1.) Define a structure called Student with the members: name, reg_no, marks in 3 tests and average_ marks. Develop a menu driven program to perform the following by writing separate function for each operation: a) read information of N students b) display student's information c) to calculate the average of best two test marks of each student. Note: Illustrate the use of pointer to an array of structure and allocate memory dynamically using malloc () /calloc()/realloc().

```
#include <stdio.h>
#include <stdlib.h>
//N->Number Of Students
int N,size,i,j;
//Defining a Structure named Student
struct student{
    char name[30];
    int reg_no;
    int marks[3];
    float avg;
};
//Function Prototypes
void read(struct student *);
void disp(struct student *);
void avg(struct student *);
//Main Function
void main()
{
    //Declaring Pointer to Student Structure
    struct student *st;
    printf("Enter the size:");
    scanf("%d",&size);
    //Dynamically Allocating Memory for array of Structures
    st = (struct student *) malloc(size*sizeof(struct student));
    int choice;
    for(;;)
    {
        printf("Enter Your Choice:\n");
        printf("1.Input\n2.Display\n3.Average\n4.Exit\n");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1 : read(st);
                     break;
            case 2 : disp(st);
                     break;
            case 3 : avg(st);
                     break;
            case 4 : printf("!!THANK YOU!!\n");
                     exit(0);
            default : printf("Invalid Choice\n!!Thank You!!\n");
                      free(st); st=NULL;
        }
    }
}
//Function to read Student Details
void read(struct student *st)
{

    printf("Enter the number of Students:\n");
    scanf("%d",&N);
    if(N>size)
    {
        printf("insufficient Space\n");
        printf("Creating space by reallocating memory\n");
        st = (struct student *) realloc(st,(N-size)*sizeof(struct student));
    }
    for(i=0;i<N;i++)
    {
        printf("Enter the Name and Reg_no of Student %d\n",i+1);
        scanf("%s%d",(st+i)->name,&(st+i)->reg_no);
        printf("Enter the marks Scored in three tests:\n");
        for(j=0;j<3;j++)
        {
            scanf("%d",&(st+i)->marks[j]);
        }
    }
}
//Function to display Student Details
void disp(struct student *st)
{
    if(N==0)
    {
        printf("List is Empty!");
        return;
    }
    printf("The details of %d students are:\n",N);
    printf("Reg-No\tName\tTest1\tTest2\tTest3\t\n");
    for(i=0;i<N;i++)
    {
        printf("%d\t%s\t",(st+i)->reg_no,(st+i)->name);
        for(j=0;j<3;j++)
        {
            printf("%d\t",(st+i)->marks[j]);
        }
        printf("\n");
    }
}
//Function to calculate average of Students and display
void avg(struct student *st)
{
    int k,temp,sum[N];
    //temp : variable used for swapping
    //sum[N] : Array to Store Total marks of N students
    for(i=0;i<N;i++)
    {
        for(j=0;j<3;j++)
        {
            for(k=0;k<3-j-1;k++)
            {
                if((st+i)->marks[k]>(st+i)->marks[k+1])
                {
                    temp = (st+i)->marks[k];
                    (st+i)->marks[k] = (st+i)->marks[k+1];
                    (st+i)->marks[k+1] = temp;
                }
            }
        }
    }
    for(i=0;i<N;i++)
    {
        sum[i] = (st+i)->marks[1] + (st+i)->marks[2];
        (st+i)->avg = (float)sum[i]/2;
    }
    printf("The details of %d students are:\n",N);
    printf("Reg-No\tName\t\tTest1\tTest2\tTest3\tAverage\n");
    for(i=0;i<N;i++)
    {
        printf("%d\t%s\t\t",(st+i)->reg_no,(st+i)->name);
        for(j=0;j<3;j++)
        {
            printf("%d\t",(st+i)->marks[j]);
        }
        printf("%.2f",(st+i)->avg);
        printf("\n");
    }
}
```

Output :

```
Enter the size:2
Enter Your Choice:
1.Input
2.Display
3.Average
4.Exit
1
Enter the number of Students:
2
```

```
Enter the Name and Reg_no of Student 1
Vinod
26
Enter the marks Scored in three tests:
20
17
18
Enter the Name and Reg_no of Student 2
Reddy
41
Enter the marks Scored in three tests:
18
19
20
Enter Your Choice:
1.Input
2.Display
3.Average
4.Exit
2
The details of 2 students are:
Reg-No  Name    Test1   Test2   Test3
26      Vinod   20      17      18
41      Reddy   18      19      20
Enter Your Choice:
1.Input
2.Display
3.Average
4.Exit
3
The details of 2 students are:
Reg-No  Name          Test1   Test2   Test3   Average
26      Vinod         17      18      20      19.00
41      Reddy         18      19      20      19.50
Enter Your Choice:
1.Input
2.Display
3.Average
4.Exit
4
 !!THANK YOU!!
```

2.) Define a structure called Time containing 3 integer members (hour, minute, second). Develop a menu driven program to perform the following by writing separate function for each operation. a) To read time b) To display time c) To Update time d) Add two times by writing Add (T1, T2) which returns the new Time. Update function increments the time by one second and returns the new time (if the increment results in 60 seconds, then the second member is set to zero and minute member is incremented by one. If the result is 60 minutes, the minute member is set to zero and the hour member is incremented by one. Finally, when the hour becomes 24, Time should be reset to zero) Note: Illustrate the use of pointer to a structure variable and passing and returning of structure type to and from the function (both by value and reference).

```
#include <stdio.h>
#include <stdlib.h>
//TIME Structure containing 3 members
struct TIME {
   int hr,min,sec;
};
//Type Defining struct TIME
typedef struct TIME Time;
//Function Prototypes
void read(Time *);
void disp(Time *);
void update(Time *);
void add(Time *, Time *, Time *);
//Main Function
void main()
{
   int choice;
   //Declaring 4 Pointers to Structure TIME
   Time T1,T2,T3,T4;
   for(;;)
   {
      printf("\nEnter your Choice : \n");
      printf("1.Read\n2.Update\n3.Add\n4.Exit\n");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1 : printf("Reading T1..\n");
                  read(&T1);
                  printf("Reading T2..\n");
                  read(&T2);
                  printf("The Time you entered are :\n");
                  printf("T1 = ");
                  disp(&T1);
                  printf("\nT2 = ");
                  disp(&T2);
```

```
                printf("\n");
                break;
            case 2 : printf("Enter the Time to be Updated\n");
                read(&T3);
                printf("The time before Updation = ");
                disp(&T3);
                update(&T3);
                printf("\nThe time before Updation = ");
                disp(&T3);
                break;
            case 3 : printf("Adding Two Times:\n");
                printf("T1 = ");
                disp(&T1);
                printf("\nT2 = ");
                disp(&T2);
                add(&T1, &T2, &T4);
                printf("\nThe added time is = ");
                disp(&T4);
                break;
            case 4 : printf("\nExecution Successfull\n!!THANK YOU!!\n");
                exit(0);
            default : printf("\nInvalid Choice\nEnter Again\n");
        }
    }
}
//Function to read Time
void read(Time *T)
{
    printf("Enter the time in Hours:Minutes:Seconds : ");
    scanf("%d%d%d",&T->hr,&T->min,&T->sec);
}
//Function to display Time
void disp(Time *T)
{
    printf("%d:%d:%d",T->hr,T->min,T->sec);
}
//Function to Update Time by 1 second
void update(Time *T)
{
    T->sec++;
    if(T->sec>=60)
    {
        T->min++;
        T->sec=T->sec%60;
        if(T->min>=60)
        {
            T->hr++;
            T->min = T->min % 60;
            if(T->hr>=24)
                T->hr=T->min=T->sec=0;
        }
    }
}
//Function to add 2 Times
//T3 stores the added time
void add(Time *T1,Time *T2,Time *T3)
{
    T3->hr=T3->min=T3->sec=0;
    T3->hr = T1->hr + T2->hr;
    T3->min = T1->min + T2->min;
    T3->sec = T1->sec + T2->sec;
    if(T3->sec>=60)
    {
        T3->min++;
        T3->sec=T3->sec%60;
        if(T3->min>=60)
        {
            T3->hr++;
            T3->min = T3->min % 60;
            if(T3->hr>=24)
                T3->hr=T3->min=T3->sec=0;
        }
    }
    if(T3->min>=60)
    {
        T3->hr++;
        T3->min = T3->min % 60;
        if(T3->hr>=24)
            T3->hr=T3->min=T3->sec=0;
    }
    if(T3->hr>=24)
        T3->hr=T3->min=T3->sec=0;
}
```

Output :-

```
Enter your Choice :
1.Read
2.Update
3.Add
4.Exit
1
Reading T1..
Enter the time in Hours:Minutes:Seconds : 15 24 18
Reading T2..
Enter the time in Hours:Minutes:Seconds : 08 14 25
The Time you entered are :
T1 = 15:24:18
T2 = 8:14:25
```

```
Enter your Choice :
1.Read
2.Update
3.Add
4.Exit
2
Enter the Time to be Updated
Enter the time in Hours:Minutes:Seconds : 21 25 48
The time before Updation = 21:25:48
The time before Updation = 21:25:49
Enter your Choice :
1.Read
2.Update
3.Add
4.Exit
3
Adding Two Times:
T1 = 15:24:18
T2 = 8:14:25
The added time is = 23:38:43
Enter your Choice :
1.Read
2.Update
3.Add
4.Exit
4
Execution Successfull
!!THANK YOU!!
```

3.) Develop a menu driven program to implement the following operations on array storage representation with static and dynamic memory allocation. i)Insert by position ii) Delete by position iii)Insert by key iv) Delete by key v) Insert by order vi) Search by key vii) Search by position viii) Reverse the contents.

```
//Static memory allocation
#include<stdio.h>
#include<stdlib.h>
int size=20; //Initializing Size Of Array
int ne = 0; //Initializing Number of Elements in Array
//Function to read an Array
void read (int x[])
{
   int i,n;
   printf("Enter the number of elements \n");
   scanf("%d", &n);
   printf("Enter the elements into the array \n");
   for (i=0; i<n;i++)
   scanf("%d",&x[i]);
   ne=n;
}
//Function to display an Array
void disp (int x[])
{
   int i;
   if (ne==0)
   {
      printf("Array empty\n");
      return;

   }
   printf("The array elements you entered are : ");
   for(i=0;i<ne;i++)
   printf("%d ",x[i]);
   printf("\n");
}
//Function to insert an element into Array at specified position
void insert_by_pos (int x[])
{
   int i, pos, ele;
   if(ne==size)
   {
      printf ("Array is full\n");
      return;
   }
   printf("Enter the element to be inserted : \n");
   scanf("%d", &ele);
   printf("Enter the position where the element should be inserted\n");
   scanf("%d", &pos);
   if(pos>=1&&pos<=ne+1)
   {
        for(i=ne;i>=pos;i--)
        {
           x[i]=x[i-1];
        }
        x[i]=ele;
        ne++;
    }
    else
    printf("Invalid position\n");
}
//Function to insert an element into Array in order
void insert_by_order (int x[])
{
    int i, ele;
    if (ne== size)
    {
        printf("Array is full....:(\n");
        return;
    }
    printf("Enter the element to be inserted...\n");
    scanf("%d",&ele);
    i=ne-1;
    while (i>=0&&x[i]>ele)
    {
        x[i+ 1] = x[i];
        i--;
    }
    x[i+1]=ele;
    ne++;
}
//Function to delete an element into Array at specified position
void delete_by_pos (int x[])
{
    int i, pos;
    if (ne ==0)
    {
        printf("Array is empty\n");
        return;
    }
    printf ("Enter the postion from where element should be deleted\n");
    scanf("%d", & pos);
    if (pos>= 1 && pos<=ne)
    {
    for(i=pos-1;i<ne-1;i++)
    {
        x[i] = x[i+1];
    }
    ne--;
    }
    else
    printf("Invalid position\n");
}
//Function to delete a key element into Array
void delete_by_ele (int x[size])
{
    int i, ele;
    if (ne ==0)
    {
        printf ("Array is empty\n");
        return;
    }
    printf ("Enter the element that should be deleted\n");
    scanf("%d",&ele);
    for(i=0;i<ne&&x[i]!=ele;i++);
    if (i==ne)
    {
        printf ("Element not found in the array list\n");
        return;
    }
    for (;i<ne-1;i++)
    {
        x[i]=x[i+1];
    }
    ne--;
}
//Function to search an element in Array
void search_by_key(int x[])
{
    int ele,i;
    if(ne==0)
    {
        printf("Array is empty\n");
        return;
    }
    printf("Enter the element that should be searched\n");
    scanf("%d",&ele);
    for(i=0;i<ne;i++)
    {
        if(x[i]==ele)
        printf("Element found at position %d\n",i+1);
    }
    printf("Element not found\n");
}
//Function to search an element in Array at specified position
void search_by_pos(int x[])
{
    int pos,i;
    if(ne==0)
    {
        printf("Array is empty\n");
        return;
    }
    printf("Enter the position from where element should be searched\n");
    scanf("%d",&pos);
    if(pos>=1&&pos<=ne)
    {
        i=pos-1;
        printf("Element at %d is %d\n",pos,x[i]);
    }
    else
    printf("Invalid position\n");
}
//Function to reverse the elements of Array
void reverse(int x[])
{
     int t,i;
    if(ne==0)
    {
        printf("Array is empty\n");
        return;
    }
    for(i=0;i<ne/2;i++)
    {
        t=x[i];
        x[i]=x[ne-i-1];
        x[ne-i-1]=t;
    }
}
//Main Function
void main()
{
    int a[size];//Array declaration
    int choice;
    printf("Menu\n1.Read\n2.Display\n3.Insert by Position\n4.Insert by Order\n5.Delete by Position\n6.Delete
by Element\n7.Search by Key\n8.Search by Position\n9.Reverse\n10.Exit\n");
    for (;;)
    {
        printf("Enter Your Choice : ");
        scanf("%d",&choice);
        switch (choice)
        {
            case 1: read(a);
                    disp(a);
                    break;
            case 2: disp(a);
                    break;
            case 3: insert_by_pos(a);
                    disp(a);
                    break;
            case 4: insert_by_order(a);
```

```
                disp(a);
                break;
            case 5: delete_by_pos(a);
                disp(a);
                break;
            case 6: delete_by_ele(a);
                disp(a);
                break;
            case 7: search_by_key(a);
                break;
            case 8: search_by_pos(a);
                break;
            case 9: reverse(a);
                disp(a);
                break;
            case 10 : printf("!!THANK YOU!!\n");
                  exit(0);
            default: printf("Invalid Choice\n");
        }
    }
}
```

