S.No	NAME OF THE EXPERIMENT
01	a)Write C program that use both recursive and non recursive functions to perform Linear search for a Key value in a given list.
02	b) Write C program that use both recursive and non recursive functions to perform Binary search for a Key value in a given list.
03	a) Write C program that implement Bubble sort, to sort a given list of integers in ascending order
04	b) Write C program that implement Quick sort, to sort a given list of integers in ascending order
05	c) Write C program that implement Insertion sort, to sort a given list of integers in ascending order
06	a) Write C program that implement radix sort, to sort a given list of integers in ascending order
07	b) Write C program that implement merge sort, to sort a given list of integers in ascendingorder
08	a) Write a C program that uses functions to create a singly linked list. b) Write a C program that uses functions to perform insertion operation on a singly linkedlist c) Write a C program that uses functions to perform deletion operation on a singly linked list
09	d) Write a C program to reverse elements of a single linked list.
10	a) Write C program that implement Queue (its operations) using arrays.
11	b) Write C program that implement Queue (its operations) using linked lists
12	a) Write C program that implement stack (its operations) using arrays
13	b) Write C program that implement stack (its operations) using Linked list
14	c) Write a C program that uses Stack operations to evaluate postfix expression.

15	d) Write a recursive C program for traversing a binary tree in preorder, inorder and postorder.
16	a) Write a C program to Create a BST
17	b) Write a C program to insert a node into a BST.
18	c) Write a C program to delete a node from a BST.

1.Exercise-1(searching)

AIM: write a c program for non recursive linear search.

Program:

```
#include<stdio.h>
int linearsearch(int a[],int n,int key)
{
int i; for(i=0;i<n;i++)
{
if(a[i]==key)
{
return i;
}
}
return -1;
}
int main()
{
int a[100],n,key,pos,i;
printf("enter n value");
scanf("%d",&n);
printf("enter %d elements into array",n);
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
printf("enter search key elements");
scanf("%d",&key); pos=linearsearch(a,n,key);
if(pos>=0)
printf("element found at %d position",pos+1);else
printf("element not found");return 1
}
```

Output:

```
Enter the size of an array 6
Enter the array elements 50 10 5 200 20 1Enter the key
element 1
The key Element is found at location 6
```

Aim:write a c program using recursive linear search.Program:

```
#include <stdio.h>
int LinearSearch(int arr[], int search,int index,int n);int main()
{
int n, search, result, m = 0, arr[100];printf("Program on
Linear Search\n"); printf("    \n");
printf("Enter the total elements in the array\n");scanf("%d", &n);
printf("Enter the array elements\n");for (int i = 0; i < n;
i++)
{
scanf("%d", &arr[i]);
}
printf("Check whether these are the elements which you haveentered\n");
for (int i = 0; i < n; i++)
{
printf("%d\n", arr[i]);
}
printf("Enter the element to search \n");scanf("%d", &search);
result = LinearSearch(arr, search,0,n);if (result != 0)
{
printf("Element found at pos %d\n ", result);
}

else
{
printf("Element not found");
}
return 0;
}
int LinearSearch(int arr[], int search,int index,int n)
{
int arrpos=0;if(index>=n)
{
return 0;
}
if (arr[index] == search)
{
arrpos = index + 1;return arrpos;
}
else
{
return LinearSearch(arr, search,index+1,n);
}
return arrpos;
}
```

Output:

```
Enter the total elements in the array 6Enter the array elements
    4
    6
    1
    2
    5
    3
    Enter the element to search 6Element found at pos 2
```

1. **AIM:b)write a c program for BINARY SEARCH using Recursive andNon-recursive.**

