Experiment 1:

Design, Develop and Implement a menu driven Program in C for the following Array operations

a. Creating an Array of N Integer Elements
b. Display of Array Elements with Suitable Headings
c. Inserting an Element (ELEM) at a given valid Position (POS)
d. Deleting an Element at a given valid Position(POS)
e. Exit.

Support the program with functions for each of the above operations

```
#include<stdio.h>
#include<stdlib.h>
int a[20];
int n,val,i,pos;

/*Function Prototype*/
void create();
void display();
void insert();
void delete();
int main()
{
int choice; while(choice)
{
printf("\n\n--------MENU-----------\n");
printf("1.CREATE\n2.DISPLAY\n 3.INSERT\n 4.DELETE\n 5.EXIT\n ");
printf("\nENTER YOUR CHOICE:\t");
scanf("%d",&choice);
switch(choice)
{
case 1: create();
break;
case 2: display();
break;
case 3:insert();
break;
case 4: delete();
break;
case 5: exit(0);
break;
default: printf("\nInvalid choice:\n");
break;
 }
}
return 0;
}
//creating an array
void create()
```

```
{
printf("\nEnter the size of the array elements:\t");
scanf("%d",&n);
printf("\nEnter the elements for the array:\n");
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
}
//displaying an array elements void
display()
{
int i;
printf("\nThe array elements are:\n");
for(i=0;i<n;i++)
{
printf("%d\t",a[i]);
}
}
//inserting an element
into an array void
insert()
{
printf("\nEnter the position for the new element:\t");
scanf("%d",&pos);
printf("\nEnter the element to be inserted :\t");
scanf("%d",&val);
for(i=n-1;i>=pos;i--)
{
a[i+1]=a[i];
}
a[pos]=val;
n=n+1;
}
//deleting an array element void delete()
{
printf("\nEnter the position of the element to be deleted:\t");
scanf("%d",&pos);
val=a[pos];
for(i=pos;i<n-1;i++)
{
a[i]=a[i+1];
}
n=n-1;
printf("\nThe deleted element is =%d",val);
}
```

Experiment 2:

Design, Develop and Implement a Program in C for the following operations on Strings

a. Read a main String (STR), a Pattern String (PAT) and a Replace String (REP)

b. Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR. Report suitable messages in case PAT does not exist in STR.

Support the program with functions for each of the above operations. Don't use Built-in functions.

```
#include<stdio.h>
void main()
{
        char STR[100],PAT[100],REP[100],ans[100];
        int i,j,c,m,k,flag=0;
        printf("\nEnter the MAIN string: \n");
        gets(STR);
        printf("\nEnter a PATTERN string: n");
        gets(PAT);
         printf("\nEnter a REPLACE string: \n");
         gets(REP);
          i = m = c = j = 0;
          while ( STR[c] != '\0')
         {
         // Checking for Match
               if ( STR[m] == PAT[i] )
               {
                     i++;
                     m++;
                     flag=1;
                     if ( PAT[i] == '\0')
                     {
                       //copy replace string in ans string
                      for(k=0; REP[k] != '\0';k++,j++)
                      ans[j] = REP[k];
                       i=0;
                       c=m;
                     }
                }
                else //mismatch
                {
                  ans[j] = STR[c];
                  j++;
                  c++;
                  m = c;
                  i=0;
                 }
          }
        if(flag==0)
                  printf("Pattern doesn't found!!!");
        else
        {
                  ans[j] = '\0';
                  printf("\nThe RESULTANT string is:%s\n" ,ans);
         }}
```

Experiment 3.

Design, Develop and Implement a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX)

a.Push an Element on to Stack
b.Pop an Element from Stack
c.Demonstrate how Stack can be used to check Palindrome
d.Demonstrate Overflow and Underflow situations on Stack
e.Display the status of Stack
f.Exit