Output:

Menu

```
1.Read
2.Display
3.Insert by Position
4.Insert by Order
5.Delete by Position
6.Delete by Element
7.Search by Key
8.Search by Position
9.Reverse
10.Exit
Enter Your Choice : 1
Enter the number of elements
5
Enter the elements into the array
1 2 3 4 5
The array elements you entered are : 1 2 3 4 5
Enter Your Choice : 3
Enter the element to be inserted :
6
Enter the position where the element should be inserted
6
The array elements you entered are : 1 2 3 4 5 6
Enter Your Choice : 5
Enter the postion from where element should be deleted
3
The array elements you entered are : 1 2 4 5 6
Enter Your Choice : 6
Enter the element that should be deleted
6
The array elements you entered are : 1 2 4 5
Enter Your Choice : 7
Enter the element that should be searched : 2
Element found at position 2
Enter Your Choice : 8
Enter the position from where element should be searched : 3
Element at 3 is 4
Enter Your Choice : 9
The array elements you entered are : 5 4 2 1
Enter Your Choice : 10
!!THANK YOU!!
```

```
//Dynamic memory allocation
#include<stdio.h>
#include<stdlib.h>
```

```
int ne=0; //Initializing Number of Elements in Array
//Function to read an Array
void read(int *x)
{
   int i,n;
   printf("Enter The number of elements\n");
   scanf("%d",&n);
   printf("Enter The elements into The Array\n");
   for(i=0;i<n;i++)
   scanf("%d",(x+i));
   ne=n;
}
//Function to display elements of an Array
void disp(int *x)
{
   int i;
   if(ne==0)
   {
      printf("Array empty\n");
      return;
   }
   printf("The Array elements you Entered are : \n");
   for(i=0;i<ne;i++)
   printf("%d\t",*(x+i));
   printf("\n");
}
//Function to insert an element into Array at specified position
void insert_by_pos(int *x,int size)
{
   int i,pos,ele;
   if(ne==size)
   {
      printf("Array is full.\n");
      printf("Reallocating Array\n");
      size++;
      x=(int*)realloc(x,size*sizeof(int));
   }
   printf("Enter The element to be inserted : \n");
   scanf("%d",&ele);
   printf("Enter The position where The elements should be inserted\n");
   scanf("%d",&pos);
   if(pos>=1&&pos<=ne+1)
   {
      for(i=ne;i>=pos;i++)
      {
         *(x+i)=*(x+i-1);
      }
      *(x+i)=ele;
      ne++;
   }
```

```
    else
    printf("invalid position\n");
}
//Function to insert an element into Array in order
void insert_by_order(int *x,int size)
{
    int i,ele;
    if(ne==size)
    {
        printf("Array is full.\n");
        printf("Reallocating Array \n");
        size++;
        x=(int*)realloc(x,size*sizeof(int));
    }
    printf("Enter The element to be inserted : ");
    scanf("%d",&ele);
    i=ne-1;
    while(i>=0&&*(x+i)>ele)
    {
        *(x+i+1)=*(x+i);
        i--;
    }
    *(x+i+1)=ele;
    ne++;
}
//Function to delete an element in Array at specified position
void delete_by_pos(int *x)
{
    int i,pos;
    if(ne==0)
    {
        printf("Array is empty.\n");
        return;
    }
    printf("Enter The position from where elements should be deleted\n");
    scanf("%d",&pos);
    if(pos>=1&&pos<=ne)
    {
        for(i=pos-1;i<ne-1;i++)
        {
            *(x+i)=*(x+i+1);
        }
        ne--;
    }
    else
    printf("invalid position.\n");
}
//Function to delete a key element into Array
void delete_by_ele(int *x)
{
    int i,ele;
    if(ne==0)
    {
        printf("Array is empty.\n");
        return;
    }
    printf("Enter The elements that should be deleted\n");
    scanf("%d",&ele);
    for(i=0;i<ne&&*(x+i)!=ele;i++);
    if(i==ne)
    {
        printf("element not found in The Array list\n");
        return;
    }
    for(;i<ne-1;i++)
    {
        *(x+i)=*(x+i+1);
    }
    ne--;
}
//Function to search an element in Array
void search_by_key(int *x)
{
    int i,ele,f=0;
    if(ne==0)
    {
        printf("Array is empty\n");
        return;
    }
    printf("Enter The element that should be searched\n");
    scanf("%d",&ele);
    for(i=0;i<ne;i++)
    {
        if(*(x+i)==ele)
        {
            printf("element found at position %d\n",i+1);
            f=1;
            break;
        }
    }
        if(f==0)
        printf("element not found\n");
}
//Function to search an element in Array at specified position
void search_by_pos(int *x)
{
    int pos,i;
    if(ne==0)
    {
        printf("Array is empty\n");
        return;
    }
    printf("Enter The position from where element should be searched\n");
    scanf("%d",&pos);
    if(pos>=1&&pos<=ne)
    {
        i=pos-1;
        printf("element at %d is %d\n",pos,*(x+i));
    }
    else
    printf("invalid position\n");
}
//Function to reverse The elements of Array
void reverse(int *x)
{
    int t,i;
    if(ne==0)
    {
        printf("Array is empty\n");
        return;
    }
    for(i=0;i<ne/2;i++)
    {
        t=*(x+i);
        *(x+i)=*(x+ne-i-1);
        *(x+ne-i-1)=t;
    }
}
void main()
{
    int *a; //Base Address of Array
    int choice,size;
    printf("Enter The size\n");
    scanf("%d",&size);
    a=(int *)malloc(size*sizeof(int)); //Dynamically Allocating memory for Array
    printf("Menu\n1.Read\n2.Display\n3.Insert by position\n4.Insert by order\n5.Delete by position\n6.Delete by
element\n7.Search by key\n8.Search by position\n9.Reverse\n10.Exit\n");
    for(;;)
    {
        printf("Enter Your Choice : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: read(a);
                    disp(a);
                    break;
            case 2: disp(a);
                    break;
            case 3: insert_by_pos(a,size);
                    disp(a);
```

```
                break;
            case 4: insert_by_order(a,size);
                disp(a);
                break;
            case 5: delete_by_pos(a);
                disp(a);
                break;
            case 6: delete_by_ele(a);
                disp(a);
                break;
            case 7: search_by_key(a);
                break;
            case 8: search_by_pos(a);
                break;
            case 9: reverse(a);
                disp(a);
                break;
            case 10 : free(a);
                 a=NULL;
                 exit(0);
            default: printf("Invalid Choice\n");
        }
    }
}
```

Output:

```
Enter The size
2
```

```
Menu
1.Read
2.Display
3.Insert by position
4.Insert by order
5.Delete by position
6.Delete by element
7.Search by key
8.Search by position
9.Reverse
10.Exit
Enter Your Choice : 1
Enter The number of elements
4
Enter The elements into The Array
1 3 4 5
The Array elements you Entered are : 1 3 4 5
Enter Your Choice : 3
Enter The element to be inserted :
6
Enter The position where The elements should be inserted
5
The Array elements you Entered are : 1 3 4 5 6
Enter Your Choice : 4
Enter The element to be inserted : 2
The Array elements you Entered are : 1 2 3 4 5 6
Enter Your Choice : 5
Enter The position from where elements should be deleted
2
The Array elements you Entered are : 1 3 4 5 6
Enter Your Choice : 6
Enter The elements that should be deleted
4
The Array elements you Entered are : 1 3 5 6
Enter Your Choice : 7
Enter The element that should be searched
5
Element found at position 3
Enter Your Choice : 9
The Array elements you Entered are : 6 5 3 1
Enter Your Choice : 10
```

4.) Develop a menu driven program to implement singly linked list with various operations such as

  i. **Insertion and Deletion at front/rear**
  ii. **Insertion and Deletion at the specified position**

iii. **Delete by Key**
iv. **Search by key**
v. **Create an ordered list**
vi. **Reverse a list**
vii. **Creating a copy of the list**

```
#include <stdio.h>
#include <stdlib.h>
//Structure for a single Node
struct node
{
   int item;
   struct node *link; //Pointer to next Node
};
//struct node * type defined as NODE
typedef struct node * NODE;
//Function to create a Node
NODE getnode(int val)
{
   NODE temp; //Pointer to newly created node
   temp = (NODE)malloc(sizeof(struct node));
   temp -> item = val;
   temp-> link = NULL;
   return temp;
}
//Function to insert Node at front
NODE insertfront(NODE first, int val)
{
   NODE temp; //Pointer to newly created node
   temp = getnode(val);
   if(first == NULL)
   {
      first = temp;
      return first;
   }
   else
   {
      temp -> link = first;
      return temp;
   }
}
//Function to insert Node at Rear End
NODE insertrear(NODE first, int val)
{
   NODE temp = getnode(val); //Pointer to newly created node
   NODE cur = first; //Pointer pointing to current node
   if(first == NULL)
   {
      first = temp;
```

```
        return first;
    }
    else
    {
        while(cur->link!=NULL)
        {
            cur = cur->link;
        }
        cur->link = temp;
        return first;
    }
}
//Function to delete Node at front
NODE deletefront(NODE first)
{
    if(first==NULL)
    {
        printf("List is empty\n");
        return first;
    }
    else
    {
        first = first->link;
        return first;
    }
}
//Function to delete Node at Rear End
NODE deleterear(NODE first)
{
    NODE cur = first; //Pointer pointing to current node
    NODE prev=NULL; //Pointer pointing to previous node
    if(first==NULL)
    {
        printf("List is empty\n");
        return first;
    }
    else
    {
        while(cur->link!=NULL)
        {
            prev = cur;
            cur = cur->link;
        }
        prev->link = NULL;
        return first;
    }
}
//Function to insert Node at Specified Position
NODE insertpos(NODE first, int val, int pos)
{
    NODE temp,cur,prev=NULL;
    temp = getnode(val);
    cur = first;
    if(pos==1&&first==NULL)
    {
        first = insertfront(first,val);
        return first;
    }
    else if(pos!=1&&first==NULL)
    {
        printf("Invalid Position\n");
        return first;
    }
    else
    {
        while(pos!=1)
        {
            prev=cur;
            cur = cur->link;
            pos--;
        }
        temp->link = cur;
        prev->link = temp;
        printf("Insertion Succesfull\n");
        return first;
    }
}
//Function to delete a Node at Specified Position
NODE deletepos(NODE first,int pos)
{
    NODE cur,prev;
    cur = first;
    prev = NULL;
    while(pos != 1)
    {
        prev = cur;
        cur = cur->link;
        pos--;
    }
    prev->link = cur->link;
    free(cur);
    cur = NULL; //Deleting the Current Node by Deallocating Memory
    printf("Deletion Succesfull\n");
    return first;
}
//Function to search a Key element in a list
int search(NODE first, int key)
{
    int i=1,flag=0,pos=0;
    NODE cur;
    cur = first;
    while(cur->link!=NULL)
    {
        if(cur->item==key)
        {
            flag = 1;
            pos = i;
            break;
        }
        cur = cur->link;
        i++;
    }
    if(flag==1)
    return pos;
    else
    return 0;
}
//Function to delete a Key element in a list
NODE deletebykey(NODE first,int key)
{
    int pos = search(first,key);
    NODE prev,cur;
    prev=NULL;cur=first;
    while(pos != 1)
    {
        prev = cur;
        cur = cur->link;
        pos--;
    }
    prev->link = cur->link;
    free(cur);
    cur = NULL;
    printf("Deletion Successfull\n");
    return first;
}
//Function to display all elements in a list
void display(NODE first)
{
    NODE cur=first;
    if(first==NULL)
    {
        printf("List is Empty\n");
    }
    else
    {
        while (cur!=NULL)
        {
            printf("%d ",cur->item);
            cur = cur->link;
        }
    }
}
//Function to reverse a linked list
NODE reverse(NODE first)
{
    NODE temp,cur,prev;
    temp = first;
    cur=prev=NULL;
    while(temp != NULL)
    {
        cur = temp->link;
        temp->link = prev;
        prev = temp;
        temp = cur;
    }
    first = prev;
    display(first);
    return first;
}
//Function to create Ordered List
NODE orderlist(NODE first, int val)
{
    NODE temp = getnode(val);
    NODE cur = first,prev = NULL;
    while(cur->item<temp->item)
    {
        prev = cur;
        cur = cur->link;
    }
    temp->link = cur;
    prev->link = temp;
    return first;
}
//Function to create a Copy List
NODE createcopy(NODE first, NODE copy)
{
    NODE cur=first;
    while(cur!=NULL)
    {
        copy=insertrear(copy,cur->item);
        cur=cur->link;
    }
    return copy;
}
void main()
{
    NODE first = NULL; // Pointer to first node of list
    NODE copy = NULL; // Poiner to first node of copied list
    int choice;
    int key,val,pos;
    printf("\nMain Menu\n1.Insert Front\n2.Insert Rear\n3.Delete Front\n4.Delete Rear\n5.Insert By
Pos\n6.Delete by Pos\n7.Search\n8.Delete By Key\n9.Display\n10.Insert By Order\n11.Create
Copy\n12.Reverse\n13.Exit\n");
    for(;;)
    {
        printf("\nEnter your Choice :");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1 : printf("Enter the value : ");
                     scanf("%d",&val);
                     first = insertfront(first,val);
                     break;
            case 2 : printf("Enter the value : ");
                     scanf("%d",&val);
                     first = insertrear(first,val);
                     break;
            case 3 : first = deletefront(first);
                     printf("Deletion Successfull\n");
                     break;
            case 4 : first = deleterear(first);
                     printf("Deletion Successfull\n");
                     break;
            case 5 : printf("Enter the value : ");
                     scanf("%d",&val);
                     printf("Enter the position : ");
                     scanf("%d",&pos);
                     first = insertpos(first,val,pos);
                     break;
            case 6 : printf("Enter the position : ");
                     scanf("%d",&pos);
                     first = deletepos(first,pos);
                     break;
            case 7 : printf("Enter the Key : ");
                     scanf("%d",&key);
                     pos = search(first,key);
                     if(pos!=0)
                     printf("The Key found at %d pos\n",pos);
                     else
                     printf("Element not found\n");
                     break;
            case 8 : printf("Enter the Key : ");
                     scanf("%d",&key);
                     first = deletebykey(first,key);
                     break;
            case 9 : display(first);
                     break;
            case 10 :printf("Enter the value : ");
                     scanf("%d",&val);
                     first = orderlist(first,val);
```

```
                    break;
                case 11 : copy = createcopy(first,copy);
                    printf("Copied List : ");
                    display(copy);
                    break;
                case 12 : first = reverse(first);
                     break;
                case 13 : printf("!!THANK YOU!!\n");
                    exit(0);
                default : printf("Invalid Choice\n Enter again\n");
            }
        }
}
```