Program:

```
#include <stdio.h> #define
MAX_LEN 10
/* Non-Recursive function*/
void b_search_nonrecursive(int l[],int num,int ele)
 {
 int l1,i,j, flag = 0;l1 =
 0;
 i = num-1;
while(l1 <= i)
 {
 j = (l1+i)/2; if( l[j]
 == ele)
 {
 printf("\nThe element %d is present at position %d in list\n",ele,j);flag
 =1;
 break;
 }
 else
 if(l[j] < ele)l1 =
 j+1;
 else
 i = j-1;
 }
 if( flag == 0)
 printf("\nThe element %d is not present in the list\n",ele);
 }
 /* Recursive function*/
  int b_search_recursive(int l[],int arrayStart,int arrayEnd,int a)
  {
  int m,pos;
```

```
if (arrayStart<=arrayEnd)
{
m=(arrayStart+arrayEnd)/2;
if (l[m]==a)
return m;
else if (a<l[m])
return b_search_recursive(l,arrayStart,m-1,a);
else
return b_search_recursive(l,m+1,arrayEnd,a);
}
return -1;
}
void read_list(int l[],int n)
{
int i;
printf("\nEnter the elements:\n");
for(i=0;i<n;i++)
scanf("%d",&l[i]);
}
void print_list(int l[],int n)
{
int i;
for(i=0;i<n;i++)
printf("%d\t",l[i]);
}
/*main function*/
main()
{
int l[MAX_LEN], num, ele,f,l1,a;
int ch,pos;
//clrscr();
printf("======================================================");
printf("\n\t\t\tMENU");
printf("\n=====================================================");
printf("\n[1] Binary Search using Recursion method");
printf("\n[2] Binary Search using Non-Recursion method");
printf("\n\nEnter your Choice:");
scanf("%d",&ch);
if(ch<=2 & ch>0)
{
printf("\nEnter the number of elements : ");
scanf("%d",&num);
read_list(l,num);
printf("\nElements present in the list are:\n\n");
print_list(l,num);
printf("\n\nEnter the element you want to search:\n\n");
scanf("%d",&ele);
switch(ch)
{
case 1:printf("\nRecursive method:\n");
pos=b_search_recursive(l,0,num,ele);
if(pos==-1)
{
printf("Element is not found");
}
else
{
printf("Element is found at %d position",pos);
}
//getch();
break;
case 2:printf("\nNon-Recursive method:\n");
b_search_nonrecursive(l,num,ele);
//getch();
break;
}
}
//getch();
}
```

output:

```
[1]  Binary Search using Recursion method
[2]  Binary Search using Non-Recursion methodEnter your
Choice:1
Enter the number of elements : 5Enter the elements:
12
22
32
42
52
Elements present in the list are:12 22 32 42 52
Enter the element you want to search:32
Recursive method:
Element is found at 3 position.
```

EXERCISE-2(SORTING-1)

AIM:a)write a C program for bubble sort in ascending order.Program:

```
#include<stdio.h>int
main(){
int count, temp, i, j, number[30];
printf("How many numbers are u going to enter?: ");scanf("%d",&count);
printf("Enter %d numbers: ",count);for(i=0;i<count;i++)
scanf("%d",&number[i]);
/* This is the main logic of bubble sort algorithm
*/
for(i=count-2;i>=0;i--){
for(j=0;j<=i;j++){
if(number[j]>number[j+1]){
temp=number[j];
number[j]=number[j+1];
number[j+1]=temp;
}
}
}
printf("Sorted elements: ");
for(i=0;i<count;i++) printf("
%d",number[i]); return 0;
}
```

Output:

```
Enter the value of num6
Enter the elements one by one23
45
67
89
12
34
Input array is23
45
67
89
12
34
Sorted array is...12
23
34
45
67
89
```

AIM: b)Write a C program to implement QUICK SORT in ascending order.
Program:

```
#include<stdio.h>
void quicksort(int number[25],int first,int last){int i, j, pivot, temp;
if(first<last){
pivot=first; i=first;
j=last; while(i<j){
while(number[i]<=number[pivot]&&i<last)i++;
while(number[j]>number[pivot])j--;
if(i<j){ temp=number[i];
number[i]=number[j];
number[j]=temp;
}
}
temp=number[pivot];
number[pivot]=number[j];
number[j]=temp;
quicksort(number,first,j-1);
quicksort(number,j+1,last);
}
}
int main(){
int i, count, number[25];
printf("How many elements are u going to enter?: ");scanf("%d",&count);
printf("Enter %d elements: ", count);
for(i=0;i<count;i++) scanf("%d",&number[i]);
quicksort(number,0,count-1); printf("Order of Sorted
elements: ");for(i=0;i<count;i++)
printf(" %d",number[i]);return 0;
}
```

Output:

```
How many elements are inserted:5
Enter 5 elements:
4
6
2
9
1
Sorted elements are:
1
2
4
6
9
```

Aim:c)write a C program for insertion sort in ascending order.