Support the program with appropriate functions for each of the above operations

```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#define max_size 5
int stack[max_size],top=-1;
//Prototyping
void push();
void pop();
void display();
void pali();
void main()
{
int choice;
while(1)
{
clrscr();
printf("\n\n--------STACK OPERATIONS-----------\n");
printf("1.Push\n");
printf("2.Pop\n");
printf("3.Palindrome\n");
printf("4.Display\n");
printf("5.Exit\n");
printf("-----------------------");
printf("\nEnter your choice:\t");
scanf("%d",&choice);
switch(choice)
{
        case 1:push();
        break;
        case 2: pop();
        break;
        case 3: pali();
        break;
        case 4: display();
        break;
        case 5: exit(0);
        break;
        default:
        printf("\nInvalid hoice:\n");
        break;
}
}
```

```
}
void push() //Inserting element into the stack
{
int item,n;
if(top==(max_size-1))
printf("\nStack Overflow:");
else
{
printf("Enter the element to be inserted:\t");
scanf("%d",&item);
top=top+1;
stack[top]=item;
}
}
void pop() //deleting an element from the stack
{
int item;
if(top==-1)
printf("Stack Underflow:");
else
{
item=stack[top];
top=top-1;
printf("\nThe poped element: %d\t",item);
}
}
void pali()
{
int digit,j,k,len=top+1,flag=0,ind=0;
int num[10],rev[10],i=0,length=0;
while(top!=-1)
{
digit= stack[top];
num[i]=digit;
top--;
i++;
}
printf("\nNumbers are :");
for(j=0;j<len;j++)
{
printf("%d\t",num[j]);
}
printf("\nReverse operation : ");
for(k=len-1;k>=0;k--)
{
rev[k]=num[ind];
ind++;
}
printf("\nReverse array : ");
for(k=0;k<len;k++)
{
printf("%d\t",rev[k]);
}
printf("\nCheck for palindrome");
```

```
for(i=0;i<len;i++)
{
  if(num[i]==rev[i])
     length = length+1;
}
if(length==len)
printf("\nIt is palindrome number\n");
else
printf("\nIt is not a palindrome number\n");
top = len-1;
}
void display()
{
int i;
if(top==-1)
{
printf("\nStack is Empty:");
}
else
{
printf("\nThe stack elements are:\n");
for(i=top;i>=0;i--)
printf("%d\n",stack[i]);
}
}
```

Experiment 4.

Design,Develop and Implement a Program in C for converting an Infix Expression to Postfix Expression.Program should support for both parenthesized and free parenthesized expressions with the operators: +,-,*,/,%,^(Power) and alphanumeric operands.*/

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
 int F(char symbol)
{
 switch(symbol)
{
 case '+' :
 case '-': return 2;
  case '*':
 case '/': return 4;
  case '^':
 case '$': return 5;
  case '(': return 0;
  case '#': return -1;
  default: return 8;
 }
}

int G(char symbol)
{
 switch(symbol)
{
 case '+':
 case '-': return 1;
  case '*':
 case '/': return 3;
  case '^':
 case '$': return 6;
  case '(': return 9;
  case ')': return 0;
  default: return 7;
 }
}
void infix_postfix(char infix[], char postfix[])
{
 int top, j, i;
 char s[30], symbol;
 top = -1;
 s[++top] = '#';
 j = 0;
 for(i=0; i < strlen(infix); i++)
 {
 symbol = infix[i];
 while(F(s[top]) > G(symbol))
 {
 postfix[j] = s[top--];
 j++;
 }
 if(F(s[top]) != G(symbol))
```

```
s[++top] = symbol;
else
top--;
}
while(s[top] != '#')
{
postfix[j++] = s[top--];
}
postfix[j] = '\0';
}
void main()
{
char infix[20], postfix[20];
clrscr();
printf("\nEnter a valid infix expression\n");
flushall();
gets(infix);
infix_postfix(infix,postfix);
printf("\nThe infix expression is:\n");
printf ("%s",infix);
printf("\nThe postfix expression is:\n");
printf ("%s",postfix);
getch();
}
```

Experiment 5

Design, Develop and Implement a Program in C for the following Stack Applications
a. Evaluation of Suffix expression with single digit operands and operators:+, -, *, /, %, ^
b. Solving Tower of Hanoi problem with n disks

//Experiment 5A: Evaluation of Suffix expression with single digit operands and operators:+, -, *, /, %, ^

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<string.h>

double compute(char symbol, double op1, double op2)
{
 switch(symbol)
 {
 case '+': return op1 + op2;
 case' -': return op1 - op2;
 case '*': return op1 * op2;
 case '/': return op1 / op2;
 case '$':
 case '^': return pow(op1,op2);
 default: return 0;
 }
}
void main()
{
 double s[20], res, op1, op2;
 int top, i;
 char postfix[20], symbol;
 clrscr();
 printf("\nEnter the postfix expression:\n");
flushall();
 gets(postfix);
 top=-1;
 for(i=0;i<strlen(postfix); i++)
 {
 symbol = postfix[i];
 if(isdigit(symbol))
 s[++top] = symbol - '0';
 else
 {
 op2 = s[top--];
 op1 = s[top--];
 res = compute(symbol, op1, op2);
 s[++top] = res;
 }
 }
 res = s[top--];
 printf("\nThe result is : %f\n", res);
 getch();
}
```

