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

   CODE:-

```
#include<stdio.h>
int stack[10],choice,n,top,x,i;
void push();
void pop();
void display();

int main()
{
    top = -1;
    printf("\n Enter the size of STACK : ");
    scanf("%d",&n);
    printf("\nSTACK IMPLEMENTATION USING ARRAYS\n");
    do
    {
        printf("\n1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT\n");
        printf("\nEnter the choice : ");
        scanf("%d",&choice);
        switch(choice)
        {
        case 1:
            {
                push();
                break;
            }
        case 2:
            {
                pop();
                break;
            }
        case 3:
            {
                display();
                break;
            }
        case 4:
            {
                break;
            }
        default:
            {
                printf ("\nInvalid Choice\n");
            }
        }
    }
    while(choice!=4);
    return 0;
    }

    void push()
    {
        if(top >= n - 1)
        {
           printf("\nSTACK OVERFLOW\n");
        }
        else
        {
            printf("Enter a ELEMENT to be pushed : ");
            scanf("%d",&x);
        top++;
        stack[top] = x;
        }
    }

    void pop()
    {
        if(top <= -1)
        {
            printf("\nSTACK UNDERFLOW\n");
        }
        else
        {
            printf("\nThe popped ELEMENT is %d",stack[top]);
        top--;
        }
    }

    void display()
    {
        if(top >= 0)
        {
            //  Print the stack
            printf("\nELEMENTS IN THE STACK\n\n");
            for(i = top ; i >= 0 ; i--)
            printf("%d\t",stack[i]);
        }
        else
        {
            printf("\nEMPTY STACK\n");
        }
    }
```

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<string.h>
int F(char symbol)
{
    switch(symbol)
    {
        case'+':
        case'-': return 2;
        case'*':
        case'/': return 4;
        case'^':
        case'$': return 5;
        case'(': return 0;
        case'#': return -1;
        default: return 8;
    }
}
int G(char symbol)
{
    switch(symbol)
    {
        case'+':
        case'-': return 1;
        case'*':
        case'/': return 3;
        case'^':
        case'$': return 6;
        case'(': return 9;
        case')': return 0;
        default: return 7;
    }
}
void infix_postfix(char infix[], char postfix[])
{
    int top,i,j;
    char s[30],symbol;
    top=-1;
    s[++top]='#';
    j=0;
    for(i=0;i<strlen(infix);i++)
    {
        symbol=infix[i];
        while(F(s[top])>G(symbol))
        {
            postfix[j]=s[top--];
            j++;
        }
        if(F(s[top])!=G(symbol))
        s[++top]=symbol;
        else
        top--;
    }
    while(s[top]!='#')
    {
        postfix[j++]=s[top--];
    }
    postfix[j]='\0';
}
void main()
{
    char infix[20];
    char postfix[20];
    printf("Enter the valid infix expression\n");
    scanf("%s",infix);
    infix_postfix(infix,postfix);
    printf("The postfix exp is \n");
    printf("%s\n",postfix);

}
```

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:-

```c
#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;
    }
    printf("Contents of queue\n");
    for(i=front;i<=rear;i++)
    {
        printf("%d\n",q[i]);
    }
}
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);
        }
    }
}
```

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<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=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;
 case 4:exit(0);
 break;
 default:printf("Invalid choice\n");
}
}
}
```

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<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 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("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 empty cannot display items\n");
 printf("Contents of the list:\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:Display_list\n 6: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:display(first);
        break;
        case 6:exit(0);
 default:printf("Invalid choice!\n");
        break;
 }
}

}
```

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<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 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("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 empty cannot display items\n");
 printf("Contents of the list:\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:Display_list\n 6: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:display(first);
        break;
        case 6:exit(0);
 default:printf("Invalid choice!\n");
        break;
 }
}

}
```

7) WAP Implement Single Link List with following operations
a) Sort the linked list.
b) Reverse the linked list.
c) Concatenation of two linked lists

