LAB PROGRAM : 6

Design, Develop and Implement a menu driven Program in C for the following operations on Circular queue of characters
a. Insert an element on to circular QUEUE
b. delete an element on to circular QUEUE
c. Demonstrate overflow and underflow situations on circular QUEUE
d. display the status of the circular QUEUE
e. Exit

```
#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#define SIZE 3


char q[SIZE];

int f = 0, r = -1, count = 0;


void insert_cq()

{ char item;


  if (count == SIZE)

  { printf(" the queue overflow\n");  return;

  }


  printf("Enter the item for insertion\n");

  scanf("\n%c",&item);


  r = (r + 1)%SIZE;

  q[r] = item;

  count++;

}


void delete_cq()

{ if (count == 0)
```

```
 { printf("Queue underflow\n");
  return;
 }
  printf("Element deleted is %c   ",q[f]);
  f = (f + 1) % SIZE;
  count--;
}

void display_cq()
{    int i,j = f;
   if (count == 0)
   { printf("Queue is empty\n");
    return;
   }
   printf(" The contents of queue are");
   for ( i = 1; i <= count; i++)
    {    printf("%c ",q[j]);
         j = ( j + 1)%SIZE;
    }
 }



void main()
{  int ch;
  for(;;)
  { printf("\n1.insert 2.delete 3.display 4: exit\n");
    printf("Enter your choice\n");      scanf("%d",&ch);
    switch(ch)
    {  case 1: insert_cq(); break;
            case 2: delete_cq(); break;
            case 3: display_cq(); break;
```

```
     default : printf("invalid choice\n");
               exit(0);
    }
   }
 }
```

LAB PROGRAM : 7

Design, Develop and Implement a menu driven Program in C for the following operations on single Linked List (SLL) of student Data with the fields:
USN, Name, branch, sem, PhNo
a. Create a SLL of N students Data by using front insertion.
b. Display the status of SLL and count the number of nodes in it
c. Perform Insertion and Deletion at End of SLL
d. Perform Insertion and Deletion at Front of SLL
e.Exit

```
#include<stdio.h>
#include<stdlib.h>
struct sll
{   char usn[10], name[20], branch[20];
    int sem, pno;
    struct sll *next;
};

typedef struct sll node;
node *start = NULL;

node* create()
{ node *new1;
  new1 = (node*) malloc(sizeof(node));
  printf(" Enter usn, name, branch, sem and pno\n ");
      scanf("%s%s%s%d%d",new1->usn,new1->name,new1->branch,&new1->sem,&new1->pno);
  new1->next = NULL;
  return(new1);
}

void insert_front()
{ node *new1;
  new1 = create();
  if (start == NULL)
    start = new1;
  else
   { new1->next = start;
     start = new1;
   }
}

void create_nnodes()
{ int n, i;
  printf("Enter No. of students\n");   scanf("%d",&n);
  for(i = 1; i<=n; i++)
     insert_front();
}

void insert_rear()
{ node *new1, *temp = start;
  new1 = create();
  if (start == NULL)
  {  start = new1;
      return;
  }

while ( temp->next != NULL)
  temp = temp->next;

  temp->next = new1;    }
void delete_front()
{ node *temp = start;

 if (start == NULL)
  {  printf("List is empty\n");
      return;
  }

 if (start ->next == NULL)
  {  printf("Deleted student is = %s",start->usn);
     free(temp);
     start = NULL;
      return;
  }
 start = start->next;
 printf("deleted student is = %s", temp->usn);
 free(temp);
}

void delete_rear()
{ node *temp = start , *prev;

 if (start == NULL)
  {  printf("List is empty\n");
      return;
  }
  if (start ->next == NULL)
  { printf("Deleted student is = %s",start->usn);
      free(temp);
      start = NULL;
      return;
  }
 while(temp->next != NULL)
  {     prev = temp;
        temp = temp->next;
  }
  printf("Deleted student is = %s",temp->usn);
  prev->next = NULL;
  free(temp);
}

void display()
{ node *temp = start; int ct = 0;

 if (start == NULL)
  {  printf("List is empty\n");
      return;
  }
printf("Contents of SLL are \n");


 while(temp != NULL)
  {     printf(" %s %s %s %d %d \n", temp->usn, temp->name, temp->branch, temp->sem, temp
->pno);
        temp = temp->next;
        ct++;
  }

  printf("No. of nodes = %d",ct);
}

void main()
{  int ch;

   for(;;)
    { printf("\n1.create_nnodes 2.display 3.Insert_rear 4.delete_rear 5.insert_front
6.delete_front 7: exit\n");
      printf("enter your choice\n");
      scanf("%d",&ch);
      switch(ch)
      {  case 1:  create_nnodes();  break;
         case 2:  display(); break;
         case 3:  insert_rear(); break;
         case 4:  delete_rear(); break;
         case 5:  insert_front();  break;
         case 6:  delete_front(); break;
                  default: printf("invalid choice\n"); exit(0);
      }
    }
}
```