Program 5B: Tower of Honoi

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

void tower(int n, int source, int temp,int destination)
{
 if(n == 0)
 return;
 tower(n-1, source, destination, temp);
 printf("\nMove disc %d from %c to %c", n, source, destination);
 tower(n-1, temp, source, destination);
}

void main()
{
 int n;
 clrscr();
 printf("\nEnter the number of discs: \n");
 scanf("%d", &n);
 tower(n, 'A', 'B', 'C');
printf("\n\nTotal Number of moves are: %d", (int)pow(2,n)-1);
 getch();
}
```

Experiment 6:

Design, Develop and Implement a menu driven Program in C for the following operations on **Circular QUEUE** of Characters (Array Implementation of Queue with maximum size **MAX**)

a. Insert an Element on to Circular QUEUE
b. Delete an Element from Circular QUEUE
c. Demonstrate ***Overflow*** and ***Underflow*** situations on Circular QUEUE
d. Display the status of Circular QUEUE
e. Exit

Support the program with appropriate functions for each of the above operations

```
#include<stdio.h>
#define MAX  4
int ch, front = 0, rear = -1, count=0;
char q[MAX], item;
void insert()
{
        if(count == MAX) printf("\nQueue is Full");
        else {
                rear = (rear + 1) % MAX;
                q[rear]=item;
                count++;
        }
}
void del()
{
        if(count == 0)
                printf("\nQueue is Empty");
        else
        {
                 if(front > rear && rear==MAX-1)
                {
                        front=0;
                        rear=-1;
                        count=0;
                }
                else
                {
                        item=q[front];
                        printf("\nDeleted item is: %c",item);
                        front = (front + 1) %MAX;
                        count--;
                }
         }
}
void display()
{
        int i, f=front, r=rear;
        if(count == 0)
                printf("\nQueue is Empty");
        else
        {
                printf("\nContents of Queue is:\n");
                for(i=f; i<=r; i++)
```

```
                {
                        printf("%c\t", q[i]);
                        f = (f + 1) % MAX;
                }
        }
}
void main()
{
        do
        {
                printf("\n1. Insert\n2. Delete\n3. Display\n4. Exit");
                printf("\nEnter the choice: ");
                scanf("%d", &ch);
                flushall();
                switch(ch)
                {
                        case 1: printf("\nEnter the character / item to be inserted: ");
                                scanf("%c", &item);
                                insert();
                                break;
                        case 2: del();
                        break;
                        case 3: display();
                                 break;
                        case 4: exit(0);
                                   break;
                }
        }while(ch!=4);
}
```

Experiment 7:

Design, Develop and Implement a menu driven Program in C for the following perations on Singly 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. Demonstrate how this SLL can be used as STACK and QUEUE
f. Exit

```
#include<stdio.h>
#include<conio.h>
int MAX=4, count;
struct student
{
        char usn[10];
        char name[30];
        char branch[5];
        int sem;
        char phno[10];
        struct student *next;
 //Self referential pointer.
};
typedef struct student NODE;

int countnodes(NODE *head)
{
        NODE *p;
        count=0;
        p=head;
        while(p!=NULL)
        {
                p=p->next;
                count++;
        }
        return count;
}

NODE* getnode(NODE *head)
{
        NODE *newnode;
        newnode=(NODE*)malloc(sizeof(NODE));
        //Create first NODE
        printf("\nEnter USN, Name, Branch, Sem, Ph.No\n");
        flushall();
        gets(newnode->usn);
        flushall();
        gets(newnode->name);
        flushall();
        gets(newnode->branch);
        scanf("%d",&(newnode->sem));
        flushall();
        gets(newnode->phno);
```

```
        newnode->next=NULL;
        //Set next to NULL...
        head=newnode;
        return head;
}
NODE* display(NODE *head)
{
        NODE *p;
        if(head == NULL)
                printf("\nNo student data\n");
        else
        {
                p=head;
                printf("\n----STUDENT DATA----\n");
                printf("\nUSN\tNAME\t\tBRANCH\tSEM\tPh.NO.");
                while(p!=NULL)
                {
                printf("\n%s\t%s\t\t%s\t%d\t%s",p->usn, p->name, p->branch, p->sem, p->phno);
                p = p->next; //Go to next node...
                }
                printf("\nThe no. of nodes in list is: %d",countnodes(head));
        }
        return head;
}
NODE* create(NODE *head)
{
        NODE *newnode;
        if(head==NULL)
        {
                newnode=getnode(head);
                head=newnode;
        }
        else
        {
                newnode=getnode(head);
                newnode->next=head;
                head=newnode;
        }
        return head;
}
NODE* insert_front(NODE *head)
{
        if(countnodes(head)==MAX)
                printf("\nList is Full //Overflow!!");
        else
                head=create(head); //create()insert nodes at front.
                return head;
}
NODE* insert_rear(NODE *head)
{
        NODE *p, *newnode;
        p=head;
        if(countnodes(head) == MAX)
                printf("\nList is Full(QUEUE)!!");
        else
```

