EXPERIMENT: 1

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. Exit.
Support the program with functions for each of the above operations.

ABOUT THE EXPERIMENT:
An Array is a collection of similar /same elements. In this experiment the array can be represented as one / single dimensional elements.

ALGORITHM:
Step 1: Start.
Step 2: Read N value.
Step 3: Read Array of N integer elements
Step 4: Print array of N integer elements.
Step 5: Stop.

PROGRAM CODE:

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

int a[10],n ;

void create();
void display();

void create()
{
        int i;
        Printf("Enter the size of array (<= 10)\n");
        scanf("%d",&n);
        Printf("Enter the elements of array\n");
        for(i=0; i<n; i++)
                scanf("%d",&a[i]);
}

void display()
{
        int i;
        Printf("The array elements are:\n");
        for(i=0; i<n; i++)
         Printf("%d\t",a[i]);
}

int main()
{
 int ch;
 while(ch)
 {
  Printf("\n\n_____MENU_____\n");
  Printf("1.Create\n2.Display\n3.Exit\n");
  Printf("Enter Your Choice:");
```

```
  scanf("%d",&ch);
  switch(ch)
  {
   case 1:create();break;
   case 2:display();break;
   case 3:exit(0);break;
   default :Printf("INVALID CHOICE\n");
  }
 }
return 0;
}
```

Output –

```
_____MENU_____
1.Create
2.Display
3.Exit
Enter Your Choice:1
Enter the size of array (<=10)
3
Enter the elements of array
1 2 3
_____MENU_____
1.Create
2.Display
3.Exit
Enter Your Choice:2
The array elements are:
1        2        3

_____MENU_____
1.Create
2.Display
3.Exit
Enter Your Choice:3
```

EXPERIMENT: 2

Design, Develop and Implement a menu driven Program in C for the following Array operations
a. Inserting an Element (ELEM) at a given valid Position (POS)
b. Deleting an Element at a given valid Position POS)
c. Display of Array Elements
d. Exit.
Support the program with functions for each of the above operations

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

int n,a[50];

void create()
{
   int i;
   Printf("enter the value of n\n");
   scanf("%d",&n);
   Printf("enter %d array elements\n",n);
   for(i=0;i<n;i++)
   scanf("%d",&a[i]);
}
void display()
{
   int i;
   Printf("entered elements are\n");
   for(i=0;i<n;i++)
   Printf("%d\n",a[i]);
}

void insertion()
{
   int i,POS,ELEM;
   Printf("enter the position and its value\n");
   scanf("%d%d",&POS,&ELEM);
   if ( POS < 1 || POS > n+1)
      Printf("Invalid position") ;

   for (i=n-1; i>=POS-1; i--)
            a[i+1]=a[i];
   a[POS-1]=ELEM;
   n=n+1;
   display();
}

void deletion()
{
int i,POS,ELEM;
Printf("enter the position to be deleted\n");
scanf("%d",&POS);
if ( POS < 1 || POS > n)
      Printf("Invalid position") ;

ELEM=a[POS+1];
for(i=POS-1;i<n-1;i++)
a[i]=a[i+1];
Printf("the deleted element is %d\n",ELEM);
```

```
        n=n-1;
        display();
        }
        void main()
        {
        int ch;
        while(1)
        {
           Printf("enter your choice\n");
        Printf("1.creat\n2.display\n3.insertion\n4.deletion\n5.exit\n");
        scanf("%d",&ch);
        switch(ch)
        {
        case 1: create();
        break;
        case 2: display();
        break;
        case 3: insertion();
        break;
        case 4: deletion();
        break;
        case 5: exit(0);
        }
        }
}
```

Output –

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 Overflow and Underflow situations on Stack
d. Display the status of Stack
e. Exit

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

#define STACKSIZE 5
int s[STACKSIZE], top=-1;

void push()
{
   if(top==STACKSIZE-1)
      Printf("\nStack overflow!!!!");
   else
   {
      Printf("\nEnter element to insert:");
      scanf("%d",&s[++top]);
   }
}

void pop()
{
   if(top==-1)
      Printf("\nStack underflow!!!");
   else
      Printf("\nElement popped is: %d",s[top--]);
}

void disp()
{
   int t=top;
   if(t==-1)
      Printf("\nStack empty!!");
   else
      Printf("\nStack elements are:\n");
   while(t>=0)
      Printf("%d ",s[t--]);
}

int main()
{
   int ch;
   do
   {
      Printf("\n...Stack operations.....\n");
      Printf("1.PUSH\n");
      Printf("2.POP\n");
      Printf("3.Display\n");
      Printf("4.Exit\n_____\n");
      Printf("Enter choice:");
      scanf("%d",&ch);
```