LAB PROGRAM :  8

Design, Develop and Implement a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields:
SSN, Name, Dept, Designation,
Sal, PhNo
a. Create a DLL of N Employees Data by using end insertion.
b. Display the status of DLL and count the number of nodes in it
c. Perform Insertion and Deletion at End of DLL
d. Perform Insertion and Deletion at Front of DLL
e. Demonstrate how this DLL can be used as Double Ended Queue.
f. Exit

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

struct dll
{
             char ssn[10],name[20],dept[10],design[10];
   int sal, pno;
   struct dll *lptr,*rptr;
};

typedef struct dll node;
node *start = NULL;

node* create()
{
   node *new1;
   new1 = (node*)malloc(sizeof(node));
   printf(" Enter ssn, name,dept,designsal and pno,");
   scanf("%s%s%s%s%d%d",new1->ssn,new1->name,new1->dept,new1->design,&new1
->sal,&new1->pno);
   new1->lptr = new1->rptr = NULL;
   return(new1);
}

void insert_front()
{
   node *new1;
   new1 = create();
   if (start == NULL)
```

```
    start = new1;
    else
    {
       new1->rptr = start;
       start = new1;
    }
}

void insert_rear()
{
    node *new1, *temp = start;
    new1 = create();
    if (start == NULL)
    {
       start = new1;
       return;
    }
    while ( temp->rptr != NULL)
       temp = temp->rptr;
    temp->rptr = new1;
    new1->lptr = temp;
}
void create_nnodes()
{
             int n, i;
             printf("Enter No. of Employees\n");
             scanf("%d",&n);
             for(i = 1; i<=n; i++)
                       insert_rear();
}


void delete_front()
{
    if (start == NULL)
    {
       printf("List is empty\n");
       return;
    }

    if (start ->rptr == NULL)
    {
       printf("deleted record with ssn = %s\n",start->ssn);
       free(start);
       start = NULL;
       return;
    }
    printf("deleted record with ssn = %s\n",start->ssn);
    start = start->rptr;
    free(start->lptr);
    start->lptr = NULL;
}
```

```
void delete_rear()
{
   node *temp = start;
    if (start == NULL)
   {
      printf("List is empty\n");
      return;
   }

   if (start->rptr == NULL)
  {
      printf("deleted record with ssn = %s\n",start->ssn);
      free(start);
      start = NULL;
      return;
   }

   while(temp->rptr != NULL)
      temp = temp->rptr;

   (temp->lptr)->rptr = NULL;
   printf("deleted record with ssn = %s\n",start->ssn);
   free(temp);
}


void display()
{
   node *temp = start; int ct = 0;
   if (start == NULL)
   {
      printf("List is empty\n");
      return;
   }
   printf("Contents of DLL are \n");
   while(temp != NULL)
   {
      printf(" %s %s %s %s %d %d\n ",temp->ssn, temp->name,temp->dept,temp->design,temp->sal,
temp->pno);
      temp = temp->rptr;
      ct++;
   }
   printf("\n no. of nodes = %d",ct);
}

void main()
{
   int ch;
   for(;;)
   {
      printf("\n1.insert_rear 2.delete_front 3.Insert_front 4.delete_rear 5.display 6: exit\n");
      printf("enter your choice\n");
      scanf("%d",&ch);
      switch(ch)
```