```
        {
                if(head == NULL)
                {
                        newnode=getnode(head);
                        head=newnode;          //set first node to be head
                }
                else
                {
                        newnode=getnode(head);
                        while(p->next!=NULL)
                        {
                                p=p->next;
                        }
                        p->next=newnode;
                }
        }
        return head;
}
NODE* insert(NODE *head)
{
        int ch;
        do
        {
                printf("\n1.Insert at Front(First)\t2.Insert at End(Rear/Last)\t3.Exit");
                printf("\nEnter your choice: ");
                scanf("%d", &ch);
                 switch(ch)
                {
                        case 1: head=insert_front(head);
                                break;
                        case 2: head=insert_rear(head);
                                break;
                        case 3: break;
                }
                head=display(head);
        }while(ch!=3);
        return head;
}
NODE* delete_front(NODE *head)
{
        NODE *p;
        if(head==NULL)
                printf("\nList is Empty/Underflow (STACK/QUEUE)");
        else
        {
                p=head;
                head=head->next; //Set 2nd NODE as head free(p);
                printf("\nFront(first)node is deleted");
        }
        return head;
}

NODE* delete_rear(NODE *head)
{
```

```
        NODE *p, *q; p=head;
        while(p->next!=NULL)
        {
                p=p->next; //Go upto -1 NODE which you want to delete
        }
        q=p->next;
        free(q);//Delete last NODE...
        p->next=NULL;
        printf("\nLast(end) entry is deleted");
        return head;
}
NODE* del(NODE *head)
{
        int ch;
        do{
             printf("\n1.Delete from Front(First)\t2. Delete from End(Rear/Last))\t3.Exit");
             printf("\nEnter your choice: ");
             scanf("%d",&ch);
                switch(ch)
                {
                        case 1: head=delete_front(head);
                        break;
                        case 2: head=delete_rear(head);
                        break;
                        case 3: break;
                }
                head=display(head);
        }while(ch!=3);
        return head;
}

NODE* stack(NODE *head)
{
        int ch;
        do
        {
                printf("\nSSL used as Stack...");
                printf("\n 1.Insert at Front(PUSH) \t 2.Delete from Front(POP))\t3.Exit");
                printf("\nEnter your choice: ");
                scanf("%d", &ch);
                switch(ch)
                {
                        case 1: head=insert_front(head);
                                break;
                        case 2: head=delete_front(head);
                                break;
                        case 3: break;
                }
                head=display(head);
        }while(ch!=3);
        return head;
}
NODE* queue(NODE *head)
{
```