Output:

```
Main Menu
1.Insert Front
2.Insert Rear
3.Delete Front
4.Delete Rear
5.Insert By Pos
6.Delete by Pos
7.Search
8.Delete By Key
9.Display
10.Insert By Order
11.Create Copy
12.Reverse
13.Exit

Enter your Choice :1
Enter the value : 10

Enter your Choice :2
Enter the value : 20

Enter your Choice :2
Enter the value : 30

Enter your Choice :2
Enter the value : 40

Enter your Choice :9
10 20 30 40

Enter your Choice :10
Enter the value : 15

Enter your Choice :9
10 15 20 30 40
```

```
Enter your Choice :3
Deletion Successful

Enter your Choice :4
Deletion Successful

Enter your Choice :9
15 20 30

Enter your Choice :6
Enter the position : 2
Deletion Successful

Enter your Choice :9
15 30

Enter your Choice :1
Enter the value : 10

Enter your Choice :11
Copied List : 10 15 30

Enter your Choice :12
30 15 10

Enter your Choice :13
!!THANK YOU!!
```

5.) Develop a menu driven program to implement Circular singly linked list with various operations such as

i. **Insertion and Deletion at front/rear**

ii. **Insertion and Deletion at the specified position**
iii. **Delete by Key**
iv. **Search by key**
v. **Create an ordered list**
vi. **Reverse a list**
vii. **Creating a copy of the list**

```
#include <stdio.h>
#include <stdlib.h>
//Structure for a single Node
struct NODE
{
   int item;
   struct NODE *link; //Pointer to next Node
};
typedef struct NODE * NODE;
//Function to create a Node
NODE getnode(int val)
{
   NODE temp; //Pointer to newly created node
   temp = (NODE)malloc(sizeof(struct NODE));
   temp -> item = val;
   temp-> link = temp;
   return temp;
}
//Function to insert Node at front
NODE insertfront(NODE last, int val)
{
   NODE temp;
   temp = getnode(val); //Pointer to newly created node
   if(last == NULL)
   {
      return temp;
   }
   else
   {
      temp->link=last->link;
      last->link=temp;
      return last;
   }
}
//Function to insert Node at Rear End
NODE insertrear(NODE last, int val)
{
   NODE temp;
   temp = getnode(val); //Pointer to newly created node
   if(last == NULL)
   {
      return temp;
   }
```

```
    else
    {
        temp->link=last->link;
        last->link=temp;
        return temp;
    }
}
//Function to delete Node at front
NODE deletefront(NODE last)
{
    NODE cur = last->link;
    if(last==NULL)
    {
        printf("List is empty\n");
        return last;
    }
    else
    {
        last->link=cur->link;
        free(cur);
        return last;
    }
}
//Function to delete Node at Rear End
NODE deleterear(NODE last)
{
    NODE cur=last->link;
    if(last==NULL)
    {
        printf("List is empty\n");
        return last;
    }
    else
    {

        while(cur->link!=last)
        {
            cur = cur->link;
        }
        cur->link = last->link;
        free(last);
        return cur;
    }
}
//Function to insert Node at Specified Position
NODE insertpos(NODE last, int val, int pos)
{
    NODE temp = getnode(val);
    NODE cur = last->link; //Pointer pointing to current node
    NODE prev=NULL;//Pointer pointing to previous node
    if(pos==1&&last==NULL)
    {
        last = insertfront(last,val);
        return last;
    }
    else if(pos!=1&&last==NULL)
    {
        printf("Invalid Position\n");
        return last;
    }
    else
    {
        while(pos!=1)
        {
            prev=cur;
            cur = cur->link;
            pos--;
        }
        temp->link = cur;
        prev->link = temp;
        printf("Insertion Succesfull\n");
        return last;
    }
}
//Function to delete a Node at Specified Position
NODE deletepos(NODE last,int pos)
{
    NODE cur,prev;
    cur = last->link; //Pointer pointing to current node
    prev = NULL; //Pointer pointing to previous node
    if(last==NULL)
    {
        printf("List Empty\n");
        return last;
    }
    while(pos != 1)
    {
        prev = cur;
        cur = cur->link;
        pos--;
    }
    prev->link = cur->link;
    free(cur); //Deleting the Current Node by Deallocating Memory
    cur = NULL;
    return last;
}
//Function to search a Key element in a list
int search(NODE last, int key)
{
    int i=1,flag=0,pos=0;
    NODE cur;
    cur = last->link;
    while(cur->link!=last->link)
    {
        if(cur->item==key)
        {
            flag = 1;
            pos = i;
            break;
        }
        cur = cur->link;
        i++;
    }
    if(flag==1)
    return pos;
    else
    return 0;
}
//Function to delete a Key element in a list
NODE deletebykey(NODE last,int key)
{
    int pos = search(last,key);
    NODE prev,cur;
    prev=NULL;cur=last->link;
    while(pos != 1)
    {
        prev = cur;
        cur = cur->link;
        pos--;
    }
    prev->link = cur->link;
    free(cur);
    cur = NULL;
    return last;
}
//Function to display all elements in a list
void display(NODE last)
{
    NODE cur=last->link;
    if(last==NULL)
    {
        printf("List is Empty\n");
    }
    else
    {
        do
        {
            printf("%d ",cur->item);
            cur = cur->link;
        }while (cur!=last->link);
```

```
   }
}
//Function to create Ordered List
NODE orderlist(NODE last, int val)
{
   NODE temp = getnode(val);
   NODE cur = last->link,prev = NULL;
   while(cur->item<temp->item)
   {
      prev = cur;
      cur = cur->link;
   }
   temp->link = cur;
   prev->link = temp;
   return last;
}
//Function to reverse a linked list
NODE reverse(NODE last)
{
   NODE cur=last,prev;
   if(last==NULL)
   return NULL;
   if(last==last->link)
   return last;
   NODE temp=last->link;
   NODE first=temp;
   while(temp!=last)
   {
      prev=temp->link;
      temp->link=cur;
      cur=temp;
      temp=prev;
   }
   temp->link=cur;
   return first;
}
//Function to create a Copy List
NODE createcopy(NODE last,NODE copy)
{
   NODE cur=last->link;
   do
   {
      copy=insertrear(copy,cur->item);
      cur=cur->link;
   }while(cur!=last->link);
   return copy;
}
void main()
{
   NODE last = NULL; // Pointer to last node of list
   NODE copy = NULL; // Poiner to first node of copied list
   int choice;
   int key,val,pos;
   printf("\nMain Menu\n1.Insert Front\n2.Insert Rear\n3.Delete Front\n4.Delete Rear\n5.Insert By
Pos\n6.Delete by Pos\n7.Search\n8.Delete By Key\n9.Display\n10.Insert By Order\n11.Reverse\n12.Create
Copy\n13.Exit\n");
   for(;;)
   {
      printf("\nEnter your Choice :");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1 : printf("Enter the value : ");
                  scanf("%d",&val);
                  last = insertfront(last,val);
                  break;
         case 2 : printf("Enter the value : ");
                  scanf("%d",&val);
                  last = insertrear(last,val);
                  break;
         case 3 : last = deletefront(last);
                  printf("Deletion Successfull\n");
                  break;
         case 4 : last = deleterear(last);
                  printf("Deletion Successfull\n");
                  break;
         case 5 : printf("Enter the value : ");
                  scanf("%d",&val);
                  printf("Enter the position : ");
                  scanf("%d",&pos);
                  last = insertpos(last,val,pos);
                  printf("Insertion Succesfull\n");
                  break;
         case 6 : printf("Enter the position : ");
                  scanf("%d",&pos);
                  last = deletepos(last,pos);
                  printf("Deletion Succesfull\n");
                  break;
         case 7 : printf("Enter the Key : ");
                  scanf("%d",&key);
                  pos = search(last,key);
                  if(pos!=0)
                  printf("The Key found at %d pos\n",pos);
                  else
                  printf("Element not found\n");
                  break;
         case 8 : printf("Enter the Key : ");
                  scanf("%d",&key);
                  last = deletebykey(last,key);
                  printf("Deletion Successfull\n");
```

```
                    break;
            case 9 : printf("The List is : ");
                    display(last);
                    break;
            case 10 : printf("Enter the Key : ");
                     scanf("%d",&key);
                     last = orderlist(last,key);
                     break;
            case 11 : last = reverse(last);
                     display(last);
                     break;
            case 12 : printf("Copied List : ");
                     copy = createcopy(last,copy);
                     display(copy);
                     break;
            case 13 : printf("!!THANK YOU!!\n");
                    exit(0);
            default : printf("Invalid Choice\n Enter again\n");
        }
    }
}
```

Output:

```
Main Menu
1.Insert Front
2.Insert Rear
3.Delete Front
4.Delete Rear
5.Insert By Pos
6.Delete by Pos
7.Search
8.Delete By Key
9.Display
10.Insert By Order
11.Create Copy
12.Reverse
13.Exit

Enter your Choice :1
Enter the value : 100

Enter your Choice :2
Enter the value : 200

Enter your Choice :2
Enter the value : 300

Enter your Choice :2
Enter the value : 400
Enter your Choice :9
100 200 300 400

Enter your Choice :10
Enter the value : 150

Enter your Choice :9
100 150 200 300 400

Enter your Choice :3
Deletion Successful

Enter your Choice :4
Deletion Successful

Enter your Choice :9
150 200 300

Enter your Choice :6
Enter the position : 2
Deletion Successful

Enter your Choice :9
150 300

Enter your Choice :1
Enter the value : 100

Enter your Choice :11
Copied List : 100 150 300

Enter your Choice :12
300 150 100

Enter your Choice :13
!!THANK YOU!!
```

