the left of the key element
6:Insert to the right of the key element
7: Search
8: Delete duplicate terms
9: Display
10: Exit
Enter the choice
8
enter element to be deleted
15
found at 2 positions and are deletedcontents of doubly linked list:

14  12  11  13

1: Insert front
2: Insert rear
3: Delete front
4: Delete rear
5: Insert to the left of the key element
6:Insert to the right of the key element
7: Search
8: Delete duplicate terms
9: Display
10: Exit
Enter the choice
```

LAB PROGRAM 8: Stacks implementation using linked list.

```
#include <stdio.h>
#include <stdlib.h>
void push();
void pop();
void display();
struct node
{
int val;
struct node * next;
};
struct node * head;
void main()
{
int choice = 0;
printf("Stacks using linked list\n");
while(choice != 4)
{
printf("\nChoose your option\n 1. Push\n 2. Pop\n 3. Display
\n 4. Exit\n Enter your choice\n");
scanf("%d", &choice);
switch(choice)
{
case 1: {push();
break; }
case 2: { pop();
break; }
case 3: { display();
break; }
case 4: { printf("Exiting"); break; }
```

```
default: { printf("Please enter a valid choice");
        } }; }}

void push()
{
int val;
struct node ptr = (struct node) malloc(sizeof(struct node));
if(ptr == NULL)
{
printf("not able to push the element");
}
else
{
printf("Enter the value");
scanf("%d", &val);
if(head == NULL)
{
 ptr -> val = val;
 ptr -> next = NULL;
 head = ptr;
 }
else
{
 ptr -> val = val;
 ptr -> next = head;
 head = ptr;
}
printf("Item pushed\n");
}}
void pop()
{
int item;
struct node *ptr;
if(head == NULL)
{
```

```
    printf("Underflow");
}
else
{
item=head->val;
ptr=head;
head=head->next;
free(ptr);
printf("item popped");
}}
void display()
{
int i;
struct node *ptr;
ptr=head;
if(ptr==NULL)
{
  printf("stack is empty\n");
}
else
{
 printf("Printing stack Elements:\n");
 while(ptr!=NULL)
 {
  printf("%d\n", ptr->val);
  ptr=ptr->next;
}}}
```

```
*********Stack operations using linked list*********

----------------------------------------------


Chose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
1
Enter the value1
Item pushed

Chose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
1
Enter the value2
Item pushed

Chose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
1
Enter the value3
Item pushed

Chose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
3
Printing Stack elements
3
2
1


Chose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
2
```

```
Enter the value2
Item pushed

Chose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
1
Enter the value3
Item pushed

Chose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
3
Printing Stack elements
3
2
1


Chose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
2
Item popped

Chose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
3
Printing Stack elements
2
1


Chose one from the below options...

1.Push
2.Pop
3.Show
4.Exit
 Enter your choice
4
Exiting....
[Program finished]
```

→ Queue implementation using linked list

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
 int data;
 struct node * next;
};
 struct node * front;
 struct node * rear;
 void insert();
 void deleteq();
 void display();
 int main()
{
  int choice;
  while (choice != 4)
  {
   printf("1. Insert an element \n 2. Delete an element \n 3.
 Display the queue \n 4. Exit \n\n Enter the choice:");
  scanf("%d", &choice);
  switch (choice)
 {
 case 1: insert();
         break;
 case 2: deleteq();
         break;
 case 3: display();
         break;
 case 4: exit(0);
         break;
 default: printf("Enter valid choice \n");
   }}}
```

```
void insert()
{
 struct node *ptr;
 int item;
 ptr = (struct node *) malloc (sizeof (struct node));
 if (ptr == NULL)
 {
  printf("Overflow");
  return;
 }
 else
 {
  printf("Enter value\n");
  scanf(" %d", &item);
  ptr -> data = item;
  if (front == NULL)
  {
   front = ptr;
   rear = ptr;
   front -> next = NULL;
   rear -> next = NULL;
  }
 else
 {
  rear -> next = ptr;
  rear = ptr;
  rear -> next = NULL;
 }}}
void deleteq()
{
 struct node *ptr;
 if(front == NULL)
 {
  printf("Underflow\n");
  return;
 }
 else
```

```
{
ptr = front;
front = front -> next;
free (ptr);
}}
void display()
{
 struct node *ptr;
 ptr = front;
 if (front == NULL)
 {
  printf (" Empty queue");
 }
 else
 {
  printf (" Printing values\n");
  while ( ptr != NULL)
  {
   printf ("\n %d \n", ptr->data);
   ptr = ptr -> next;
  }
 }
}
```

```
*************************Main Menu*****************************

=================================================================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?1

Enter value?
1

*************************Main Menu*****************************

=================================================================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?1

Enter value?
2

*************************Main Menu*****************************

=================================================================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?1

Enter value?
3

*************************Main Menu*****************************

=================================================================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?3

printing values .....

1

2

3
```

```
3

*************************Main Menu*****************************

=================================================================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?3

printing values .....

1

2

3

*************************Main Menu*****************************

=================================================================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?2

*************************Main Menu*****************************

=================================================================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?3

printing values .....

2

3

*************************Main Menu*****************************

=================================================================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit

Enter your choice ?4

[Program finished]
```