```
        int ch;
        do
        {
                printf("\nSSL used as Queue...");
                printf("\n 1.Insert at Rear(INSERT) \t 2.Delete from
                Front(DELETE))\t3.Exit");
                printf("\nEnter your choice: ");
                scanf("%d", &ch);
                switch(ch)
                {
                        case 1: head=insert_rear(head);
                                break;
                        case 2: head=delete_front(head);
                                break;
                        case 3: break;
                }
                head=display(head);
        }while(ch!=3);
        return head;
}
void main()
{
        int ch, i, n;
        NODE *head;
        head=NULL;
        clrscr();
        printf("\n*----------Studednt Database----------*");
        do
        {
          printf("\n 1.Create\t 2.Display\t 3.Insert\t 4.Delete\t 5.Stack\t 6.Queue\t 7.Exit");
          printf("\nEnter your choice: ");
          scanf("%d", &ch);
                switch(ch)
                {
                        case 1: printf("\nHow many student data you want to create: ");
                                scanf("%d", &n);
                                for(i=0;i<n;i++)
                                        head=create(head);//Call to Create NODE...
                                break;
                        case 2: head=display(head);  //Call to Display...
                                break;
                        case 3: head=insert(head);      //Call to Insert...
                                break;
                        case 4: head=del(head);                 //Call to delete
                                break;
                        case 5: head=stack(head);
                                break;
                        case 6: head=queue(head);
                                break;
                        case 7: exit();                 //Exit...
                        }
                }while(ch!=7);
        }
```

Experiment 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>
int MAX=4,count;
struct emp
{
        int ssn;
        char name[20], dept[10], desig[15], phno[10];
        int sal;
        struct emp *left;
        struct emp *right;
};
typedef struct emp NODE;

int countnodes(NODE *head)
{
  NODE *p;
  count=0;
  p=head;
  while(p!=NULL)
  {
        p=p->right;
        count++;
  }
 return count;
}

NODE* getnode(NODE *head)
{
   NODE *newnode;
   newnode=(NODE*)malloc(sizeof(NODE));
   newnode->right=newnode->left=NULL;
   printf("\nEnter SSN, Name, Dept, Designation, Sal, Ph.No\n");
   scanf("%d",&newnode->ssn);
   flushall();
   gets(newnode->name);
   flushall();
   gets(newnode->dept);
   flushall();
   gets(newnode->desig);
   scanf("%d",&newnode->sal);
   flushall();
   gets(newnode->phno);
   head=newnode;
```

```
   return head;
}

NODE* display(NODE *head)
{
  NODE *p;
  if(head==NULL)
        printf("\nNo Employee data\n");
  else
  {
        p=head;
        printf("\n----EMPLOYEE DATA----\n");
        printf("\nSSN\tNAME\tDEPT\tDESINGATION\tSAL\t\tPh.NO.");
        while(p!=NULL)
        {
       printf("\n%d\t%s\t%s\t%s\t\t%d\t\t%s",p->ssn,p->name,p->dept,p->desig,p->sal,p->phno);
        p = p->right;    //Go to next node...
        }
   printf("\nThe no. of nodes in list is: %d",countnodes(head));
  }
  return head;
}

NODE* create(NODE *head) // creating & inserting at end.
{
   NODE *p, *newnode;
   p=head;
   if(head==NULL)
   {
        newnode=getnode(head);
        head=newnode;
   }
   else
   {
        newnode=getnode(head);
        while(p->right!=NULL)
        {
                p=p->right;
        }
        p->right=newnode;
        newnode->left=p;
    }
    return head;
}

NODE* insert_end(NODE *head)
{
        if(countnodes(head)==MAX)
            printf("\nList is Full!!");
        else
            head=create(head);
        return head;
}
```

```
NODE* insert_front(NODE *head)
{
    NODE *p, *newnode;
    if(countnodes(head)==MAX)
        printf("\nList is Full!!");
    else
    {
        if(head==NULL)
        {
            newnode=getnode(head);
            head=newnode;    //set first node to be head
        }
        else
        {
            newnode=getnode(head);
            newnode->right=head;
            head->left=newnode;
            head=newnode;
        }
    }
    return head;
}

NODE* insert(NODE *head)
{
    int ch;
    do
    {
        printf("\n 1.Insert at Front(First) \t 2.Insert at End(Rear/Last)\t3.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch(ch)
        {
                case 1: head=insert_front(head);
                        break;
                case 2: head=insert_end(head);
                        break;
                case 3: break;
        }
        head=display(head);
     }while(ch!=3);
     return head;
}

NODE* delete_front(NODE *head)
{
        NODE *p; if(head==NULL)
        printf("\nList is Empty (QUEUE)");
        else
        {
                p=head;
                head=head->right;
                head->right->left=NULL;
                free(p);
```

```
                printf("\nFront(first)node is deleted");
        }
        return head;
}

NODE* delete_end(NODE *head)
{
NODE *p, *q;
p=head;
while(p->right!=NULL)
{
p=p->right;     //Go upto -1 node which you want to delete
}
q=p->left;
q->right=NULL;
p->left=NULL;
free(p);//Delete last node...
printf("\nLast(end) entry is deleted");
return head;
}

NODE *del(NODE *head)
{
    int ch;
    do {
          printf("\n1.Delete from Front(First)\t2.Delete from End(Rear/Last))\t3.Exit");
          printf("\nEnter your choice: ");
          scanf("%d", &ch);
          switch(ch)
          {
            case 1: head=delete_front(head);
                    break;
           case 2: head=delete_end(head);
                    break;
            case 3: break;
          }
          head=display(head);
        }while(ch!=3);
        return head;
}

NODE* queue(NODE *head)
{
   int ch, ch1, ch2;
   do
   {
        printf("\nDLL used as Double Ended Queue");
        printf("\n1.QUEUE- Insert at Rear & Delete from Front");
        printf("\n2.QUEUE- Insert at Front & Delete from Rear");
        printf("\n3.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch(ch)
        {
```

```
            case 1:
            do
            {
                  printf("\n1.Insert at Rear\t2.Delete from From Front\t3.Exit");
                  printf("\nEnter your choice: ");
                  scanf("%d", &ch1);
                  switch(ch1)
                  {
                    case 1: head=insert_end(head);
                              break;
                    case 2: head=delete_front(head);
                              break;
                    case 3: break;
                  }
                 }while(ch1!=3);
                  break;
                  case 2:
                  do{
                        printf("\n1.Insert at Front\t2.Delete from Rear\t3.Exit");
                        printf("\nEnter your choice: ");
                        scanf("%d", &ch2); switch(ch2)
                        {
                          case 1: head=insert_front(head);
                                    break;
                          case 2: head=delete_end(head);
                                    break;
                          case 3: break;
                         }
                       }while(ch2!=3);
                       break;
                  case 3: break;
          }
}while(ch!=3);
head=display(head); return head;
}
void main()
{
int ch, i, n;
NODE *head;
head=NULL;
clrscr();
printf("\n----------Employee Database----------");
do
{
printf("\n1.Create\t2.Display\t3.Insert\t4.Delete\t5.Queue\t6.Exit");
printf("\nEnter your choice: ");
scanf("%d", &ch); switch(ch)
{
case 1: printf("\nHow many employees data you want to create: ");
          scanf("%d", &n);
          for(i=0;i<n;i++)
          head=create(head);//Call to Create node...
          break;
```

```
case 2: head=display(head);  //Call to Display...
        break;
case 3: head=insert(head);      //Call to Insert...
        break;
case 4: head=del(head);         //Call to delete
        break;
case 5: head=queue(head);
        break;
case 6: exit(0);            //Exit...
        break;
}
}while(ch!=6);

}
```