6.) Develop a menu driven program to implement Double linked list with various operations such as

a. **Insertion and Deletion at front/rear**
b. **Insertion and Deletion at the specified position**

c. **Delete by Key**
d. **Search by key**
e. **Create an ordered list**
f. **Reverse a list**
g. **Creating a copy of the list**

```
#include <stdio.h>
#include<stdlib.h>
//Structure for a single Node
struct node
{
   int item;
   struct node * next; //Pointer to next Node
   struct node * prev; //Pointer to Previous Node
};
//struct node * type defined as NODE
typedef struct node * NODE;
//Function to create a Node
NODE getnode(int val)
{
   NODE temp; //Pointer to newly created node
   temp = (NODE)malloc(sizeof(struct node));
   temp->item=val;
   temp->next=NULL;
   temp->prev=NULL;
   return temp;
}
//Function to insert Node at front
NODE insertfront(NODE first,int val)
{
   {
   NODE temp; //Pointer to newly created node
   temp = getnode(val);
   if(first == NULL)
   {
      first = temp;
      return first;
   }
   else
   {
      temp -> next = first;
      first -> prev = temp;
      return temp;
   }
}
}
//Function to insert Node at Rear End
NODE insertrear(NODE first, int val)
{
   NODE temp,cur;
```

```
    temp = getnode(val); //Pointer to newly created node
    cur = first; //Pointer to Current Node
    if(first == NULL)
    {
        first = temp;
        return first;
    }
    else
    {
        while(cur->next!=NULL)
        {
            cur = cur->next;
        }
        cur->next = temp;
        temp->prev = cur;
        return first;
    }
}
//Function to delete Node at front
NODE deletefront(NODE first)
{
    if(first==NULL)
    {
        printf("List is empty\n");
        return first;
    }
    else
    {
        first = first->next;
        first->prev = NULL;
        return first;
    }
}
//Function to delete Node at Rear End
NODE deleterear(NODE first)
{
    NODE cur = first; //Pointer to Current Node
    NODE temp=NULL; //Pointer to Previous Node
    if(first==NULL)
    {
        printf("List is empty\n");
        return first;
    }
    else
    {

        while(cur->next!=NULL)
        {
            temp = cur;
            cur = cur->next;
        }
        temp->next = NULL;
        cur->prev=NULL;
        free(cur);
        return first;
    }
}
//Function to insert Node at Specified Position
NODE insertpos(NODE first, int val, int pos)
{
    NODE temp = getnode(val);
    NODE cur = first; //Pointer to Current Node
    NODE previous = NULL; //Pointer to Previous Node
    if(pos==1&&first==NULL)
    {
        first = insertfront(first,val);
        return first;
    }
     else if(pos!=1&&first==NULL)
    {
        printf("Invalid Position\n");
        return first;
    }
    else
    {
        while(pos!=1&&cur!=NULL)
        {
            previous=cur;
            cur = cur->next;
            pos--;
        }
        temp->next = cur;
        temp->prev = previous;
        previous->next = temp;
        cur->prev=temp;
        return first;
    }
}
//Function to delete a Node at Specified Position
NODE deletepos(NODE first,int pos)
{
    NODE after = first; //Pointer to Next Node
    NODE cur = NULL; //Pointer to Current Node
    NODE previous = NULL; // Pointer to Previous Node
    if(first==NULL)
    {
        printf("List is empty\n");
        return first;
    }
    while(pos != 0)
    {
        previous=cur;
        cur = after;
        after = after->next;
        pos--;
    }
    previous->next = after;
    after->prev=previous;
    free(cur); //Deleting the Current Node by Deallocating Memory
    cur = NULL;
    return first;
}
//Function to search a Key element in a list
int search(NODE first, int key)
{
    int i=1,flag=0,pos=0;
    NODE cur;
    cur = first;
    while(cur->next!=NULL)
    {
        if(cur->item==key)
        {
            flag = 1;
            pos = i;
            break;
        }
        cur = cur->next;
        i++;
    }
    if(flag==1)
    return pos;
    else
    return 0;
}
//Function to delete a Key element in a list
NODE deletebykey(NODE first,int key)
{
    int pos = search(first,key); //Position of the found Key
    NODE prev,cur;
    if(pos!=0)
    {
        first = deletepos(first,pos);
        return first;
    }
    else
    {
        printf("Element not found\n");
        return first;
    }
}
```

```
//Function to display all elements in a list
NODE display(NODE first)
{
   NODE cur=first;
   if(first==NULL)
   {
   printf("List is Empty\n");
   return first;
   }
   else
   {
      while (cur!=NULL)
      {
         printf("%d ",cur->item);
         cur = cur->next;
      }
      return first;
   }
}
//Function to reverse a linked list
NODE reverse(NODE first)
{
   NODE temp,cur;
   cur = first;
   temp=NULL;
   while(cur!= NULL)
   {
      temp = cur->prev;
      cur->prev = cur->next;
      cur->next = temp;
      cur = cur->prev;
   }
   if(temp!=NULL)
   first = temp->prev;
   display(first);
   return first;
}
NODE orderlist(NODE first, int val)
{
   NODE temp = getnode(val);
   NODE previous = NULL;
   NODE cur = first;
   while(cur->next!=NULL)
   {
      if(cur->item>temp->item)
      {
         break;
      }
      previous=cur;
      cur = cur->next;
```

```
    }
    temp->next = cur;
    temp->prev = previous;
    previous->next = temp;
    cur->prev=temp;
    return first;
}
//Function to create a Copy List
NODE createcopy(NODE first, NODE copy)
{
    NODE cur=first;
    while(cur!=NULL)
    {
        copy=insertrear(copy,cur->item);
        cur=cur->next;
    }
    return copy;
}
//Main Function
void main()
{
    NODE first = NULL; //Pointer to first node of list
    NODE copy = NULL; //Poiner to first node of copied list
    int choice;
    int key,val,pos;
    printf("\nMain Menu\n1.Insert Front\n2.Insert Rear\n3.Delete Front\n4.Delete Rear\n5.Insert By
Pos\n6.Delete by Pos\n7.Search\n8.Delete By Key\n9.Display\n10.Reverse\n11.Ordered List\n12.Create
Copy\n13.Exit\n");
    for(;;)
    {
        printf("\nEnter your Choice :");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1 : printf("Enter the value : ");
                     scanf("%d",&val);
                     first = insertfront(first,val);
                     break;
            case 2 : printf("Enter the value : ");
                     scanf("%d",&val);
                     first = insertrear(first,val);
                     break;
            case 3 : first = deletefront(first);
                     printf("Deletion Successfull\n");
                     break;
            case 4 : first = deleterear(first);
                     printf("Deletion Successfull\n");
                     break;
            case 5 : printf("Enter the value : ");
                     scanf("%d",&val);
```

```
                  printf("Enter the position : ");
                  scanf("%d",&pos);
                  first = insertpos(first,val,pos);
                  printf("Insertion Succesfull\n");
                  break;
            case 6 : printf("Enter the position : ");
                  scanf("%d",&pos);
                  first = deletepos(first,pos);
                  printf("Deletion Succesfull\n");
                  break;
            case 7 : printf("Enter the Key : ");
                  scanf("%d",&key);
                  pos = search(first,key);
                  if(pos!=0)
                  printf("The Key found at %d pos\n",pos);
                  else
                  printf("Element not found\n");
                  break;
            case 8 : printf("Enter the Key : ");
                  scanf("%d",&key);
                  first = deletebykey(first,key);
                  printf("Deletion Successfull\n");
                  break;
            case 9 : display(first);
                  break;
            case 10 : first = reverse(first);
                   break;
            case 11 : printf("Enter the value : ");
                  scanf("%d",&val);
                   first = orderlist(first,val);
                   break;
            case 12 : printf("Copied List : ");
                   copy = createcopy(first,copy);
                   display(copy);
                   break;
            case 13 : printf("!!THANK YOU!!\n");
                  exit(0);
            default : printf("Invalid Choice\n Enter again\n");
         }
      }
   }
```

Output :

```
Main Menu
1.Insert Front
```

```
2.Insert Rear
3.Delete Front
4.Delete Rear
5.Insert By Pos
6.Delete by Pos
7.Search
8.Delete By Key
9.Display
10.Reverse
11.Ordered List
12.Create Copy
13.Exit

Enter your Choice :1
Enter the value : 10

Enter your Choice :2
Enter the value : 20

Enter your Choice :2
Enter the value : 40

Enter your Choice :5
Enter the value : 30
Enter the position : 3

Enter your Choice :9
10 20 30 40

Enter your Choice :3
Deletion Successful

Enter your Choice :4
Deletion Successful

Enter your Choice :1
Enter the value : 5

Enter your Choice :2
Enter the value : 35

Enter your Choice :9
5 20 30 35

Enter your Choice :7
Enter the Key : 20
The Key found at 2 pos
Enter your Choice :11
Enter the value : 10
Enter your Choice :9
5 10 20 30 35

Enter your Choice :12
Copied List : 5 10 20 30 35

Enter your Choice :10
35 30 20 10 5

Enter your Choice :13
!!THANK YOU!!
```

7.) Develop a menu driven program to implement Circular Double linked list with Header node to perform various operations such as

  a. **Insertion and Deletion at front/rear**

b. **Insertion and Deletion at the specified position**
c. **Delete by Key**
d. **Search by key**
e. **Create an ordered list**
f. **Reverse a list**
g. **Creating a copy of the list**

```
#include <stdio.h>
#include<stdlib.h>
//Structure for a single Node
struct node
{
   int item;
   struct node * next; //Pointer to next Node
   struct node * prev; //Pointer to Previous Node
};
//struct node * type defined as NODE
typedef struct node * NODE;
//Function to create a Node
NODE getnode(int val)
{
   NODE temp; //Pointer to newly created node
   temp = (NODE)malloc(sizeof(struct node));
   temp->item=val;
   temp->next=temp;
   temp->prev=temp;
   return temp;
}
//Function to insert Node at front
NODE insertfront(NODE head,int val)
{
   NODE temp; //Pointer to newly created node
   temp = getnode(val);
   NODE first = head->next;
   if(head->next==NULL)
   {
      head->next = head->prev = temp;
      temp->prev = temp->next = head;
      return head;
   }
   else
   {
      temp -> next = first;
      first->prev = temp;
      temp -> prev = head;
      head->next=temp;
      return head;
   }
}
//Function to insert Node at Rear End
```

```
NODE insertrear(NODE head, int val)
{
   NODE temp,last;
   temp = getnode(val); //Pointer to newly created node
   last = head->prev; //Pointer to Current Node
   if(head->next==NULL)
   {
      head->next = head->prev = temp;
      temp->prev = temp->next = head;
      return head;
   }
   else
   {
      last->next=temp;
      temp->next=head;
      head->prev=temp;
      temp->prev=last;
      return head;
   }
}
//Function to delete Node at front
NODE deletefront(NODE head)
{
   NODE first=head->next;
   if(head->next==NULL)
   {
      printf("List is empty\n");
      return head;
   }
   else
   {
      head->next=first->next;
      free(first);
      return head;
   }
}
//Function to delete Node at Rear End
NODE deleterear(NODE head)
{
   NODE last = head->prev; //Pointer to Current Node
   NODE temp=NULL; //Pointer to Previous Node
   if(head->next==NULL)
   {
      printf("List is empty\n");
      return head;
   }
   else
   {
      head->prev=last->prev;
      free(last);
```

```
        return head;
    }
}
//Function to insert Node at Specified Position
NODE insertpos(NODE head, int val, int pos)
{
    NODE temp = getnode(val);
    NODE cur = head->next; //Pointer to Current Node
    NODE previous = NULL; //Pointer to Previous Node
    if(pos==1&&head==NULL)
    {
        head = insertfront(head,val);
        return head;
    }
     else if(pos!=1&&head==NULL)
    {
        printf("Invalid Position\n");
        return head;
    }
    else
    {
        while(pos!=1&&cur!=head)
        {
            previous=cur;
            cur = cur->next;
            pos--;
        }
        temp->next = cur;
        temp->prev = previous;
        previous->next = temp;
        cur->prev=temp;
        return head;
    }
}
//Function to delete a Node at Specified Position
NODE deletepos(NODE head,int pos)
{
    NODE after = head->next; //Pointer to Next Node
    NODE cur = NULL; //Pointer to Current Node
    NODE previous = NULL; // Pointer to Previous Node
    if(head->next==NULL)
    {
        printf("List is empty\n");
        return head;
    }
    while(pos != 0)
    {
        previous=cur;
        cur = after;
        after = after->next;
        pos--;
    }
    previous->next = after;
    after->prev=previous;
    free(cur); //Deleting the Current Node by Deallocating Memory
    cur = NULL;
    return head;
}
//Function to search a Key element in a list
int search(NODE head, int key)
{
    int i=1,flag=0,pos=0;
    NODE cur;
    cur = head->next;
    while(cur->next!=head)
    {
        if(cur->item==key)
        {
            flag = 1;
            pos = i;
            break;
        }
        cur = cur->next;
        i++;
    }
    if(flag==1)
    return pos;
    else
    return 0;
}
//Function to delete a Key element in a list
NODE deletebykey(NODE head,int key)
{
    int pos = search(head,key); //Position of the found Key
    if(pos!=0)
    {
        head = deletepos(head,pos);
        return head;
    }
    else
    {
        printf("Element not found\n");
        return head;
    }
}
//Function to display all elements in a list
NODE display(NODE head)
{
    NODE cur=head->next;
    if(head->next==NULL)
    {
        printf("List is Empty\n");
        return head;
    }
    else
    {
        while (cur!=head)
        {
            printf("%d ",cur->item);
            cur = cur->next;
        }
        return head;
    }
}
//Function to reverse a linked list
NODE reverse(NODE head)
{
    NODE temp,cur;
    cur = head->next;
    temp=NULL;
    do
    {
        temp = cur->prev;
        cur->prev = cur->next;
        cur->next = temp;
        cur = cur->prev;
    }while(cur!= head);
    if(temp!=NULL)
    head = temp->prev;
    display(head);
    return head;
}
//Function to create Ordered List
NODE orderlist(NODE head, int val)
{
    NODE temp = getnode(val);
    NODE previous = NULL;
    NODE cur = head->next;
    while(cur->next!=head)
    {
        if(cur->item>temp->item)
        {
            break;
        }
        previous=cur;
        cur = cur->next;
    }
    temp->next = cur;
    temp->prev = previous;
    previous->next = temp;
    cur->prev=temp;
    return head;
}
//Function to create a Copy List
NODE createcopy(NODE head, NODE copy)
{
    NODE cur=head->next;
    while(cur!=head)
    {
        copy=insertrear(copy,cur->item);
        cur=cur->next;
    }
    return copy;
}
//Main Function
void main()
{
    NODE head = getnode(0); //Pointer to header node of list
    NODE copy = getnode(0); //Poiner to header node of copied list
    int choice;
    int key,val,pos;
    printf("\nMain Menu\n1.Insert Front\n2.Insert Rear\n3.Delete Front\n4.Delete Rear\n5.Insert By
Pos\n6.Delete by Pos\n7.Search\n8.Delete By Key\n9.Display\n10.Reverse\n11.Ordered List\n12.Create
Copy\n13.Exit\n");
    for(;;)
    {
        printf("\nEnter your Choice :");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1 : printf("Enter the value : ");
                     scanf("%d",&val);
                     head = insertfront(head,val);
                     break;
            case 2 : printf("Enter the value : ");
                     scanf("%d",&val);
                     head = insertrear(head,val);
                     break;
            case 3 : head = deletefront(head);
                     printf("Deletion Successfull\n");
                     break;
            case 4 : head = deleterear(head);
                     printf("Deletion Successfull\n");
                     break;
            case 5 : printf("Enter the value : ");
                     scanf("%d",&val);
                     printf("Enter the position : ");
                     scanf("%d",&pos);
                     head = insertpos(head,val,pos);
                     printf("Insertion Succesfull\n");
```

```
                break;
            case 6 : printf("Enter the position : ");
                scanf("%d",&pos);
                head = deletepos(head,pos);
                printf("Deletion Succesfull\n");
                break;
            case 7 : printf("Enter the Key : ");
                scanf("%d",&key);
                pos = search(head,key);
                if(pos!=0)
                printf("The Key found at %d pos\n",pos);
                else
                printf("Element not found\n");
                break;
            case 8 : printf("Enter the Key : ");
                scanf("%d",&key);
                head = deletebykey(head,key);
                printf("Deletion Successfull\n");
                break;
            case 9 : display(head);
                break;
            case 10 : head = reverse(head);
                 break;
            case 11 : printf("Enter the value : ");
                scanf("%d",&val);
                 head = orderlist(head,val);
                 break;
            case 12 : printf("Copied List : ");
                 copy = createcopy(head,copy);
                 display(copy);
                 break;
            case 13 : printf("!!THANK YOU!!\n");
                exit(0);
            default : printf("Invalid Choice\n Enter again\n");
        }
    }
}
```