```
            switch(ch)
            {
               case 1:push();break;
               case 2:pop();break;
               case 3:disp();break;
               case 4:exit(0);
               default:Printf("\nInvalid choice");
            }
         }
         while(1);
         return 0;

}
```

Output –

EXPERIMENT 4

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

```
#include <stdio.h>
 #include <conio.h>
#include <math.h>
#define MAX 20

struct stack
{
int top;
float str[MAX];
} s;//stack

char postfix[MAX];//postfix void push(float);
float pop();
int isoperand(char);
float operate(float,float,char);

 int main()
{
int i=0;

Printf("Enter Expression:");
scanf("%s",postfix);

float ans,op1,op2;

 while (postfix[i]!='\0')
{
if(isoperand(postfix[i]))
push(postfix[i]-48);

 else
{
op1=pop();
op2=pop();
ans=operate(op1,op2,postfix[i]);
push(ans);
Printf("%f %c %f = %f\n",op2,postfix[i],op1,ans);
}
 i++;
}
Printf("%f",s.str[s.top]); getch();
}

int isoperand(char x)
{
if(x>='0' && x<='9')
return 1;
else
 return 0;
}
```

```
 void push(float x)
{
if(s.top==MAX-1)
Printf("Stack is full\nStack overflow\n");
else
{
s.top++;
s.str[s.top]=x;
}
}

float pop()
{
if(s.top==-1)
{
Printf("Stack is emplty\nSTACK UNDERFLOW\n");
getch();
}
else
{
s.top--;
return s.str[s.top+1];
}
}

float operate(float op1,float op2,char a)
{
switch(a)
{
case '+' : return op2+op1;
case '-' : return op2-op1;
case '*' : return op2*op1;
case '/' : return op2/op1;
case '^' : return pow(op2,op1);
}

}
```

4b.

```
#include <stdio.h>
#include <conio.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();
}
```

Sample Output 1

```
Insert a postfix notation:: 22^32*+
Result :: 10
```

Sample Output 2

```
Insert a postfix notation :: 23+

Result :: 5
```

Sample Output 1

```
Enter the number of discs: 3

Move disc 1 from A to C
Move disc 2 from A to B
Move disc 1 from C to B
Move disc 3 from A to C
Move disc 1 from B to A
Move disc 2 from B to C
Move disc 1 from A to C

Total Number of moves are: 7
```

5. Singly Linked List (SLL) of Integer Data a. Create a SLL stack of N integer. b. Display of SLL c. Linear search

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

 struct node
{
   int info;
   struct node *link;
 } ;
typedef struct node *NODE ;

NODE getnode() ;
NODE insert_front(int item, NODE first) ;
NODE delete_front(NODE first) ;
NODE search(int key, NODE first) ;
 void display(NODE first);

  NODE getnode()
 {
     NODE x ;
      x = (NODE) malloc(sizeof(struct node)) ;
      if ( x== NULL)
     {
          printf("out of memory\n") ;
          exit(0) ;
      }
 }

 NODE insert_front(int item, NODE first)
 {
     NODE temp ;
     temp = getnode() ;
     temp->info = item ;
     temp->link = first ;
     return temp ;
}
```

```
NODE delete_front(NODE first)
{
    NODE temp ;
    if ( first == NULL)
    {
        printf("List is empty cannot delete \n") ;
        return first ;
     }
     temp = first ;
     temp = temp->link ;
    printf("Item deleted = %d\n, first->info) ;
    free(first);
    return temp ;
}

void  search(int key, NODE first)
{
       NODE cur ;
       if ( first == NULL)
       {
           printf("List is empty\n") ;
           return ;
        }
        cur = first ;
        while ( cur !=NULL)
         {
            if ( key == cur->info) break ;
            cur = cur->link ;
          }
         if ( cur == NULL)
        {
              printf("Search is unsuccessful\n") ;
              return ;
        }
        printf("Search is successful\n") ;
}


void display(NODE first)
```

```
{
        NODE temp ;
        If (first == NULL)
        {
              printf("List is empty\n") ;
              return ;
         }
         printf(" The contents of singly linked list\n");
         temp = first ;
         while ( temp != NULL)
         {
              printf("%d", temp->info) ;
               temp = temp->link ;
          }
           printf("\n") ;
}

void main()
{
  NODE first ;
  int choice, item ;
 first  = NULL ;
 for (;;)
 {
        printf("1. Push Item\n   2. Pop \n") ;
       printf("3. Search\n") ;
        printf("4. Display\n       5. Exit\n")  ;
        printf("Enter the choice\n") ;
        scanf("%d", &choice) ;
        switch(choice)
        {
              case 1:   printf("Enter the item to be inserted\n") ;
                        scanf("%d", &item) ;
                        first = insert_front(item, first) ;
                        break ;

             case 2: first = delete_front(first) ;
                       break ;
```

```
            case 3: printf("Enter the key to be searched \n") ;
                    scanf("%d", &item) ;
                    search(item, first) ;
                    break ;

            case 4: display(first) ;
                     break ;
            default:
                     exit(0) ;
        }
    }
}
```