Experiment 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<alloc.h>
#include<math.h>
struct node
{
        int cf, px, py,
        pz;
        int flag;
        struct node *link;
};
typedef struct node NODE;

NODE* getnode()
{
        NODE *x;
        x=(NODE*)malloc(sizeof(NO
        DE)); if(x==NULL)
        {
                printf("Insufficient memory\n");
                exit(0);
        }
        return x;
}

void display(NODE *head)
{
        NODE *temp;
        if(head->link==head)
        {
                printf("Polynomial does not exist\n");
                return;
        }
        temp=head-
        >link;
        printf("\n");
        while(temp!
        =head)
        {
                printf("%d x^%d y^%d z^%d",temp->cf,temp->px,temp->py,temp->pz);
                if(temp->link != head)
                printf(" + ");
                temp=temp->link;
        }
        printf("\n");
}


NODE* insert_rear(int cf,int x,int y,int z,NODE *head)
```

```
{
        NODE *temp,*cur; temp=getnode();
        temp->cf=cf;
        temp->px=x;
        temp->py=y;
        temp->pz=z;
        cur=head->link;
        while(cur->link!=head)
        {
                cur=cur->link;
        }
        cur->link=temp;
        temp->link=head;
        return head;
}

NODE* read_poly(NODE *head)
{
        int px, py, pz, cf, ch;
        printf("\nEnter coeff: ");
        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)
        {
                printf("\nEnter coeff: ");
                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);
        }
        return head;
}

NODE* add_poly(NODE *h1,NODE *h2,NODE *h3)
{
        NODE *p1,*p2;
        int x1,x2,y1,y2,z1,z2,cf1,cf2,cf;
        p1=h1->link;
        while(p1!=h1)
        {
                x1=p1->px;  y1=p1->py;  z1=p1->pz;  cf1=p1->cf;
                p2=h2->link;
                while(p2!=h2)
                {
                        x2=p2->px;  y2=p2->py;  z2=p2->pz;  cf2=p2->cf;
                        if(x1==x2 && y1==y2 && z1==z2) break;
                        p2=p2->link;
                }
                if(p2!=h2)
                {
```

```
                cf=cf1+cf2;
                        p2->flag=1;
                        if(cf!=0)
                                h3=insert_rear(cf,x1,y1,z1,h3);
                }
                else
                        h3=insert_rear(cf1,x1,y1,z1,h3);
                        p1=p1->link;
        }
        p2=h2->link;
        while(p2!=h2)
        {
                if(p2->flag==0)
                h3=insert_rear(p2->cf, p2->px, p2->py, p2->pz, h3);
                p2=p2->link;
        }
        return h3;
}

void evaluate(NODE *h1)
{
        NODE *head;
        int x, y, z;
        float
        result=0.0;
        head=h1;
        printf("\nEnter x, y, z, terms to evaluate:\n");
        scanf("%d%d%d", &x, &y, &z);
        while(h1->link != head)
        {
            result=result+(h1->cf*pow(x,h1->px) * pow(y,h1->py) * pow(z,h1->pz));
             h1=h1->link;
        }
         result = result + (h1->cf * pow(x,h1->px) * pow(y,h1->py) * pow(z,h1->pz));
         printf("\nPolynomial result is: %f", result);
}

void main()
{
        NODE *h1,*h2,*h3;
        int ch;
        clrscr();
        h1=getnode();
        h2=getnode();
        h3=getnode();
        h1->link=h1;
        h2->link=h2;
        h3->link=h3;
        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:  printf("\nEnter polynomial to evaluate:\n");
                         h1=read_poly(h1);
                         display(h1);
                         evaluate(h1);
                         break;
                case 2:   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);
                          break;
                default:  printf("\nInvalid
                entry"); break;
                }
                getch();
        }
}
```

Experiment - 10

Design, Develop and Implement a menu driven Program in C for the following operations on **Binary Search Tree (BST)** of Integers

a. Create a BST of **N** Integers: **6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2**
b. Traverse the BST in Inorder, Preorder and Post Order
c. Search the BST for a given element (**KEY**) and report the appropriate message
d. Delete an element (**ELEM**) from BST
e. Exit