```
        {
           case 1: create_nnodes();break;
           case 2: insert_rear();  break;
           case 3: delete_front(); break;
                  case 4: insert_front();  break;
           case 5: delete_rear(); break;
                         case 6: display(); break;
                  default :printf("invalid choice\n"); exit(0);
        }
   }
}
```

LAB  PROGRAM : 9

Design, Develop and Implement a Program in C for the following operations on Singly Circular Linked List (SCLL) with header nodes
a. Represent and Evaluate a Polynomial P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z-2xyz3
b. Find the sum of two polynomials POLY1(x,y,z) and POLY2(x,y,z) and store the result in POLYSUM(x,y,z)
Support the program with appropriate functions for each of the above operations

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

struct poly
{         int cf, px, py, pz, flag;
          struct poly *next;
};

typedef struct poly node;

node* getnode()
{
          node *new1;
          new1 = (node*)malloc(sizeof(node));
          new1->next = new1;
          return new1;
}

void display(node *head)
{
          node *temp = head->next;
          if(head->next == head)
          {         printf("Polynomial does not exist\n");
                    return;
          }

          while(temp != head)
          {
                    printf("\n %d x^%d y^%d z^%d",temp->cf,temp->px,temp->py,temp->pz);
                    if(temp->next != head)
                    printf(" + ");
                    temp=temp->next;
```

```
        }
        printf("\n");
}

node* insert_rear(int f,int x,int y,int z,node *head)
{
        node *new1,*temp;
        new1 = getnode();
        new1->cf = f;
        new1->px = x;
        new1->py = y;
        new1->pz = z;
        new1->flag = 0;
        temp = head->next;
        while(temp->next != head)
        {
                temp = temp->next;
        }
        temp->next = new1;
        new1->next = head;
        return head;
}

node* read_poly(node *head)
{
        int px, py, pz, cf, ch;
        do
        {
                printf("\nEntercoeff: ");
                scanf("%d",&cf);
                printf("\nEnter x, y, z powers(0-indiacate NO term): ");
                scanf("%d%d%d",&px,&py,&pz);
                head = insert_rear(cf,px,py,pz,head);
                printf("\nIf you wish to continue press 1 otherwise 0: ");
                scanf("%d",&ch);
        } while(ch != 0);
        return head;
}

node* add_poly(node *h1, node *h2, node *h3)
{
        node *p1,*p2;
        p1 = h1->next;
        while(p1 != h1)
        {
                p2 = h2->next;
                while(p2 != h2)
                {
                        if( p1->px == p2->px && p1->py == p2->py&& p1->pz==p2->pz)
                        {
                                h3 = insert_rear(p1->cf + p2->cf, p2->px,p2->py, p2->pz,h3);
                                p1->flag = 1;
                                p2->flag = 1;
                                break;
```

```
            }
            p2 = p2->next;
        }
        if ( p1->flag ==0 )
        {
            h3 = insert_rear(p1->cf, p1->px, p1->py, p1->pz, h3);
        }
        p1 = p1->next;
    }
 p2 = h2->next;
    while(p2 != h2)
    {
        if ( p2->flag ==0 )
     {
      h3 = insert_rear(p2->cf, p2->px,p2->py, p2->pz,h3);
     }
        p2 = p2->next;
    }
    return (h3);
}


void evaluate(node *he)
{
    node *temp ;
    int x, y, z;
    float result = 0.0;
    printf("\nEnter x, y, z, terms to evaluate:\n");
    scanf("%d%d%d",&x,&y,&z);
    temp = he->next;
    while(he != temp)
    {
        result = result + (temp->cf * pow(x,temp->px) * pow(y,temp->py) * pow(z,temp
->pz));
        temp = temp->next;
    }
    printf("\nPolynomial result is: %f", result);
}

void main()
{
    node *h1,*h2,*h3,*he;
    int ch;
    while(1)
    {
        printf("\n\n1.Evaluate polynomial\n2.Add two polynomials\n3.Exit\n");
        printf("Enter your choice: ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: he = getnode();
                    printf("\nEnter polynomial to evaluate:\n");
                    he = read_poly(he);
                    display(he);
                    evaluate(he);
                    free(he);
                    break;
            case 2: h1 = getnode();
                    h2 = getnode();
                    h3 = getnode();
                    printf("\nEnter the first polynomial:");
                    h1 = read_poly(h1);
                    printf("\nEnter the second polynomial:");
                    h2 = read_poly(h2);
                    h3 = add_poly(h1,h2,h3);
                    printf("\nFirst polynomial is: ");
                    display(h1);
                    printf("\nSecond polynomial is: ");
                    display(h2);
                    printf("\nThe sum of 2 polynomials is: ");
                    display(h3);
                    break;
            case 3: exit(0);
            default:printf("\nInvalid entry");
                    break;
        }
    }
}
```