6. Design, Develop and Implement a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Professor Data with the fields: ID, Name, Branch, Area of specialization.
   a. Create a DLL stack of N Professor's Data
   b. Create a DLL queue of N Professor's Data

   Display the status of DLL and count the number of nodes in it

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

struct node
{
   int ProfID;
   char  name[30] ;
   char  branch[3] ;
   char  aos[30] ;
   struct node *llink;
   struct node *rlink;
} ;
typedef struct node *NODE ;
NODE Professor =  NULL , tail = NULL;

NODE getnode()
{
    NODE x ;
     x = (NODE) malloc(sizeof(struct node)) ;
     if ( x== NULL)
     {
         printf("out of memory\n") ;
         exit(0) ;
      }
}

  void create(int ID, char n[30], char brn[3], char specialization[30])
  {
     NODE newnode = NULL;

     newnode = getnode() ;
     newnode->ProfID = ID ;
     strcpy(newnode->name, n);
     strcpy(newnode->branch, brn) ;
     strcpy(newnode->aos, specialization) ;

     newnode->llink = newnode->rlink = NULL ;

     if  (Professor == NULL)
     {
         Professor = tail = newnode ;
    }
    else
    {
              tail->rlink = newnode;
             newnode->llink = tail ;
```

```
            tail = newnode ;

        }
    }


  void Display(int Stack)
   {
        NODE temp ;
        printf("The Professor list -\n") ;

        if ( Stack == 1)
        {
             temp = tail ;
             while (temp  !=NULL )
             {
                 printf("%d\n %s\n\n", temp-> ProfID, temp-> name) ;
                 temp = temp->llink ;
             }
         }
          else
         {
              temp = Professor ;
              while (temp  !=NULL)
              {
                 printf("%d\n %s\n\n", temp-> ProfID, temp-> name) ;
                 temp = temp->rlink ;
              }
         }
    }


    void CountNodes()
    {
         int count = 0 ;
         NODE temp ;
         temp = Professor ;
         while ( temp != NULL)
         {
            count++ ;
             temp = temp->rlink;
         }
          printf("Number of Professors are %d\n", count) ;
   }

    void main()
    {
      int choice, ID ;
      char name[30], branch[3], specialization[30] ;

     for (;;)
     {
          printf("1. Create Professors \n ") ;
          printf("2. Display Professor Stack\n") ;
          printf("3. Display Professor Queue       4. Count of Professors\n   5. Exit")  ;
          printf("Enter the choice\n") ;
          scanf("%d", &choice) ;
```

```
        switch(choice)
        {
              case 1:   printf("Enter professor ID, name, branch & specialization\n") ;
                        scanf("%d%s%s%s", &ID, name, branch, specialization) ;
                        create(ID, name, branch, specialization) ;
                        break ;

              case 2: Display (1) ;
                        break ;

              case 3: Display(0) ;
                        break ;

              case 4: CountNodes() ;
                        break ;

              default:
                        exit(0) ;
         }
      }
   }
```

7. Given an array of elements, construct a complete binary tree from this array in level order fashion. That is , elements from left in the array will be filled in the tree level wise starting from level 0. Ex – Input

   arr[] = { 1, 2,3,4,5,6}

```
// Recursive C program for level  order traversal of Binary Tree
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *left, *right;
};

void printCurrentLevel(struct node* root, int level);
int height(struct node* node);
struct node* newNode(int data);

void printLevelOrder(struct node* root)
{
    int h = height(root);
    int i;
    for (i = 1; i <= h; i++)
        printCurrentLevel(root, i);
}

void printCurrentLevel(struct node* root, int level)
{
    if (root == NULL)
        return;
    if (level == 1)
        printf("%d ", root->data);
    else if (level > 1) {
        printCurrentLevel(root->left, level - 1);
        printCurrentLevel(root->right, level - 1);
    }
}
```