CODE:-

```
#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("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("Item deleted at rear-end is %d",cur->info);
free(cur);
prev->link=NULL;
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 delete_info(int key,NODE first)
{
NODE prev,cur;
if(first==NULL)
{
printf("List is empty\n");
return NULL;
}
if(key==first->info)
{
cur=first;
first=first->link;
freenode(cur);
return first;
}
prev=NULL;
cur=first;
while(cur!=NULL)
{
if(key==cur->info)break;
prev=cur;
cur=cur->link;
}
if(cur==NULL)
{
printf("Search is unsuccessfull\n");
return first;
}
prev->link=cur->link;
printf("Key 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");
 printf("Contents of the list:\n");
 for(temp=first;temp!=NULL;temp=temp->link)
  {
  printf("%d\n",temp->info);
  }
}
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 main()
{
int item,choice,key,n,i;
NODE first=NULL,a,b;
for(;;)
{
printf("\n1:Insert_front\n2:Delete_front\n3:Insert_rear\n4:Delete_rear\n");
printf("5:Order_list\n6:Display_list\n7:Concat\n8:Reverse\n9: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 item to be inserted in ordered_list\n");
        scanf("%d",&item);
        first=order_list(item,first);
        break;
  /*case 6:printf("Enter the key to be deleted\n");
```

```
        scanf("%d",&key);
        first=delete_info(key,first);
        break;*/
  case 6:display(first);
        break;
  case 7: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);
                 display(a);
                break;
   case 8:first=reverse(first);
                display(first);
                break;
  case 9:exit(0);
        break;
        default:printf("Invalid choice\n");
 }
}

}
```

8) WAP to implement Stack & Queues using Linked Representation

CODE:-

```
#include <stdio.h>
#include <stdlib.h>
#include <process.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 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 item\n");
        return first;
    }
    temp=first;
    temp=temp->link;
    printf("Item Deleted at the 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("The List is Empty, Cannot Delete Item\n");
        return first;
    }
    if(first->link==NULL)
    {
        printf("Item Deleted is: %d",first->info);
        free(first);
        return NULL;
    }
    prev=NULL;
    cur=first;
    while(cur->link!=NULL)
    {
        prev=cur;
        cur=cur->link;
    }
    printf("Item Deleted at the rear-end is : %d",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 Position\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 is EMPTY , Cannot Display Items\n");
    printf("\n***************************************************************\n");
    for(temp=first;temp!=NULL;temp=temp->link)
    {
        printf("%d\n",temp->info);
    }
    printf("\n***************************************************************\n");

}

void main()
{
    int item,choice,pos;
    NODE first=NULL;
    for(;;)
    {
        printf("\n1:PUSH\n2:POP\n3:insert_pos\n4:display_list\n5:Exit\n");
        printf("Enter the choice: ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:printf("Enter the item at front-end: ");
                   scanf("%d",&item);
                   first=insert_front(first,item);
                   break;
            case 2:first=delete_front(first);
                   break;
            /*case 1:printf("Enter the item at rear-end: ");
                   scanf("%d",&item);
                   first=insert_rear(first,item);
                   break;
            case 2:first=delete_rear(first);
                   break;*/
            case 3:printf("Enter the position: ");
                      scanf("%d",&pos);
                      first=insert_pos(item,pos,first);
                      break;
            case 4:display(first);
                   break;
            default:exit(0);
                   break;
        }
    }
}
```

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 *llink;
  struct node *rlink;
};
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 dinsert_front (int item, NODE head)
{
  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 ("empty\n");
      return head;
    }
  cur = head->rlink;
  next = cur->rlink;
  head->rlink = next;
  next->llink = head;
  printf ("Deleted node is %d", cur->info);
  freenode (cur);
  return head;
}

NODE ddelete_rear (NODE head)
{
  NODE cur, prev;
  if (head->rlink == head)
    {
      printf ("empty\n");
      return head;
    }
  cur = head->llink;
  prev = cur->llink;
  head->llink = prev;
  prev->rlink = head;
  printf ("Deleted node 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 was not found\n");
      return head;
    }
  prev = cur->llink;
  printf ("Enter Item 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, prev;
  if (head->llink == head)
    {
      printf ("List is Empty\n");
      return head;
    }
  cur = head->llink;
  while (cur != head)
    {
      if (item == cur->info)
       break;
      cur = cur->llink;
    }
  if (cur == head)
    {
      printf ("Key was not found\n");
      return head;
    }
  prev = cur->rlink;
  printf ("Enter Item towards Right of %d :", item);
  temp = getnode ();
  scanf ("%d", &temp->info);
  prev->llink = temp;
  temp->rlink = prev;
  cur->rlink = temp;
  temp->llink = cur;
  return head;
}