LAB PROGRAM :  10

Design, Develop and Implement a menu driven Program in C for the following operations on Binary search tree of integers
a. Create a BST of integers : 6,9,5,2,8,15,24
b. Traverse the BST in inorder, preorder, postorder.
c. Search the BST for a given element (key) and report the appropriate message
d. Delete an element from BST
e. Exit

```
#include<stdio.h>

#include<stdlib.h>


struct bst

{  int item;

   struct bst *lptr, *rptr;

};

typedef struct bst node;


node* insert(node *root)

{  node *new1, *cur = root, *prev= NULL;

   new1 = (node *)malloc(sizeof(node));

   printf("\nEnter The Element ");

         scanf("%d",&new1->item);

   new1->lptr = new1->rptr = NULL;

   if (root == NULL)

        return new1;
```

```
  while(cur != NULL)
  {  prev = cur;
    cur = new1->item < cur->item ?
        cur->lptr : cur->rptr;
  }
  if (new1->item < prev->item)
          prev->lptr = new1;
  else
          prev->rptr = new1;
  return root;
}

void inorder(node *root)
{  if (root != NULL)
  {  inorder(root->lptr);
    printf("%d\t", root->item);
    inorder(root->rptr);
  }
}

void preorder(node *root)
{  if (root != NULL)
  {  printf("%d\t", root->item);
    preorder(root->lptr);
    preorder(root->rptr);
  }
}

void postorder(node *root)
{  if (root != NULL)
  {  postorder(root->lptr);
```

```
      postorder(root->rptr);
      printf("%d\t", root->item);
   }
}

int FindMin(node *root)
{  node *cur = root;
   if (root == NULL)
      return -1;
   while(cur->lptr != NULL)
         cur = cur->lptr;

   return cur->item;
}

node* Delete(node *root, int data)
{            node *temp;     int min;
             if (root == NULL)
             { printf("tree is empty\n");
               return NULL;
             }
             // data is in the left sub tree.
             if (data  <  root->item)
             {       root->lptr = Delete(root->lptr, data);
                          return(root);
             }
             // data is in the right sub tree.
             if (data  >  root->item)
             {   root->rptr = Delete(root->rptr, data);
                 return(root);
             }
```

```
           // data is present but no children
          if (root->lptr == NULL && root->rptr == NULL)
          {    printf("deleted data %d",root->item);
               free(root);   root = NULL;
                   return(root);
          }
           // data is present but no right subtree
          if (root->rptr == NULL)
          {  temp = root->lptr;
             printf("deleted data %d",root->item);
         free(root); return(temp);
          }
           // data is present but no left subtree
           if (root->lptr == NULL)
           {     temp = root->rptr;
                  printf("deleted info %d",root->item);
                 free(root); return(temp);
          }
          // If both left and right subtree are present, find the min element
          // in right subtree and place it in root node and call delete function
          //for right subtree.
           min = FindMin(root->rptr);
           root->item = min;
           root->rptr = Delete(root->rptr, min);

          return(root);

}



node* search(node *root, int key)
```

```
{           node *cur = root;
            if(root == NULL)
                        return NULL;

            while(cur != NULL)
            {           if(key == cur->item)
                                    return cur;
                        if(key < cur->item)
                                    cur = cur->lptr;
                        else
                                    cur = cur->rptr;
            }
            return(cur);
}


int main()
{  int choice, key,n,i;
   node *root = NULL, *temp, parent;
while(1)
{    printf("\n 1.Create");
     printf("\n 2.Traverse the Tree in Pre, In,Post");
     printf("\n 3.Search");
     printf("\n 4.Delete an element from the Tree");
     printf("\n 5.Exit");
     printf("\nEnter your choice :");       scanf("%d",&choice);
switch (choice)
{  case 1:   printf("\n enter no. of nodes");
                scanf("%d",&n);
            for(i=0;i<n;i++)
                        root = insert(root);  break;
```

```
 case 2:  if (root == NULL)
      printf("Tree Is Not Created");
         else
         { printf("\nThe Inorder Traversal : ");
           inorder(root);
          printf("\nThe Preorder Traversal: ");
           preorder(root);
          printf("\nThe Postorder Traversal : ");
           postorder(root);
         }  break;
case 3: printf("\nEnter Element to be searched :");
        scanf("%d",&key);
        temp = search(root,key);
        if(temp == NULL)
       printf("Element does not exists\n");
        else
           printf("\nThe element %d  found",temp->item);
             break;
case 4: printf("\nEnter Element to be deleted :");
             scanf("%d", &key);
             root = Delete(root,key); break;
default: exit(0);
}
  }
    }
```