Program:

```
#include<stdio.h>int
main(){
/* Here i & j for loop counters, temp for swapping,
 *   count for total number of elements, number[] to
 *   store the input numbers in array. You can increase
 *   or decrease the size of number array as per requirement
*/
int i, j, count, temp, number[25];
printf("How many numbers u are going to enter?: ");scanf("%d",&count);
printf("Enter %d elements: ", count);
// This loop would store the input numbers in arrayfor(i=0;i<count;i++)
scanf("%d",&number[i]);
// Implementation of insertion sort algorithmfor(i=1;i<count;i++){
temp=number[i];j=i-
1;
while((temp<number[j])&&(j>=0)){
number[j+1]=number[j];
j=j-1;
}
number[j+1]=temp;
}
printf("Order of Sorted elements: ");for(i=0;i<count;i++)printf(" %d",number[i]);
return 0;
}
```

Output:

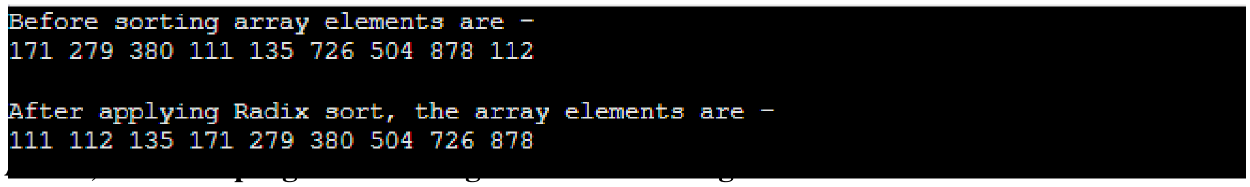
```
How many elements are entered:4Enter 4
elements:
22
11
33
44
Sorted elements are:
11
22
33
44
```

Exercise-3(sorting-2)
Aim:a) write a C program for radix sort in ascending order
Program:

```
#include<stdio.h>int
getMax(int arr[], int n) {
int mx = arr[0];
int i;
 for (i = 1; i < n; i++)if (arr[i] > mx)
       mx = arr[i];
       return mx;
         }

         void countSort(int arr[], int n, int exp) {
         int output[n]; // output array
         int i, count[10] = { 0 };

         // Store count of occurrences in count[]
         for (i = 0; i < n; i++)
         count[(arr[i] / exp) % 10]++;

         for (i = 1; i < 10; i++)
         count[i] += count[i - 1];

         // Build the output array
         for (i = n - 1; i >= 0; i--) {
         output[count[(arr[i] / exp) % 10] - 1] = arr[i];
         count[(arr[i] / exp) % 10]--;
         }

         for (i = 0; i < n; i++)
         arr[i] = output[i];
         }

         // The main function to that sorts arr[] of size                    n using
Radix Sort
 void radixsort(int arr[], int n) {
         int     m = getMax(arr, n);

         int     exp;
         for     (exp = 1; m / exp > 0; exp *= 10)
```

```
        countSort(arr, n, exp);
        }

        void print(int arr[], int n) {
        int i;
for (i = 0; i < n; i++)
 printf("%d ", arr[i]);
 }
 int main() {
 int arr[] = { 170, 45, 75, 90, 802, 24, 2, 66 };
        int n = sizeof(arr) / sizeof(arr[0]);
        radixsort(arr, n);
        print(arr, n);
        return 0;
}
```

Output:



```
Before sorting array elements are -
171 279 380 111 135 726 504 878 112

After applying Radix sort, the array elements are -
111 112 135 171 279 380 504 726 878
```

Program:

```
/* C program for Merge Sort */
#include <stdio.h>
#include <stdlib.h>

// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int arr[], int l, int m, int r)
{
        int i, j, k;
        int n1 = m - l + 1;
        int n2 = r - m;

        /* create temp arrays */
        int L[n1], R[n2];

        /* Copy data to temp arrays L[] and R[] */for
        (i = 0; i < n1; i++)
                L[i] = arr[l + i];
        for (j = 0; j < n2; j++)
                R[j] = arr[m + 1 + j];
```

```
/* Merge the temp arrays back into arr[l..r]*/i =
0; // Initial index of first subarray
j = 0; // Initial index of second subarray k
= l; // Initial index of merged subarray
while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
                arr[k] = L[i];
                i++;
        }
        else {
                arr[k] = R[j];
                j++;
        }
        k++;
}
```

```
        /* Copy the remaining elements of L[], if there
        are any */
        while (i < n1) {
                arr[k] = L[i];
                i++;
                k++;
        }

        /* Copy the remaining elements of R[], if there
        are any */
        while (j < n2) {
                arr[k] = R[j];
                j++;
                k++;
        }
}

/* l is for left index and r is right index of the
sub-array of arr to be sorted */
void mergeSort(int arr[], int l, int r)
{
        if (l < r) {
                // Same as (l+r)/2, but avoids overflow for
                // large l and h
                int m = l + (r - l) / 2;

                // Sort first and second halves
                mergeSort(arr, l, m);
                mergeSort(arr, m + 1, r);

                merge(arr, l, m, r);


        }
}

/* UTILITY FUNCTIONS */
/* Function to print an array */
void printArray(int A[], int size)
{
        int i;
        for (i = 0; i < size; i++)
                printf("%d ", A[i]);
        printf("\n");
}

/* Driver code */
```

```
int main()
{
	int arr[] = { 12, 11, 13, 5, 6, 7 };
	int arr_size = sizeof(arr) / sizeof(arr[0]);

	printf("Given array is \n");
	printArray(arr, arr_size);

	mergeSort(arr, 0, arr_size - 1);

	printf("\nSorted array is \n");
	printArray(arr, arr_size);
	return 0;
}
```