WAP to:-
a) construct a binary search tree
b) traverse the tree using all the methods, i.e; inorder, preorder, postorder.
c) Display the elements in the tree.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include<string.h>
struct node
{
int info;
struct node * llink;
struct node * rlink;
};
typedef struct node *NODE;
NODE getnode()
{
NODE x;
x= (NODE) malloc(sizeof(struct node));
if (x==NULL)
{
printf("Memory is not available\n");
exit(0);
}
return x;
}
void freenode (NODE x)
{
free(x);
}
NODE insert (int item, NODE root)
{
NODE temp, cur, prev;
char direction[10];
int i;
temp=getnode();
```

```
temp -> info = item;
temp -> llink = NULL;
temp -> rlink = NULL;
if (root == NULL)
   return temp;
printf("Give direction to insert\n");
scanf("%s", direction);
prev = NULL;
cur = root;
for (i=0; i< strlen(direction) && cur != NULL; i++)
{
prev = cur;
 if (direction[i] == 'l')
  cur = cur -> llink;
  else
  cur = cur -> rlink;
 }
 if (cur != NULL || i != strlen(direction))
 {
 printf("Insertion not possible\n");
 freenode(temp);
 return(root);
  }
  if (cur == NULL)
   {
    if(direction[i-1] == 'l')
    prev -> llink = temp;
     else
    prev -> rlink = temp;
   }
 return(root);
}
void preorder(NODE root)
{
if (root != NULL)
```

```
{
printf("The item is %d\n", root->info);
preorder(root->llink);
preorder(root->rlink);
}}
void inorder(NODE root)
{
if(root!=NULL)
{
inorder(root->llink);
printf("The item is %d\n", root->info);
inorder(root->rlink);
}}
void postorder(NODE root)
{
if(root!=NULL)
{
postorder(root->llink);
postorder(root->rlink);
printf("The item is %d\n", root->info);
}}
void display(NODE root, int i)
{
int j;
if(root!=NULL)
{
display(root->rlink, i+1);
for(j=1; j<=i; j++)
printf("  ");
printf("%d\n", root->info);
display(root->llink, i+1);
}}
void main()
{
```

```
NODE root=NULL;
int choice, i, item;
for (;;)
{
printf("1. Insert\n 2. preorder\n 3. inorder\n 4. postorder\n
5. Display\n");
printf("Enter the choice\n");
scanf("%d", &choice);
switch(choice)
{
case 1: printf("Enter the item\n");
        scanf("%d", &item);
        root = insert(item, root);
        break;
case 2: if (root==NULL)
        {
        printf("Tree is empty");
        }
        else
        {
        printf("Given tree is\n");
        display(root, 1);
        printf("The preorder traversal is\n");
        preorder(root);
        }
        break;
case 3: if (root==NULL)
        {
        printf("Tree is empty");
        }
        else
        {
        printf("Given tree is\n");
        display(root, 1);
        printf("The inorder traversal is\n");
```

```
1.insert
2.preorder
3.inorder
4.postorder
5.display
enter the choice
5
        6
    3
        7
  1
        5
    2
        4
.insert
.preorder
.inorder
.postorder
.display
nter the choice

iven tree is
        6
    3
        7
 1
        5
```

```
enter the choice
3
given tree is
        6
    3
        7
  1
        5
    2
        4
he inorder traversal is
he item is 4
he item is 2
he item is 5
he item is 1
he item is 7
he item is 3
he item is 6
.insert
.preorder
.inorder
.postorder
.display
nter the choice

iven tree is
        6
```

```
        6
    3
        7
  1
        5
    2
        4
the preorder traversal is
the item is 1
the item is 2
the item is 4
the item is 5
the item is 3
the item is 7
the item is 6
1.insert
2.preorder
3.inorder
4.postorder
5.display
enter the choice
3
given tree is
        6
    3
        7
  1
```

```
4.postorder
5.display
enter the choice
4
given tree is
        6
     3
        7
  1
        5
     2
        4
the postorder traversal is
the item is 4
the item is 5
the item is 2
the item is 7
the item is 6
the item is 3
the item is 1
1.insert
2.preorder
3.inorder
4.postorder
5.display
enter the choice
```