## LAB PROGRAM : 11

Design, Develop and Implement a menu driven Program in C for the following operations on Graphs GI of cities
a. Create a graph of N cities using adjacency matrix
b. Print all the nodes reachable from a given starting node in a diagraph using DFS/ BFS method

```
#include <stdio.h>
#include <stdlib.h>
int a[20][20],q[20],visited[20],reach[10],n,i,j,f=0,r= -1,count=0;
void bfs(int v)
{
  for(i=1;i<=n;i++)
    if(a[v][i] && !visited[i])
       q[++r]=i;
  if(f <= r)
  {
    visited[q[f]]=1;
    bfs(q[f++]);
  }
}

void dfs(int v)
{
  int i;
  reach[v]=1;
  for(i=1;i<=n;i++)
```

```
 {
   if(a[v][i] && !reach[i])
   {
       printf("\n %d->%d",v,i);
       count++;
       dfs(i);
   }
 }
}

void main()
{ int v, choice;
  printf("\n Enter the number of vertices:");
  scanf("%d",&n);
  for(i=1;i<=n;i++)
  {
    q[i]=0;
    visited[i]=0;
  }

  for(i=1;i<=n-1;i++)
    reach[i]=0;

  printf("\n Enter graph data in matrix form:\n");
  for(i=1;i<=n;i++)
   for(j=1;j<=n;j++)
      scanf("%d",&a[i][j]);
 for(;;)
 {
   printf("\n 1.BFS\n 2.DFS\n 3.Exit\n");
   scanf("%d",&choice);
   switch(choice)
   {case 1:
     printf("\n Enter the starting vertex:");
     scanf("%d",&v);
     bfs(v);
     if( v < 1 || v > n )
        printf("\n Bfs is not possible");
     else
     {  printf("\n The nodes which are reachable from %d:\n",v);
        for(i=1; i<=n; i++)
        if( visited[i] )
          printf("%d\t",i);
     } break;
    case 2:
        dfs(1);
        if(count==n-1)
          printf("\n Graph is connected");
        else
          printf("\n Graph is not connected");
        break;
    default: printf("Invalid Choice\n");  exit(0);
   }
 }
}
```

## LAB PROGRAM : 12

Give a file of n employee records with a set k of keys (4-digit) which uniquely determine the records in the file F. Assume that file F is maintained in memory by a hash table of M memory locations with L as the set of memory addresses of locations in hash table let the key in K and addresses in L are integers . Design and develop a program in C that uses hash functions.

```
#include<stdio.h>

#include<stdlib.h>

FILE *fp;

struct employee

{    char name[20];

      int key,salary;

} emp[20];

int n,m,*ht,index,count = 0;


void insert(int key)

{               index = key % m;

                while(ht[index] != -1)

                { printf("\ncollision detected for %d and resloved using linear probing",key);

                        index = (index+1)%m;

                }

                ht[index] = key;

                count++;

 }
```

```
void display()
{           int i;
            if(count == 0)
            {    printf("\nHash Table is empty");
                         return;
            }

            printf("\nHash Table contents are:\n ");
            for(i=0; i<m; i++)
                        printf("\n T[%d] --> %d ", i, ht[i]);
}

void main()
{          int i;
           printf("\nEnter the number of employee  records (N) :   ");
           scanf("%d", &n);

           printf("\nEnter the two digit memory locations (m) for hash table:   ");
           scanf("%d",&m);

           ht = (int *)malloc(m*sizeof(int));
           for(i=0; i<m; i++)
                    ht[i] = -1;
           fp=fopen("C:\\PROGRAM1.txt","w");
           printf("\nEnter  four digit key,name,salary values (K) for N Emp Records:\n  ");
           for(i=0; i<n; i++)
           {            scanf("%d%s%d",&emp[i].key,emp[i].name,&emp[i].salary);
                        fprintf(fp,"%d\t%s\t%d\n",emp[i].key,emp[i].name,emp[i].salary);
           }
           fclose(fp);
           fp=fopen("C:\\PROGRAM1.txt","r");
```

```
       for(i=0;i<n;i++)
       {          if(count == m)
                   {  printf("\n~~~Hash table is full. Cannot insert record %d key~~~",i+1);
                              break;
                   }
                   fscanf(fp,"%d",&emp[i].key);
                   insert(emp[i].key);
       }
      fclose(fp);
         //Displaying Keys inserted into hash table
          display();
}
```