```
int height(struct node* node)
{
    if (node == NULL)
        return 0;
    else {
        /* compute the height of each subtree */
        int lheight = height(node->left);
        int rheight = height(node->right);

        /* use the larger one */
        if (lheight > rheight)
            return (lheight + 1);
        else
            return (rheight + 1);
    }
}

struct node* newNode(int data)
{
    struct node* node
        = (struct node*)malloc(sizeof(struct node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;

    return (node);
}

int main()
{
    struct node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);

    printf("Level Order traversal of binary tree is \n");
    printLevelOrder(root);
```

```
    return 0;
}
```

Output -

Level Order traversal of binary tree is

1 2 3 4 5

8. 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 b. Traverse the BST in Inorder, Preorder and Post Order.

```
#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;
}


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);
          }

        }


        void main()
        {
        int data, ch, i, n;
        NODE *root=NULL;
        clrscr();
        while (1)
        {
        printf("\n1.Insertion in Binary Search Tree");
        printf("\n2.Inorder\n3.Preorder\n4.Postorder\n7.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
         switch (ch)
        {
        case 1:
                    printf("\nEnter number of elements: " );
                    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("\nInorder Traversal: \n");
                    inorder(root);
                    break;

        case 3:   printf("\nPreorder Traversal: \n");
                    preorder(root);
                    break;

        case 4:
                    printf("\nPostorder Traversal: \n");
                    postorder(root);
                    break;

        case 5: ex  e
exit(0) ;
}
}
}
```

Output -

9. 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 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 main()
{
int v ;
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]);

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);
}
}
```

Output –

```
Enter the number of vertices:
5
Enter graph data in matrix form:
0 1 0 1 0
1 0 1 0 1
0 1 0 1 0
1 0 1 0 0
0 1 0 0 0
Enter the starting vertex:1
The nodes which are reachable from 1:
1    2       3          4          5
```

10. Design and develop a program in C that uses Hash function H:K->L as H(K)=K mod m (reminder 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 100
int create(int);
void display (int[]);
void main()
{
int a[MAX],num,key,i;
int ans=1;
printf(" collision handling by linear probing : \n");
for (i=0;i<MAX;i++)
{
a[i] = -1;
}
do
{
printf("\n Enter the data");
scanf("%4d", &num);
key=create(num);
linear_prob(a,key,num);
printf("\n Do you wish to continue ? (1/0) ");
scanf("%d",&ans);
}while(ans);
display(a);
}
int create(int num)
{
int key;
key=num%100;
return key;
}

void linear_prob(int a[MAX], int key, int num)
{
int flag, i, count=0;
 flag=0;

 if(a[key]== -1)
{
a[key] = num;
}
else
{
printf("\nCollision Detected...!!!\n");
i=0;
while(i<MAX)
{
if (a[i]!=-1)
count++;
i++;
}
printf("Collision avoided successfully using LINEAR PROBING\n");
if(count == MAX)
```

```
{
printf("\n Hash table is full");
display(a);
exit(1);
}
for(i=key+1; i<MAX; i++)
if(a[i] == -1)
{
a[i] = num;
flag =1;
break;
}
//for(i=0;i<key;i++)
i=0;
while((i<key) && (flag==0))
{
if(a[i] == -1)
{

a[i] = num;
flag=1;
break;
}
 i++;
}
}
}
void display(int a[MAX])
{
int i,choice;
printf("1.Display ALL\n 2.Filtered Display\n");
scanf("%d",&choice);
if(choice==1)
{
printf("\n the hash table is\n");
for(i=0; i<MAX; i++)
 printf("\n %d %d ", i, a[i]);
}
else
{
printf("\n the hash table is\n");
for(i=0; i<MAX; i++)
if(a[i]!=-1)
{
printf("\n %d %d ", i, a[i]);
continue;
}
}
}
```

Output -

collision handling by linear probing :

Enter the data1234

Do you wish to continue ? (1/0) 1 Enter the data2548
Do you wish to continue ? (1/0) 1 Enter the data3256
Do you wish to continue ? (1/0) 1 Enter the data1299
Do you wish to continue ? (1/0) 1 Enter the data1298

Do you wish to continue ? (1/0) 1 Enter the data1398
Collision Detected...!!!
Collision avoided successfully using LINEAR PROBING

Do you wish to continue ? (1/0) 0 1.Display ALL
2.Filtered Display 2
the hash table is 0 1398
34 1234
48 2548
56 3256
98 1298
99 1299