Output:

```
Given array is12

11 13 5 6 7

Sorted array is5 6 7

11 12 13
```

Exercise-4(singly linked list)

Aim:a)write a C program to create a singly linked list. b)write a C program to insert a node in singly linked list.

c)write a C program to delete a node in singly linked list.Program:

```
#include<stdio.h>
#include<stdlib.h>struct node
{
        int data;
        struct node *next;
};
struct node *head;
void beginsert ();
void lastinsert (); void
randominsert(); void
begin_delete(); void
last_delete(); void
random_delete();void
display();
void search();void
main ()
{
        int choice =0;
        while(choice != 9)
        {
                printf("\n\n*********Main            Menu*********\n"); printf("\nChoose
                one option from the following list ...\n");

printf("\n===============================================\n");
                printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at
any random location\n4.Delete from Beginning\n 5.Delete from last\n6.Delete
node after specified location\n7.Search for anelement\n8.Show\n9.Exit\n");
            printf("\nEnter your choice?\n"); scanf("\n%d",&choice);switch(choice)
            {
                  case 1:
                  beginsert();
                  break;
                  case 2:
                  lastinsert();
                  break;
                  case 3:
                  randominsert();
                  break;
                  case 4:
                  begin_delete();
                  break;
                  case 5:
                  last_delete();
                  break;
                  case 6:
                  random_delete();
                  break;
                  case 7:
                  search();
                  break; case
                  8: display();
                  break; case
                  9:
                  exit(0);
                  break;
```

```
                    default:
                    printf("Please enter valid choice..");
                }
        }
}
void beginsert()
{
        struct node *ptr;int
        item;
        ptr = (struct node *) malloc(sizeof(struct node *));if(ptr == NULL)
        {
                printf("\nOVERFLOW");
        }
        else
        {
                printf("\nEnter              value\n");
                scanf("%d",&item);
                ptr->data = item; ptr-
                >next = head; head =
                ptr;
                printf("\nNode inserted");
        }
```

```
}
void lastinsert()
{
        struct node *ptr,*temp;

        int item;
        ptr = (struct node*)malloc(sizeof(struct node));if(ptr == NULL)
        {
                printf("\nOVERFLOW");
        }
        else
        {
                printf("\nEnter value?\n");
                scanf("%d",&item);
                ptr->data = item;
                if(head == NULL)
                {
                        ptr -> next = NULL;
                        head = ptr;
                        printf("\nNode inserted");
                }
                else
                {
                        temp = head;
                        while (temp -> next != NULL)
                        {
```

```
                        temp = temp -> next;
                    }
                    temp->next = ptr;ptr-
                    >next = NULL;
                    printf("\nNode inserted");

                }

        }
}
void randominsert()
{
        int i,loc,item;
        struct node *ptr, *temp;
        ptr = (struct node *) malloc (sizeof(struct node));if(ptr == NULL)
        {
```

```
");
```

```
}
else
{
```

```
printf("\nOVERFLOW");
printf("\nEnter element value");scanf("%d",&item);

ptr->data = item;

printf("\nEnter the location after which you want to insert


scanf("\n%d",&loc);temp=head;

for(i=0;i<loc;i++)

{


                    temp = temp->next;

                    if(temp == NULL)

                    {

                          printf("\ncan't  insert\n");return;

                    }


               }

               ptr ->next = temp ->next; temp -

               >next = ptr; printf("\nNode

               inserted");

          }

     }

     void begin_delete()

     {

          struct node *ptr;

          if(head == NULL)

          {


                         }
          }
                    }
          else  printf("\nList is empty\n");
          {
                ad = ptr->next;
                free(ptr);
ptr = head;
                printf("\nNode deleted from the begining ...\n");
h

e

  void last_delete()
  {
        struct node *ptr,*ptr1;
        if(head == NULL)
        {
                printf("\nlist is empty");
        }
        else if(head -> next == NULL)
        {
                head = NULL;
                free(head);
                printf("\nOnly node of the list deleted ...\n");
        }

        else
        {
                ptr = head;
                while(ptr->next != NULL)
                {
                        ptr1 = ptr;
                        ptr = ptr ->next;
                }
                ptr1->next = NULL;
                free(ptr);
                printf("\nDeleted Node from the last ...\n");
        }
}
void random_delete()
{
        struct node *ptr,*ptr1;int
        loc,i;
        printf("\n Enter the location of the node after which you want toperform deletion \n");
        scanf("%d",&loc);
        ptr=head;
        for(i=0;i<loc;i++)
        {
            ptr1 = ptr;
            ptr = ptr->next;

            if(ptr == NULL)
            {
                printf("\nCan't delete");return;
            }
        }
        ptr1 ->next = ptr ->next;
        free(ptr);
        printf("\nDeleted node %d ",loc+1);
    }
    void search()
    {
        struct node *ptr; int
        item,i=0,flag;ptr   =
        head; if(ptr == NULL)
        {
            printf("\nEmpty List\n");
        }
        else
        {
            printf("\nEnter item which you want to search?\n");scanf("%d",&item);
            while (ptr!=NULL)
            {
                if(ptr->data == item)
```

```
                    {
                            printf("item found at location %d ",i+1);flag=0;

                    }
                    else
                    {       flag=1;

                    }
                    i++;
                    ptr = ptr -> next;
            }
            if(flag==1)
            {
                    printf("Item not found\n");
            }
    }

}

void display()
{
        struct node *ptr;ptr
        = head; if(ptr ==
        NULL)
        {
                printf("Nothing to print");
        }
        else
        {
```

```
        printf("\nprinting values.................................. \n");
        while (ptr!=NULL)
        {
            printf("\n%d",ptr->data);ptr = ptr
            -> next;
        }
    }
```