Output :

```
Main Menu
1.Insert Front
```

```
2.Insert Rear
3.Delete Front
4.Delete Rear
5.Insert By Pos
6.Delete by Pos
7.Search
8.Delete By Key
9.Display
10.Reverse
11.Ordered List
12.Create Copy
13.Exit

Enter your Choice :1
Enter the value : 5

Enter your Choice :2
Enter the value : 15

Enter your Choice :2
Enter the value : 35

Enter your Choice :5
Enter the value : 25
Enter the position : 3

Enter your Choice :9
5 15 25 35

Enter your Choice :3
Deletion Successful

Enter your Choice :4
Deletion Successful

Enter your Choice :1
Enter the value : 5

Enter your Choice :2
Enter the value : 35

Enter your Choice :9
5 15 25 35

Enter your Choice :7
Enter the Key : 15
The Key found at 2 pos
Enter your Choice :11
Enter the value : 10
Enter your Choice :9
5 10 15 25 35

Enter your Choice :12
Copied List : 5 10 15 25 35

Enter your Choice :10
35 25 15 10 5

Enter your Choice :13
!!THANK YOU!!
```

8.) Develop a menu driven program to implement Stack with static and dynamic memory allocation mechanisms using array storage representation. (Represent Stack using structure)

a) Static Allocation :-

```
#include <stdio.h>
#include <stdlib.h>
#define size 10
struct Stack{
   int stk[size]; // Array to Implement Stack
   int top; // Index Of Top Of Stack
}S;
//Function to Push an Element into Stack
void push(int X)
{
   //Checking if Stack is Full
   if(S.top==size-1)
      printf("Stack Overflow\n");
   else
      S.stk[++S.top] = X;
}
//Function to Pop an Element from Stack
void pop()
{
   //Checking if Stack is Empty
   if(S.top==-1)
      printf("Stack UnderFlow\n");
   else
   {
      printf("Deleted Element : %d\n",S.stk[S.top]);
      S.top--;
   }
}
//Function to Display the top Element of Stack
void peek()
{
   //Checking if Stack is Empty
   if(S.top==-1)
      printf("Stack UnderFlow\n");
   else
      printf("%d\n",S.stk[S.top]);
}
//Function to Display all Elements of Stack
void display()
{
   //Checking if Stack is Empty
   if(S.top==-1)
      printf("Stack UnderFlow\n");
   else
   {
      printf("The Stack is :\n");
      for(int i=S.top;i>=0;i--)
         printf("%d\n",S.stk[i]);
   }
}
//Main Function
void main()
{
   int choice,X;
   S.top=-1; //Initailizing Empty Stack
   for(;;)
   {
      printf("Menu\n1.Push\n2.Pop\n3.Peek\n4.Display\n5.Exit\n");
      printf("Enter Your Choice : ");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1 : printf("Enter the value to be pushed : ");
                  scanf("%d",&X);
                  push(X);
                  break;
         case 2 : pop();
                  break;
         case 3 : peek();
                  break;
         case 4 : display();
                  break;
         case 5 : printf("!! THANK YOU !!\n");
                  exit(0);
         default : printf("Invalid Choice\n");
      }
   }
}
```

Output :

```
Menu
1.Push
2.Pop
3.Peek
4.Display
5.Exit
Enter Your Choice : 1
Enter the value to be pushed : 10

Enter Your Choice : 1
Enter the value to be pushed : 20

Enter Your Choice : 1
Enter the value to be pushed : 30

Enter Your Choice : 1
Enter the value to be pushed : 40

Enter Your Choice : 1
Enter the value to be pushed : 50
Enter Your Choice : 1
Enter the value to be pushed : 60
Stack Overflow

Enter Your Choice : 3
50

Enter Your Choice : 4
The Stack is :
50
40
30
20
10

Enter Your Choice : 2
Deleted Element : 50

Enter Your Choice : 2
Deleted Element : 40
Enter Your Choice : 5
!! THANK YOU !!
```

b) Dynamic Allocation :-

```
#include <stdio.h>
#include <stdlib.h>
int size;
struct Stack{
   int *stk; //Pointer to Array storing Stack
   int top; //Index of Top of Stack
}S;
//Function to Push an Element into Stack
void push(int X)
{
   //Checking if Stack is Full
   if(S.top==size-1)
   {
      printf("Stack Overflow\n");
      printf("Reallocating Memory\n");
      size++;
      //Reallocating Memory
      S.stk=(int *)realloc(S.stk,size*sizeof(int));
   }
   S.top++;
   *(S.stk+S.top)=X;
}
//Function to Pop an Element from Stack
void pop()
{
    //Checking if Stack is Empty
    if(S.top==-1)
        printf("Stack UnderFlow\n");
    else
    {
        printf("Deleted Element : %d\n",*(S.stk+S.top));
        S.top--;
    }
}
//Function to Display the top Element of Stack
void peek()
{
    //Checking if Stack is Empty
    if(S.top==-1)
        printf("Stack UnderFlow\n");
    else
        printf("%d\n",*(S.stk+S.top));
}
//Function to Display all Elements of Stack
void display()
{
    //Checking if Stack is Empty
    if(S.top==-1)
        printf("Stack UnderFlow\n");
    else
    {
        printf("The Stack is :\n");
        for(int i=S.top;i>=0;i--)
            printf("%d\n",*(S.stk+i));
    }
}
void main()
{
    int choice,X;
    printf("Enter the size of Stack : ");
    scanf("%d",&size);
    //Allocating Memory for the Stack Array
    S.stk = (int *)calloc(size,sizeof(int));
    S.top=-1; //Initializing Empty Stack
    for(;;)
    {
        printf("Menu\n1.Push\n2.Pop\n3.Peek\n4.Display\n5.Exit\n");
        printf("Enter Your Choice : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1 : printf("Enter the value to be pushed : ");
                     scanf("%d",&X);
                     push(X);
                     break;
            case 2 : pop();
                     break;
            case 3 : peek();
                     break;
            case 4 : display();
                     break;
            case 5 : printf("!! THANK YOU !!\n");
                     exit(0);
            default : printf("Invalid Choice\n");
        }
    }
}
```

Output:

```
Enter the size of Stack : 3
Menu
1.Push
2.Pop
3.Peek
4.Display
5.Exit
Enter Your Choice: 1
Enter the value to be pushed: 10
Enter Your Choice: 1
Enter the value to be pushed: 20
Enter Your Choice: 1
Enter the value to be pushed: 30
Enter Your Choice: 1
Enter the value to be pushed: 40
Stack Overflow
Reallocating Memory
Enter Your Choice: 3
40
Enter Your Choice: 4
The Stack is:
40
30
20
10
Enter Your Choice: 2
Deleted Element: 40
Enter Your Choice: 4
The Stack is:
30
20
10
Enter Your Choice: 5
!! THANK YOU!!
```

9. a) Develop a menu driven program to convert infix expressions to postfix expression and evaluate the postfix expression.

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
//Function to check Stack Precedence
int stackprec(char symbol)
{
   switch(symbol)
   {
      case '+' :
      case '-' : return 2;
      case '*' :
      case '/' :
      case '%' : return 4;
      case '^' :
      case '$' : return 5;
      case '(' : return 0;
      case '#' : return -1;
      default : return 8;
   }
}
//Function to check Input Precedence
int inprec(char symbol)
{
   switch(symbol)
   {
      case '+' :
      case '-' : return 1;
      case '*' :
      case '/' :
      case '%' : return 3;
      case '^' :
      case '$' : return 6;
      case '(' : return 9;
      case ')' : return 0;
      default : return 7;
   }
}
//Function to convert Infix to Postfix Expression
void infixtopostfix(char infix[], char postfix[])
{
   char S[30]; //Stack to store the Expression
   int i=0; //Index for Infix Expression
   int j=0; //Index for Postfix Expression
   int top=-1; //Initializing Empty Stack
   S[++top] = '#';
   char symbol;
   for(i=0;i<strlen(infix);i++)
   {
      symbol = infix[i];
```

```
        while(stackprec(S[top])>inprec(symbol))
        {
           postfix[j++]=S[top--];
        }
        if(stackprec(S[top])!=inprec(symbol))
        {
           S[++top] = symbol;
        }
        else
        {
           top--;
        }
    }
    while(S[top]!='#')
    {
        postfix[j]=S[top--];
        j++;
    }
    postfix[j]='\0';
}
//Function to Evaluate Postfix Expression
void postfixevaluate(char postfix[])
{
    int i,op1,op2,res;
    int Stack[10]; //Stack to store Operands
    int top=-1; //Initializing Empty Stack
    char ch;
    for(i=0;i<strlen(postfix);i++)
    {
        ch = postfix[i];
        if(isdigit(ch))
        {
           Stack[++top]=ch-'0'; //Pushing Operand into Stack
        }
        else
        {
           if(top<1)
              printf("Not enough operands\n");
           else
           {
              op2=Stack[top--]; //Popping Operand-2
              op1=Stack[top--]; //Popping Operand-1
              switch (ch)
              {
              case '+' : res = op1 + op2;
                         break;
              case '-' : res = op1 - op2;
                         break;
              case '/' : res = op1 / op2;
                         break;
```