```
#include <stdio.h>
#include <stdlib.h>
struct BST
{
        int data;
        struct BST *left;
        struct BST *right;
};
typedef struct BST NODE;
NODE *node;

NODE* createtree(NODE *node, int data)
{
   if (node == NULL)
   {
        NODE *temp;
        temp= (NODE*)malloc(sizeof(NODE));
        temp->data = data;
        temp->left = temp->right = NULL;
        return temp;
   }
    if (data < (node->data))
    {
        node->left = createtree(node->left, data);
    }
   else if (data > node->data)
    {
        node -> right = createtree(node->right, data);
    }
   return node;
}

NODE* search(NODE *node, int data)
{
  if(node == NULL)
  printf("\nElement not found");
  else if(data < node->data)
  {
        node->left=search(node->left, data);
  }
  else if(data > node-->data)
  {
node->right=search(node->right, data);
}
else
```

```
printf("\nElement found is: %d", node->data);
return node;
}

void inorder(NODE *node)
{
   if(node != NULL)
   {
            inorder(node->left);
            printf("%d\t", node->data);
            inorder(node->right);
   }
}

void preorder(NODE *node)
{
   if(node != NULL)
   {
            printf("%d\t", node->data);
            preorder(node->left);
            preorder(node->right);
   }
}


void postorder(NODE *node)
{
  if(node != NULL)
  {
            postorder(node->left);
            postorder(node->right);
            printf("%d\t", node->data);
  }
}

NODE* findMin(NODE *node)
{
 if(node==NULL)
  {
            return NULL;
   }
  if(node->left)
            return findMin(node->left);
   else
    return node;
}


NODE* del(NODE *node, int data)
{
    NODE *temp;
     if(node == NULL)
     printf("\nElement not found");
     else if(data < node->data)
```

```
    {
       node->left = del(node->left, data);
    }
    else if(data > node->data)
    {
          node->right = del(node->right, data);
    }
    else
{//Now We can delete this node and replace with either minimum element in the right sub tree or maximum element in the left subtree
    if(node->right && node->left)
    {       /* Here we will replace with minimum element in the right sub tree */
       temp = findMin(node->right); node -> data = temp->data;
          /* As we replaced it with some other node, we have to delete that node */
       node -> right = del(node->right,temp->data);
    }
    else
    {
/* If there is only one or zero children then we can directly remove it from the tree and connect its parent to its child */
          temp = node;
          if(node->left == NULL)
          node = node->right;
          else if(node->right == NULL)
          node = node->left;
          free(temp);      /* temp is longer required */
    }
  }
  return node;
}

void main()
{
   int data, ch, i, n;
   NODE *root=NULL;
   clrscr();
   while (1)
   {
     printf("\n1.Insertion in Binary Search Tree");
     printf("\n2.Search Element in Binary Search Tree");
     printf("\n3.Delete Element in Binary Search Tree");
     printf("\n4.Inorder\n5.Preorder\n6.Postorder\n7.Exit");
     printf("\nEnter your choice: ");
     scanf("%d", &ch);
     switch (ch)
     {
        case 1: printf("\nEnter N value: " );
          scanf("%d", &n);
          printf("\nEnter the values to create BST like(6,9,5,2,8,15,24,14,7,8,5,2)\n");
          for(i=0; i<n; i++)
          {
                scanf("%d", &data);
                root=createtree(root, data);
          }
          break;
```