NODE delete_all_key (int item, NODE head)
{
  NODE prev, cur, next;
  int count;
  if (head->rlink == head)
    {
      printf ("List is Empty.");
      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 was not found");
  else
    printf ("Key found in %d Position(s) are Deleted\n", count);
  return head;
}
void dsearch(int item,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!=item)
        {
                cur=cur->rlink;
                count++;
        }
        if (cur==head)
        {
                printf("Search unsuccessfull\n");
        }
        else
        {
                printf("Key element found at the position %d\n",count);
        }
}
void display (NODE head)
{
  NODE temp;
  if (head->rlink == head)
    {
      printf ("List is Empty. Cannot Display Items.\n");
      return;
    }
  printf ("Contents of List:\n");
  temp = head->rlink;
  while (temp != head)
    {
      printf ("%d ", temp->info);
      temp = temp->rlink;
    }
  printf ("\n");
}

void main ()
{
  NODE head, last;
  int item, choice;
  head = getnode ();
  head->rlink = head;
  head->llink = head;
  for (;;)
    {
      printf
       ("\n1.Insert Front\n2.Insert Rear\n3.Insert Left of Node\n4.Insert Right of Node\n5.Delete Front\n6.Delete Rear\n7.Delete Duplicates\n8.Display\n9.Search\n");
      printf ("Enter Choice :");
      scanf ("%d", &choice);
      switch (choice)
       {
       case 1:
         printf ("Enter the Item to insert at Front end :");
         scanf ("%d", &item);
         last = dinsert_front (item, head);
         break;
       case 2:
         printf ("Enter the Item to insert at Rear end :");
         scanf ("%d", &item);
         last = dinsert_rear (item, head);
         break;
       case 3:
         printf ("Enter the Key Item :");
         scanf ("%d", &item);
         head = insert_leftpos (item, head);
         break;
       case 4:
         printf ("Enter the Key Item :");
         scanf ("%d", &item);
         head = insert_rightpos (item, head);
```

```
          break;
        case 5:
          last = ddelete_front (head);
          break;
        case 6:
          last = ddelete_rear (head);
          break;
        case 7:
          printf ("Enter the Key Item to be Deleted:");
          scanf ("%d", &item);
          delete_all_key (item, head);
          break;
        case 8:
          display (head);
          break;
        case 9:
            printf("Enter the key element to be searched:\n");
               scanf("%d",&item);
               dsearch(item,head);
               break;
        default:
          exit (0);
        }
}
}
```

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<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("Memory full\n");
  exit(0);
 }
 return x;
}
void freenode(NODE x)
{
free(x);
}
NODE insert(NODE root,int item)
{
NODE temp,cur,prev;
temp=getnode();
temp->rlink=NULL;
temp->llink=NULL;
temp->info=item;
if(root==NULL)
 return temp;
prev=NULL;
cur=root;
while(cur!=NULL)
{
```

```
prev=cur;
cur=(item<cur->info)?cur->llink:cur->rlink;
}
if(item<prev->info)
 prev->llink=temp;
else
 prev->rlink=temp;
return root;
}
void display(NODE root,int i)
{
int j;
if(root!=NULL)
 {
  display(root->rlink,i+1);
  for(j=0;j<i;j++)
         printf("  ");
   printf("%d\n",root->info);
        display(root->llink,i+1);
 }
}
NODE delete(NODE root,int item)
{
NODE cur,parent,q,suc;
if(root==NULL)
{
printf("Tree 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:printf("Contents of Binary Search Tree:\n");
      display(root,0);
                break;
  case 3:printf("Pre-order:\n");
      preorder(root);
                break;
  case 4:printf("Post-order:\n");
      postorder(root);
                break;
  case 5:printf("In-order:\n");
      inorder(root);
                break;
  case 6:printf("Enter the item\n");
                scanf("%d",&item);
                root=delete(root,item);
                break;
  case 7:exit(0);
  default:printf("Invalid choice\n");
          }
        }
 }
```