```
                case '*' : res = op1 * op2;
                        break;
                case '%' : res = op1 % op2;
                        break;
                default: printf("Invalid Operator\n");
                }
                Stack[++top] = res; //Pushing Result into Stack
            }
        }
    }
    printf("Final Result = %d\n",Stack[top]);
}
void main()
{
    char infix[30]; //Array to store Infix Expression
    char postfix[30]; //Array to store Postfix Expression
    int choice;
    printf("Menu\n1.Enter Infix Expression\n2.Convert to Postfix\n3.Evaluate Postfix\n4.Exit\n");
    for(;;)
    {
        printf("Enter Your Choice: ");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1 : printf("Enter the Infix Expression : ");
                scanf("%s",infix);
                break;
        case 2 : infixtopostfix(infix,postfix);
                printf("The Postfix Expression is : ");
                printf("%s\n",postfix);
                break;
        case 3 : postfixevaluate(postfix);
                break;
        case 4 : printf("!! THANK YOU !!\n");
                exit(0);
        default: printf("Invalid Choice\n");
        }
    }
}
```

9. b) Develop a menu driven program to convert infix expression to prefix and evaluate the prefix expression.

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

```
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
//Function to check Stack Precedence
int stackprec(char symbol)
{
   switch(symbol)
   {
      case '+' :
      case '-' : return 1;
      case '*' :
      case '/' :
      case '%' : return 3;
      case '^' :
      case '$' : return 6;
      case ')' : return 0;
      case '#' : return -1;
      default : return 8;
   }
}
//Function to check Input Precedence
int inprec(char symbol)
{
   switch(symbol)
   {
      case '+' :
      case '-' : return 2;
      case '*' :
      case '/' :
      case '%' : return 4;
      case '^' :
      case '$' : return 5;
      case '(' : return 0;
      case ')' : return 9;
      default : return 7;
   }
}
//Function to convert Infix to Prefix Expression
void infixtoprefix(char infix[], char prefix[])
{
   char S[30]; //Stack to store the Expression
   int i=0; //Index for Infix Expression
   int j=0; //Index for Postfix Expression
   int top=-1; //Initializing Empty Stack
   S[++top] = '#';
   char symbol;
   strrev(infix);
   for(i=0;i<strlen(infix);i++)
   {
      symbol = infix[i];
```

```
        while(stackprec(S[top])>inprec(symbol))
        {
            prefix[j++]=S[top--];
        }
        if(stackprec(S[top])!=inprec(symbol))
        {
            S[++top] = symbol;
        }
        else
        {
            top--;
        }
    }
    while(S[top]!='#')
    {
        prefix[j]=S[top--];
        j++;
    }
    prefix[j]='\0';
    strrev(prefix);
}
//Function to Evaluate Prefix Expression
void prefixevaluate(char prefix[])
{
    int i,op1,op2,res;
    strrev(prefix);
    int Stack[10]; //Stack to store Operands
    int top=-1; //Initializing Empty Stack
    char ch;
    for(i=0;i<strlen(prefix);i++)
    {
        ch = prefix[i];
        if(isdigit(ch))
        {
            Stack[++top]=ch-'0'; //Pushing Operand into Stack
        }
        else
        {
            if(top<1)
                printf("Not enough operands\n");
            else
            {
                op1=Stack[top--]; //Popping Operand-1
                op2=Stack[top--]; //Popping Operand-2
                switch (ch)
                {
                case '+' : res = op1 + op2;
                           break;
                case '-' : res = op1 - op2;
                           break;
                case '/' : res = op1 / op2;
                           break;
                case '*' : res = op1 * op2;
                           break;
                case '%' : res = op1 % op2;
                           break;
                default: printf("Invalid Operator\n");
                }
                Stack[++top] = res; //Pushing Result into Stack
            }
        }
    }
    printf("Final Result = %d\n",Stack[top]);
}
void main()
{
    char infix[30]; //Array to store Infix Expression
    char prefix[30]; //Array to store Prefix Expression
    int choice;
    printf("Menu\n1.Enter Infix Expression\n2.Convert to Prefix\n3.Evaluate prefix\n4.Exit\n");
    for(;;)
    {
        printf("Enter Your Choice: ");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1 : printf("Enter the Infix Expression : ");
                 scanf("%s",infix);
                 break;
        case 2 : infixtoprefix(infix,prefix);
                 printf("The Prefix Expression is : ");
                 printf("%s\n",prefix);
                 break;
        case 3 : prefixevaluate(prefix);
                 break;
        case 4 : printf("!! THANK YOU !!\n");
                 exit(0);
        default: printf("Invalid Choice\n");
        }
    }
}
```

Output :

```
Menu
1.Enter Infix Expression
```

```
2.Convert to Postfix
3.Evaluate Postfix
4.Exit

Enter Your Choice: 1
Enter the Infix Expression : (5+6)*(7-2)

Enter Your Choice: 2
The Postfix Expression is : 56+72-*

Enter Your Choice: 3
Final Result = 55

Enter Your Choice: 4
!! THANK YOU !!
```

Output :

```
Menu
1.Enter Infix Expression
2.Convert to Prefix
3.Evaluate prefix
4.Exit

Enter Your Choice: 1
Enter the Infix Expression : (8-5)*(7+3)

Enter Your Choice: 2
The Prefix Expression is : *-85+73

Enter Your Choice: 3
Final Result = 30

Enter Your Choice: 4
!! THANK YOU !!
```

10.a) Develop a menu driven program to implement ordinary Queue with static and dynamic memory allocation mechanisms using array storage representation. (Represent Queue using structure).

```
//Simple Queue Static
#include <stdio.h>
#include <stdlib.h>
#define size 5
struct Queue{
   int front; //Index Of front element in Queue
   int rear; //Index of rear element in Queue
   int queue[size]; //Array to store Queue
}Q;
//Function to insert an element to rear of Queue
void insert(int val)
{
   //Checking if Queue is Full
   if(Q.rear==size-1)
   {
      printf("Queue is Full\n");
      return;
   }
   if(Q.front==-1)
   {
      Q.front = 0; //Initializing First element
   }
   Q.rear++;
   Q.queue[Q.rear] = val;
   printf("Insertion Successful\n");
}
//Function to delete the front element of Queue
void delete()
{
   //Checking if Queue is Empty
   if(Q.front==-1)
   {
      printf("Queue is Empty");
   }
   else
   {
      //Deleting Element from front
      printf("Deleted Element : %d\n",Q.queue[Q.front]);
      Q.front++;
   }
   if(Q.front==Q.rear)
   {
      Q.front=Q.rear=-1;
   }
}
//Function Display all elements of Queue
void display()
{
   //Checking if Queue is Empty
   if(Q.rear==-1)
```

```
    {
        printf("Queue is Empty");
    }
    else
    {
        for(int i=Q.front;i<=Q.rear;i++)
        printf("%d\n",Q.queue[i]);
    }
}
void main()
{
    int choice,val;
    Q.front=Q.rear=-1; //Initializing Empty Queue
    printf("Main Menu\n1.Insert\n2.Delete\n3.Display\n4.Exit\n");
    for(;;)
    {
        printf("Enter Your choice : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1 : printf("Enter the element to be inserted : ");
                     scanf("%d",&val);
                     insert(val);
                     break;
            case 2 : delete();
                     break;
            case 3 : display();
                     break;
            case 4 : printf("!! THANK YOU !!\n");
                     exit(0);
            default : printf("Invalid Choice\nEnter Again\n");
        }
    }
}
```

Output:

```
Main Menu
1.Insert
```

```
2.Delete
3.Display
4.Exit
Enter Your choice : 1
Enter the element to be inserted : 10
Insertion Successful
Enter Your choice : 1
Enter the element to be inserted : 20
Insertion Successful
Enter Your choice : 1
Enter the element to be inserted : 30
Insertion Successful
Enter Your choice : 1
Enter the element to be inserted : 40
Insertion Successful
Enter Your choice : 1
Enter the element to be inserted : 50
Insertion Successful
Enter Your choice : 1
Enter the element to be inserted : 60
Queue is Full
Enter Your choice : 3
10
20
30
40
50
Enter Your choice : 2
Deleted Element : 10
Enter Your choice : 2
Deleted Element : 20
Enter Your choice : 3
30
40
50
Enter Your choice : 4
!! THANK YOU !!
```

```c
//Simple Queue Dynamic
#include <stdio.h>
#include <stdlib.h>
```

```
int size;
struct Queue{
   int front; //Index Of front element in Queue
   int rear; //Index of rear element in Queue
   int *queue; //Base address of array to store Queue
}Q;
//Function to insert an element to rear of Queue
void insert(int val)
{
   //Checking if Queue is Full
   if(Q.rear==size-1)
   {
      printf("Queue is Full\n");
      printf("Reallocating Memory\n");
      size++;
      Q.queue = (int *)realloc(Q.queue,size*sizeof(int)); //Reallocating Array for Queue
   }
   if(Q.front==-1)
   {
      Q.front = 0; //Initializing First element
   }
   Q.rear++;
   *(Q.queue+Q.rear) = val;
   printf("Insertion Successful\n");
}
//Function to delete the front element of Queue
void delete()
{
   //Checking if Queue is Empty
   if(Q.front==-1)
   {
      printf("Queue is Empty");
   }
   else
   {
      //Deleting Element from front
      printf("Deleted Element : %d\n",*(Q.queue+Q.front));
      Q.front++;
   }
   if(Q.front==Q.rear)
   {
      Q.front=Q.rear=-1;
   }
}
//Function Display all elements of Queue
void display()
{
   //Checking if Queue is Empty
   if(Q.rear==-1)
   {
        printf("Queue is Empty");
    }
    else
    {
        for(int i=Q.front;i<=Q.rear;i++)
        printf("%d\n",*(Q.queue+i));
    }
}
void main()
{
    int choice,val;
    Q.front=Q.rear=-1; //Initializing Empty Queue
    printf("Enter the Size : ");
    scanf("%d",&size);
    Q.queue=(int *)malloc(size*sizeof(int)); //Dynamically Allocating Array for Queue
    printf("Main Menu\n1.Insert\n2.Delete\n3.Display\n4.Exit\n");
    for(;;)
    {
        printf("Enter Your choice : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1 : printf("Enter the element to be inserted : ");
                     scanf("%d",&val);
                     insert(val);
                     break;
            case 2 : delete();
                     break;
            case 3 : display();
                     break;
            case 4 : printf("!! THANK YOU !!\n");
                     exit(0);
            default : printf("Invalid Choice\nEnter Again\n");
        }
    }
}
```

Output:

```
Enter the Size : 3
```

```
Main Menu
1.Insert
2.Delete
3.Display
4.Exit
Enter Your choice : 1
Enter the element to be inserted : 10
Insertion Successful
Enter Your choice : 1
Enter the element to be inserted : 20
Insertion Successful
Enter Your choice : 1
Enter the element to be inserted : 30
Insertion Successful
Enter Your choice : 1
Enter the element to be inserted : 40
Queue is Full
Reallocating Memory
Insertion Successful
Enter Your choice : 3
10
20
30
40
Enter Your choice : 2
Deleted Element : 10
Enter Your choice : 3
20
30
40
Enter Your choice : 4
!! THANK YOU !!
```

10.b) Develop a menu driven program to implement Circular Queue with static and dynamic memory allocation mechanisms using array storage representation.(Represent Queue using structure)

```
//Circular Queue Static
```

```
#include <stdio.h>
#include <stdlib.h>
#define size 3
struct Queue{
   int front; //Index Of front element in Queue
   int rear; //Index of rear element in Queue
   int queue[size]; //Array to store Queue
}Q;
//Function to insert an element to rear of Queue
void insert()
{
   int val;
   //Checking if Queue is Full
   if(Q.front==0&&Q.rear==size-1)
   {
      printf("Queue is Full\n");
      return;
   }
   if(Q.front==-1&Q.rear==-1)
      Q.front=0; //Initializing First element
   printf("Enter the element to be inserted : ");
   scanf("%d",&val);
   Q.rear=(Q.rear+1)%size;
   Q.queue[Q.rear] = val;
   printf("Insertion Successful\n");
}
//Function to delete the front element of Queue
void delete()
{
   //Checking if Queue is Empty
   if(Q.front==-1)
   {
      printf("Queue is Empty\n");
   }
   else if(Q.front==Q.rear)
   {
      printf("Deleted Element : %d\n",Q.queue[Q.front]);
      Q.front=Q.rear=-1;
   }
   else
   {
      //Deleting Element from front
      printf("Deleted Element : %d\n",Q.queue[Q.front]);
      Q.front=(Q.front+1)%size;
   }
}
//Function Display all elements of Queue
void display()
{
   int i;
```

```
    //Checking if Queue is Empty
    if(Q.front==-1)
    {
        printf("Queue is Empty\n");
    }
    else
    {
        if(Q.front<Q.rear)
        for(i=Q.front;i<=Q.rear;i++)
        {
            printf("%d\n",Q.queue[i]);
        }
        else
        {
            for(i=Q.front;i<=size-1;i++)
                printf("%d\n",Q.queue[i]);
            for(i=0;i<=Q.rear;i++)
                printf("%d\n",Q.queue[i]);
        }
    }
}
void main()
{
    int choice,val;
    Q.front=Q.rear=-1; //Initializing Empty Queue
    printf("Main Menu\n1.Insert\n2.Delete\n3.Display\n4.Exit\n");
    for(;;)
    {
        printf("Enter Your choice : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1 : insert();
                     break;
            case 2 : delete();
                     break;
            case 3 : display();
                     break;
            case 4 : printf("!! THANK YOU !!\n");
                     exit(0);
            default : printf("Invalid Choice\nEnter Again\n");
        }
    }
}
```

Output:

```
Main Menu
```

```
1.Insert
2.Delete
3.Display
4.Exit
Enter Your choice : 1
Enter the element to be inserted : 10
Insertion Successful
Enter Your choice : 1
Enter the element to be inserted : 20
Insertion Successful
Enter Your choice : 1
Enter the element to be inserted : 30
Insertion Successful
Enter Your choice : 3
10
20
30
Enter Your choice : 1
Queue is Full
Enter Your choice : 2
Deleted Element : 10
Enter Your choice : 1
Enter the element to be inserted : 40
Insertion Successful
Enter Your choice : 3
20
30
40
Enter Your choice : 4
!! THANK YOU !!
```

```
//Circular Queue Dynamic
#include <stdio.h>
#include <stdlib.h>
int size;
```

```
struct Queue{
   int front; //Index Of front element in Queue
   int rear; //Index of rear element in Queue
   int *queue; //Base address of array to store Queue
}Q;
//Function to insert an element to rear of Queue
void insert()
{
   int val;
   //Checking if Queue is Full
   if(Q.front==0&&Q.rear==size-1)
   {
      printf("Queue is Full\n");
      printf("Reallocating Memory\n");
      size++;
      Q.queue = (int *)realloc(Q.queue,size*sizeof(int));
   }
   if(Q.front==-1&Q.rear==-1)
      Q.front=0; //Initializing First element
   printf("Enter the element to be inserted : ");
   scanf("%d",&val);
   Q.rear=(Q.rear+1)%size;
   *(Q.queue+Q.rear) = val;
   printf("Insertion Successful\n");
}
//Function to delete the front element of Queue
void delete()
{
   //Checking if Queue is Empty
   if(Q.front==-1)
   {
      printf("Queue is Empty\n");
   }
   else if(Q.front==Q.rear)
   {
      printf("Deleted Element : %d\n",*(Q.queue+Q.front));
      Q.front=Q.rear=-1;
   }
   else
   {
      //Deleting Element from front
      printf("Deleted Element : %d\n",*(Q.queue+Q.front));
      Q.front=(Q.front+1)%size;
   }
}
//Function Display all elements of Queue
void display()
{
   int i;
   //Checking if Queue is Empty
   if(Q.front==-1)
   {
      printf("Queue is Empty\n");
   }
   else
   {
      if(Q.front<Q.rear)
      for(i=Q.front;i<=Q.rear;i++)
      {
         printf("%d\n",*(Q.queue+i));
      }
      else
      {
         for(i=Q.front;i<=size-1;i++)
         {
            printf("%d\n",*(Q.queue+i));
         }
         for(i=0;i<=Q.rear;i++)
         {
            printf("%d\n",*(Q.queue+i));
         }
      }
   }
}
void main()
{
   int choice,val;
   Q.front=Q.rear=-1; //Initializing Empty Queue
   printf("Enter the Size : ");
   scanf("%d",&size);
   Q.queue=(int *)malloc(size*sizeof(int)); //Dynamically Allocating Array for Queue
   printf("Main Menu\n1.Insert\n2.Delete\n3.Display\n4.Exit\n");
   for(;;)
   {
      printf("Enter Your choice : ");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1 : insert();
                  break;
         case 2 : delete();
                  break;
         case 3 : display();
                  break;
         case 4 : printf("!! THANK YOU !!\n");
                  exit(0);
         default : printf("Invalid Choice\nEnter Again\n");
      }
   }
}
```

Output:

```
Enter the Size : 3
Main Menu
1.Insert
2.Delete
3.Display
4.Exit
Enter Your choice : 1
Enter the element to be inserted : 10
Insertion Successful
Enter Your choice : 1
Enter the element to be inserted : 20
Insertion Successful
Enter Your choice : 1
Enter the element to be inserted : 30
Insertion Successful
Enter Your choice : 3
10
20
30
Enter Your choice : 1
Queue is Full
Reallocating Memory
Enter the element to be inserted : 40
Insertion Successful
Enter Your choice : 3
10
20
30
40
Enter Your choice : 2
Deleted Element : 10
Enter Your choice : 3
20
30
40
Enter Your choice : 4
!! THANK YOU !!
```

11.a) Develop a menu driven program to implement Double Ended Queue with static and dynamic memory allocation mechanisms using array storage representation. (Represent Queue using structure)

```
//Double Ended Queue Static
#include <stdio.h>
```

```
#include <stdlib.h>
#define size 5
struct Queue
{
   int queue[size]; //Array to store Queue
   int front; //Index Of front element in Queue
   int rear; //Index of rear element in Queue
}Q;
//Function to insert an element to rear of Queue
void pushrear(int val)
{
   //Checking if Queue is Full
   if(Q.rear==size-1&&Q.front==0||Q.front>Q.rear)
   {
      printf("Queue is FUll\n");
      return;
   }
   if(Q.front==-1)
   {
     Q.front=Q.rear=0;
   }
   if(Q.rear==size-1)
   {
      Q.rear = 0;
   }
   else
   {
      Q.rear++;
   }
   Q.queue[Q.rear] = val;
   printf("Insertion Succesfull\n");
}
//Function to insert an element to front of Queue
void pushfront(int val)
{
   //Checking if Queue is Full
   if(Q.rear==size-1&&Q.front==0||Q.front>Q.rear)
   {
      printf("Queue is FUll\n");
   }
   if(Q.front==-1)
   {
      Q.front=Q.rear=0;
   }
   else if(Q.front==0)
   {
      Q.front = size-1;
   }
   else
   {
        Q.front--;
    }
    Q.queue[Q.front]=val;
    printf("Insertion Succesfull\n");
}
//Function to delete the front element of Queue
void popfront()
{
    if(Q.front==-1)
    {
        printf("Queue is Empty");
    }
    else
    {
        //Deleting Element from front
        printf("Deleted Element : %d\n",Q.queue[Q.front]);
    }
    if(Q.front==Q.rear)
    {
        Q.front=Q.rear=-1;
    }
    else if (Q.front==size-1)
    {
        Q.front=0;
    }
    else
    {
        Q.front++;
    }
}
//Function to delete the rear element of Queue
void poprear()
{
if(Q.front==-1)
    {
        printf("Queue is Empty");
    }
    else
    {
        //Deleting Element from rear
        printf("Deleted Element : %d\n",Q.queue[Q.rear]);
    }
    if(Q.front==Q.rear)
    {
        Q.front=Q.rear=-1;
    }
    else if(Q.rear == 0)
    {
        Q.rear = size-1;
    }
    else
    {
        Q.rear--;
    }
}
//Function Display all elements of Queue
void display()
{
    int i,fpos=Q.front,rpos=Q.rear;
    //Checking if Queue is Empty
    if(Q.front==-1)
    {
        printf("Queue is Empty");
    }
    if (fpos<=rpos)
    {
        for(i=fpos;i<=rpos;i++)
        printf("%d ",Q.queue[i]);
    }

    else
    {
        for(i=fpos;i<=size-1;i++)
        printf("%d ",Q.queue[i]);
        fpos=0;
        for(i=fpos;i<=rpos;i++)
        printf("%d ",Q.queue[i]);
    }
    printf("\n");
}
void main()
{
    Q.front=Q.rear=-1; //Initializing Empty Queue
    int val,choice;
    printf("Main Menu\n1.Push-Front\n2.Pop-Front\n3.Push-Rear\n4.Pop-Rear\n5.Display\n6.Exit\n");
    for(;;)
    {
        printf("Enter Your choice : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1 : printf("Enter the element to be inserted : ");
                     scanf("%d",&val);
                     pushfront(val);
                     break;
            case 2 : popfront();
                     break;
            case 3 : printf("Enter the element to be inserted : ");
                     scanf("%d",&val);
                     pushrear(val);
```

```
                    break;
            case 4 : poprear();
                    break;
            case 5 : printf("Queue : ");
                    display();
                    break;
            case 6 : printf("!! THANK YOU !!\n");
                    exit(0);
            default : printf("Invalid Choice\nEnter Again\n");
        }
    }
}
```

Output:

```
 Main Menu
 1.Push-Front
 2.Pop-Front
 3.Push-Rear
 4.Pop-Rear
 5.Display
 6.Exit
 Enter Your choice : 1
 Enter the element to be inserted : 10
 Insertion Successful
 Enter Your choice : 3
 Enter the element to be inserted : 20
 Insertion Successful
 Enter Your choice : 3
 Enter the element to be inserted : 30
 Insertion Successful
 Enter Your choice : 3
 Enter the element to be inserted : 40
 Insertion Successful
 Enter Your choice : 3
 Enter the element to be inserted : 50
 Insertion Successful
 Enter Your choice : 3
 Enter the element to be inserted : 60
 Queue is Full
 Enter Your choice : 5
 Queue : 10 20 30 40 50
 Enter Your choice : 2
 Deleted Element : 10
 Enter Your choice : 4
 Deleted Element : 50
 Enter Your choice : 6
!! THANK YOU !!
```

```
//Double Ended Queue Dynamic
#include <stdio.h>
#include <stdlib.h>
int size;
```

```
struct Queue
{
   int *queue; //Base address of array to store Queue
   int front; //Index Of front element in Queue
   int rear; //Index of rear element in Queue
}Q;
//Function to insert an element to rear of Queue
void pushrear(int val)
{
   //Checking if Queue is Full
   if(Q.rear==size-1&&Q.front==0||Q.front>Q.rear)
   {
      printf("Queue is Full\n");
      printf("Reallocating Memory\n");
      size++;
      Q.queue = (int *)realloc(Q.queue,size*sizeof(int));
   }
   if(Q.front==-1)
     Q.front=Q.rear=0;
   if(Q.rear==size-1)
   {
      Q.rear = 0;
   }
   else
   {
      Q.rear++;
   }
   *(Q.queue+Q.rear) = val;
   printf("Insertion Succesfull\n");
}
//Function to insert an element to front of Queue
void pushfront(int val)
{
   //Checking if Queue is Full
   if(Q.rear==size-1&&Q.front==0||Q.front>Q.rear)
   {
      printf("Queue is Full\n");
      printf("Reallocating Memory\n");
      size++;
      Q.queue = (int *)realloc(Q.queue,size*sizeof(int));
   }
   if(Q.front==-1)
   {
      Q.front=Q.rear=0;
   }
   else if(Q.front==0)
   {
      Q.front = size-1;
   }
   else
```

```
    {
        Q.front--;
    }
    *(Q.queue+Q.front)=val;
    printf("Insertion Succesfull\n");
}
//Function to delete the front element of Queue
void popfront()
{
    if(Q.front==-1)
    {
        printf("Queue is Empty");
    }
    else
    {
        //Deleting Element from front
        printf("Deleted Element : %d\n",*(Q.queue+Q.front));
    }
    if(Q.front==Q.rear)
    {
        Q.front=Q.rear=-1;
    }
    else if (Q.front==size-1)
    {
        Q.front=0;
    }
    else
    {
        Q.front++;
    }
}
//Function to delete the rear element of Queue
void poprear()
{
if(Q.front==-1)
    {
        printf("Queue is Empty");
    }
    else
    {
        //Deleting Element from rear
        printf("Deleted Element : %d\n",*(Q.queue+Q.rear));
    }
    if(Q.front==Q.rear)
        Q.front=Q.rear=-1;

else if(Q.rear == 0)
        Q.rear = size-1;
    else
        Q.rear--;
```

```
}
//Function Display all elements of Queue
void display()
{
   int i,fpos=Q.front,rpos=Q.rear;
   //Checking if Queue is Empty
   if(Q.front==-1)
   {
      printf("Queue is Empty");
   }
   if (fpos<=rpos)
   {
      for(i=fpos;i<=rpos;i++)
      printf("%d ",*(Q.queue+i));
   }

   else
   {
      for(i=fpos;i<=size-1;i++)
      printf("%d ",*(Q.queue+i));
      fpos=0;
      for(i=fpos;i<=rpos;i++)
      printf("%d ",*(Q.queue+i));
   }
   printf("\n");
}
void main()
{
   Q.front=Q.rear=-1; //Initializing Empty Queue
   int val,choice;
   printf("Enter the Size : ");
   scanf("%d",&size);
   Q.queue=(int *)malloc(size*sizeof(int)); //Dynamically Allocating Array for Queue
   printf("Main Menu\n1.Push-Front\n2.Pop-Front\n3.Push-Rear\n4.Pop-Rear\n5.Display\n6.Exit\n");
   for(;;)
   {
      printf("Enter Your choice : ");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1 : printf("Enter the element to be inserted : ");
                  scanf("%d",&val);
                  pushfront(val);
                  break;
         case 2 : popfront();
                  break;
         case 3 : printf("Enter the element to be inserted : ");
                  scanf("%d",&val);
                  pushrear(val);
                  break;
```

```
            case 4 : poprear();
                    break;
            case 5 : printf("Queue : ");
                    display();
                    break;
            case 6 : printf("!! THANK YOU !!\n");
                    exit(0);
            default : printf("Invalid Choice\nEnter Again\n");
        }
    }
}
```

Output:

```
Enter the Size : 2
Main Menu
1.Push-Front
2.Pop-Front
3.Push-Rear
4.Pop-Rear
5.Display
6.Exit
Enter Your choice : 1
Enter the element to be inserted : 10
Insertion Succesfull
Enter Your choice : 3
Enter the element to be inserted : 20
Insertion Succesfull
Enter Your choice : 3
Enter the element to be inserted : 30
Queue is Full
Reallocating Memory
Insertion Succesfull
Enter Your choice : 5
Queue : 10 20 30
Enter Your choice : 2
Deleted Element : 10
Enter Your choice : 4
Deleted Element : 30
Enter Your choice : 5
Queue : 20
Enter Your choice : 6
!! THANK YOU !!
```

11. b) Develop a menu driven program to implement Priority Queue with static and dynamic memory allocation mechanisms using array storage representation. (Represent Queue using structure)

```
//Ascending Priority Queue Static
#include <stdio.h>
```

```
#include <stdlib.h>
#define size 5
struct Queue
{
   int a[size]; //Array to store Queue
   int front; //Index Of front element in Queue
   int rear; //Index of rear element in Queue
}Q;
//Function to check Priority
int prioritycheck(int X)
{
   int i=0;
   while(Q.a[i]<X&&i<=Q.rear)
   {
      i++;
   }
   return i;
}
//Function to insert an element to Queue
void insert(int X)
{
   int i,pos;
   //Checking if Queue is Full
   if(Q.rear==size-1)
   printf("Queue is Full\n");
   else if(Q.rear==-1&&Q.front==-1)
   {
      Q.rear=Q.front=0; //Initializing First element
      Q.a[Q.rear]=X;
   }
   else
   {
      pos = prioritycheck(X);
      Q.rear++;
      for(i=Q.rear;i>=pos;i--)
      {
         Q.a[i]=Q.a[i-1];
      }
      Q.a[pos]=X;
   }
}
//Function to delete the front element of Queue
void delete()
{
   if(Q.front==-1)
      printf("Queue is Empty\n");
   else if(Q.front==Q.rear)
   {
      printf("Deleted Element : %d\n",Q.a[Q.front]);
      Q.front=Q.rear=-1;
```

```
    }
    else
    {
        //Deleting Element from front
        printf("Deleted Element : %d\n",Q.a[Q.front]);
        Q.front++;
    }
}
//Function Display all elements of Queue
void display()
{
    if(Q.front==-1)
        printf("Queue is Empty\n");
    else
    {
        for(int i=Q.front;i<=Q.rear;i++)
        printf("%d ",Q.a[i]);
        printf("\n");
    }
}
void main()
{
    int choice,val;
    Q.front=-1;Q.rear=-1; //Initializing Empty Queue
    printf("Main Menu\n1.Insert\n2.Delete\n3.Display\n4.Exit\n");
    for(;;)
    {
        printf("Enter Your choice : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1 : printf("Enter the element to be inserted : ");
                     scanf("%d",&val);
                     insert(val);
                     break;
            case 2 : delete();
                     break;
            case 3 : display();
                     break;
            case 4 : printf("!! THANK YOU !!\n");
                     exit(0);
            default : printf("Invalid Choice\nEnter Again\n");
        }
    }
}
```

Output:

```
Main Menu
1.Insert
2.Delete
```

```
3.Display
4.Exit
Enter Your choice : 1
Enter the element to be inserted : 10
Enter Your choice : 1
Enter the element to be inserted : 5
Enter Your choice : 1
Enter the element to be inserted : 20
Enter Your choice : 1
Enter the element to be inserted : 15
Enter Your choice : 1
Enter the element to be inserted : 25
Enter Your choice : 1
Enter the element to be inserted : 30
Queue is Full
Enter Your choice : 3
5 10 15 20 25
Enter Your choice : 2
Deleted Element : 5
Enter Your choice : 3
10 15 20 25
Enter Your choice : 4
!! THANK YOU !!
```

```c
//Ascending Priority Queue Dynamic
#include <stdio.h>
#include <stdlib.h>
int size;
```

```
struct Queue
{
   int *a; //Base address of array to store Queue
   int front; //Index Of front element in Queue
   int rear; //Index of rear element in Queue
}Q;
//Function to check Priority
int prioritycheck(int X)
{
   int i=0;
   while(*(Q.a+i)<X&&i<=Q.rear)
   {
      i++;
   }
   return i;
}
//Function to insert an element to Queue
void insert(int X)
{
   int i,pos;
   //Checking if Queue is Full
   if(Q.rear==size-1)
   {
      printf("Queue is Full\n");
      printf("Reallocating Memory\n");
      size++;
      Q.a= (int *)realloc(Q.a,size*sizeof(int));//Reallocating Memory for Queue
   }
   if(Q.rear==-1&&Q.front==-1)
   {
      Q.rear=Q.front=0; //Initializing First element
      *(Q.a+Q.rear)=X;
   }
   else
   {
      pos = prioritycheck(X);
      Q.rear++;
      for(i=Q.rear;i>=pos;i--)
      {
         *(Q.a+i)=*(Q.a+i-1);
      }
      *(Q.a+pos)=X;
   }
}
//Function to delete the front element of Queue
void delete()
{
   if(Q.front==-1)
      printf("Queue is Empty\n");
   else if(Q.front==Q.rear)
```

```
    {
        printf("Deleted Element : %d\n",*(Q.a+Q.front));
        Q.front=Q.rear=-1;
    }
    else
    {
        //Deleting Element from front
        printf("Deleted Element : %d\n",*(Q.a+Q.front));
        Q.front++;
    }
}
//Function Display all elements of Queue
void display()
{
    if(Q.front==-1)
        printf("Queue is Empty\n");
    else
    {
        for(int i=Q.front;i<=Q.rear;i++)
        printf("%d ",*(Q.a+i));
        printf("\n");
    }
}
void main()
{
    int choice,val;
    Q.front=-1;Q.rear=-1; //Initializing Empty Queue
    printf("Enter the Size : ");
    scanf("%d",&size);
    Q.a=(int *)malloc(size*sizeof(int)); //Dynamically Allocating Array for Queue
    printf("Main Menu\n1.Insert\n2.Delete\n3.Display\n4.Exit\n");
    for(;;)
    {
        printf("Enter Your choice : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1 : printf("Enter the element to be inserted : ");
                     scanf("%d",&val);
                     insert(val);
                     break;
            case 2 : delete();
                     break;
            case 3 : display();
                     break;
            case 4 : printf("!! THANK YOU !!\n");
                     exit(0);
            default : printf("Invalid Choice\nEnter Again\n");
        }
    }
```

```
}
```

Output:

```
Enter the Size : 2
Main Menu
1.Insert
2.Delete
3.Display
4.Exit
Enter Your choice : 1
Enter the element to be inserted : 10
Enter Your choice : 1
Enter the element to be inserted : 5
Enter Your choice : 1
Enter the element to be inserted : 15
Queue is Full
Reallocating Memory
Enter Your choice : 3
5 10 15
Enter Your choice : 2
Deleted Element : 5
Enter Your choice : 3
10 15
Enter Your choice : 4
!! THANK YOU !!
```

12. a) Develop a menu driven program to implement binary search tree and traversal techniques.

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

```
struct node
{
   int item;
   struct node *llink;
   struct node *rlink;
};
typedef struct node *NODE;
NODE getnode(int X)
{
   NODE temp;
   temp = (NODE)malloc(sizeof(struct node)); //Dynamically Allocating a new Node
   temp->item = X;
   temp->llink=temp->rlink=NULL;
   return temp;
}
NODE insert(NODE root, int X)
{
   NODE temp = getnode(X);
   if(root==NULL)
   return temp;
   if(X<root->item)
   root->llink = insert(root->llink,X);
   else
   root->rlink = insert(root->rlink,X);
   return root;
}
void inorder(NODE root)
{
   if(root!=NULL)
   {
      inorder(root->llink);
      printf("%d\n",root->item);
      inorder(root->rlink);
   }
}
void preorder(NODE root)
{
   if(root!=NULL)
   {
      printf("%d\n",root->item);
      preorder(root->llink);
      preorder(root->rlink);
   }
}
void postorder(NODE root)
{
   if(root!=NULL)
   {
      postorder(root->llink);
      postorder(root->rlink);
```

```
        printf("%d\n",root->item);
    }
}
void main()
{
    int choice,X;
    NODE root=NULL;
    printf("Menu\n1.Insert\n2.InOrder\n3.PreOrder\n4.PostOrder\n5.Exit\n");
    for(;;)
    {
        printf("Enter Your Choice : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1 : printf("Enter an Element : ");
                     scanf("%d",&X);
                     root = insert(root,X);
                     break;
            case 2 : inorder(root);
                     break;
            case 3 : preorder(root);
                     break;
            case 4 : postorder(root);
                     break;
            case 5 : printf("!!THANK YOU!!\n");
                     exit(0);
            default : printf("Invalid Choice\n");
        }
    }
}
```

Output:

```
 Menu
 1.Insert
 2.InOrder
 3.PreOrder
 4.PostOrder
 5.Exit
 Enter Your Choice : 1
 Enter an Element : 100
 Enter Your Choice : 1
 Enter an Element : 50
 Enter Your Choice : 1
 Enter an Element : 75
 Enter Your Choice : 1
 Enter an Element : 25
 Enter Your Choice : 1
 Enter an Element : 150
 Enter Your Choice : 1
 Enter an Element : 125
 Enter Your Choice : 1
 Enter an Element : 175
 Enter Your Choice : 2
 25
 50
 75
 100
 125
 150
 175
 Enter Your Choice : 3
 100
 50
 25
 75
 150
 125
 175
 Enter Your Choice : 4
 25
 75
 50
 125
 175
 150
 100
 Enter Your Choice : 5
!!THANK YOU!!
```

12.b) Develop a menu driven program to implement Graph traversal techniques.

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

```
int N; //Number of Nodes
//Function for BFS Traversal
void bfs(int a[][N], int visited[], int start)
{
   int Q[N],rear=-1,front=-1; //Initializing Empty Queue
   int i;
   //Initializing the Visited Array with 0
   for(i=0;i<N;i++)
   visited[i]=0;
   //Inserting the First node into Queue
   Q[++rear]=start;
   ++front;
   visited[start]=1;
   while(rear>=front)
   {
      start = Q[front++];
      printf("%c ",start+65);
      for(i=0;i<N;i++)
      {
         if(a[start][i]&&visited[i]==0)
         {
            Q[++rear] = i; //Inserting the Adjacent nodes into Queue
            visited[i] = 1;
         }
      }
   }
}
void dfs(int a[][N], int visited[], int start)
{
   int S[N],top=-1; //Initializing Empty Stack
   int i;
   //Initializing the Visited Array with 0
   for(i=0;i<N;i++)
   visited[i]=0;
   S[++top] = start;
   visited[start]=1;
   while(top!=-1)
   {
      start = S[top--];
      printf("%c ",start+65);
      for(i=0;i<N;i++)
      {
         if(a[start][i]&&visited[i]==0)
         {
            S[++top] = i; //Inserting the Adjacent node into Stack
            visited[i] = 1;
            break;
         }
      }
   }
}
void main()
{
   int matrix[10][10]; //Declaring adjacency matrix of Graph
   int visited[10]; //Declaring the Visited Array
   int choice,i,j;
   char node; //The node from where traversal is started
   printf("Menu\n1.Enter the Graph\n2.BFS Traversal\n3.DFS Traversal\n4.Exit\n");
   for(;;)
   {
      printf("\nEnter Your Choice : ");
      scanf("%d",&choice);
      switch (choice)
      {
      case 1 : printf("Enter the Number of Nodes : ");
             scanf("%d",&N);
             printf("Enter the Adjacency Matrix :\n");
             for(i=0;i<N;i++)
             for(j=0;j<N;j++)
             scanf("%d",&matrix[i][j]);
             break;
      case 2 : printf("The Nodes are named as A,B,C,D...\n");
             printf("Enter the node to start with : ");
             scanf(" %c",&node);
             printf("BFS Traversal : ");
             bfs(matrix,visited,node-65);
             break;
      case 3 : printf("The Nodes are named as A,B,C,D...\n");
             printf("Enter the node to start with : ");
             scanf(" %c",&node);
             dfs(matrix,visited,node-65);
             break;
      case 4 : printf("!!THANK YOU!!\n");
             exit(0);
      default: printf("Invalid Choice\n");
      }
   }
}
```

Output :

```
Menu
1.Enter the Graph
```

```
2.BFS Traversal
3.DFS Traversal
4.Exit

Enter Your Choice : 1
Enter the Number of Nodes : 6
Enter the Adjacency Matrix :
0 1 1 1 1 0
1 0 0 1 0 0
1 0 0 1 1 1
1 1 1 0 0 1
1 0 1 0 0 0
0 0 1 1 0 0

Enter Your Choice : 2
The Nodes are named as A,B,C,D...
Enter the node to start with : A
BFS Traversal : A B C D E F
Enter Your Choice : 3
The Nodes are named as A,B,C,D...
Enter the node to start with : A
A B D C E F
Enter Your Choice : 4
!!THANK YOU!!
```