```
            case 2: printf("\nEnter the element to search: ");
                    scanf("%d", &data);
                    root=search(root, data);
                    break;
            case 3: printf("\nEnter the element to delete: ");
                    scanf("%d", &data);
                    root=del(root, data);
                    break;
            case 4: printf("\nInorder Traversal: \n");
                    inorder(root);
                    break;
            case 5: printf("\nPreorder Traversal: \n");
                    preorder(root);
                    break;
            case 6: printf("\nPostorder Traversal: \n");
                    postorder(root);
                    break;
            case 7: exit(0);
            default: printf("\nWrong option");
                    break;
            }
    }
    getch();
}
```

Experiment - 11

Design, Develop and Implement a Program in C for the following operations on **Graph(G)** 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 digraph using **BFS** method
c. Check whether a given graph is **connected** or not using **DFS** method.

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

int a[10][10], n, m, i, j, source, s[10], b[10];
int visited[10];

void create()
{
        printf("\nEnter the number of vertices of the digraph: ");
        scanf("%d", &n);
        printf("\nEnter the adjacency matrix of the graph:\n");
        for(i=1; i<=n; i++)
                for(j=1; j<=n; j++)
                     scanf("%d", &a[i][j]);
}

void bfs()
{
        int q[10], u, front=0, rear=-1;
        printf("\nEnter the source vertex to find other nodes
        reachable or not: "); scanf("%d", &source);
        q[++rear] = source;
        visited[source] = 1;
        printf("\nThe reachable vertices are: ");
        while(front<=rear)
        {
                u = q[front++];
                for(i=1; i<=n; i++)
                {
                        if(a[u][i] == 1 && visited[i] == 0)
                        {
                            q[++rear] = i;
                            visited[i] = 1;
                            printf("\n%d", i);
                        }
                }
        }
}

void dfs(int source)
{
        int v, top = -1;
        s[++top] = 1;
        b[source] = 1;
        for(v=1; v<=n; v++)
        {
```

```
                if(a[source][v] == 1 && b[v] == 0)
                {
                        printf("\n%d -> %d", source, v);
                        dfs(v);
                }
        }
}

void main()
{
        int ch;
        clrscr();
        while(1)
        {
           printf("\n1.Create Graph\n2.BFS\n3.Check graph connected or not(DFS)\n4.Exit");
           printf("\nEnter your choice: ");
           scanf("%d", &ch);
           switch(ch)
           {
              case 1: create(); break;
              case 2: bfs();
                      for(i=1;i<=n;i++)
                         if(visited[i]==0)
                            printf("\nThe vertex that is not rechable %d" ,i);
                                break;
              case 3: printf("\nEnter the source vertex to find the connectivity: ");
                      scanf("%d",&source);
                      m=1;
                      dfs(source);
                      for(i=1;i<=n;i++)
                      {
                           if(b[i]==0)
                               m=0;
                      }
                      if(m==1)
                           printf("\nGraph is Connected");
                      else
                           printf("\nGraph is not Connected");
                           break;
             default: exit(0);
           }
        }
}
```

Experiment - 12

Given a File of **N** employee records with a set **K** of Keys(4-digit) which uniquely determine the records in file **F**. Assume that file **F** is maintained in memory by a Hash Table(HT) of **m** memory locations with **L** as the set of memory addresses (2-digit) of locations in HT. Let the keys in **K** and addresses in **L** are Integers. Design and develop a Program in C that uses Hash function **H: K ®L** as H(**K**)=**K** mod **m (remainder** method), and implement hashing technique to map a given key **K** to the address space **L.** Resolve the collision (if any) using **linear probing**.

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 10

struct employee
{
        int id;
        char name[15];
};
typedef struct employee EMP;
EMP emp[MAX];
int a[MAX];

int create(int num)
{
        int key;
        key = num % 100;
        return key;
}

int getemp(EMP emp[],int key)
{

        printf("\nEnter emp id: ");
        scanf("%d",&emp[key].id);
        printf("\nEnter emp name: ");
        flushall();
        gets(emp[key].name);
        return key;
}

void display()
{
        int i, ch;
        printf("\n1.Display ALL\n2.Filtered Display");
        printf("\nEnter the choice: ");
        scanf("%d",&ch);
        if(ch == 1)
        {
                printf("\nThe hash table is:\n");
                printf("\nHTKey\tEmpID\tEmpName");
                for(i=0; i<MAX; i++)
                        printf("\n%d\t%d\t%s", i, emp[i].id, emp[i].name);
        }
        else
        {
```

```
                printf("\nThe hash table is:\n");
                printf("\nHTKey\tEmpID\tEmpName");
                for(i=0; i<MAX; i++)
                        if(a[i] != -1)
                        {
                                printf("\n%d\t%d\t%s", i, emp[i].id, emp[i].name);
                                continue;
                        }
        }
}

void linear_prob(int key, int num)
{
        int flag, i, count = 0;
        flag = 0;
        if(a[key] == -1)
        {
                a[key]=getemp(emp, key);
        }
        else
        {
                printf("\nCollision Detected...!!!\n");
                i = 0; while(i < MAX)
                {
                        if (a[i] != -1)
                                count++;
                        else
                                i++;
                }
        printf("\nCollision avoided successfully using LINEAR PROBING\n");
        if(count == MAX)
        {
                printf("\n Hash table is full");
                display(emp);
                exit(1);
        }
        for(i=key; i<MAX; i++)
           if(a[i] == -1)
           {
                        a[i] = num;
                        flag = 1; break;
                }
                i = 0;
                while((i < key) && (flag == 0))
                {
                        if(a[i] == -1)
                        {
                                a[i] = num;
                                flag=1;
                                break;
                        }
                        i++;
                } // end while
          } // end else
```

```
} // end linear_prob()

void main()
{
        int num,
        key, i; int
        ans = 1;
        clrscr();
        printf("\nCollision handling by linear probing: ");
        for (i=0; i < MAX; i++)
        {
                a[i] = -1;
        }
        do
        {
                printf("\nEnter the data: ");
                scanf("%d", &num);
                key=create(num);
                linear_prob(key,num);
                printf("\nDo you wish to continue? (1/0): ");
                scanf("%d",&ans);
        }while(ans);
        display(emp);
        getch();
}
```