Output:

```
Enter
1. Insert Front
2. Delete Front
3. Display the list
4. Exit
2
Cannot delete. Empty ListEnter
1. Insert Front
2. Delete Front
3. Display the list
4. Exit
3
Contents of linked list is:Cannot
print. Empty list Enter
1. Insert Front
2. Delete Front
3. Display the list
4. Exit
1
Enter item to be inserted10
Enter
1. Insert Front
2. Delete Front
3. Display the list
4. Exit
1
Enter item to be inserted20
Enter
1. Insert Front
```

```
2. Delete Front
3. Display the list
4. Exit
1
Enter item to be inserted30
Enter
1. Insert Front
2. Delete Front
3. Display the list
4. Exit
1
Enter item to be inserted40
Enter
1. Insert Front
2. Delete Front
3. Display the list
4. Exit
1
Enter item to be inserted50
Enter
1. Insert Front
2. Delete Front
3. Display the list
4. Exit
3
Contents of linked list is: 50        40
            30        20        10
Enter
1. Insert Front
2. Delete Front
3. Display the list
4. Exit
2
Deleted node is 50
Enter
1. Insert Front
2. Delete Front
3. Display the list
4. Exit
2
Deleted node is 40
Enter
1. Insert Front
```

```
2. Delete Front
3. Display the list
4. Exit
2
Deleted node is 30
Enter
1. Insert Front
2. Delete Front
3. Display the list
4. Exit
3
Contents of linked list is:
20          10
Enter
1. Insert Front
2. Delete Front
3. Display the list
4. Exit
```

Aim:d) write a C program to reverse the elements of singly linked listProgram:

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

/* Link list node */
struct Node {
        int data;
        struct Node* next;
};

/* Function to reverse the linked list */
static void reverse(struct Node** head_ref)
{
        struct Node* prev = NULL;
        struct Node* current = *head_ref;
        struct Node* next = NULL;
        while (current != NULL) {
                // Store next
                next = current->next;

                // Reverse current node's pointer
                current->next = prev;

                // Move pointers one position ahead.
                prev = current;
                current = next;
        }
        *head_ref = prev;
}

/* Function to push a node */
void push(struct Node** head_ref, int new_data)
{
        struct Node* new_node
                = (struct Node*)malloc(sizeof(struct Node));
        new_node->data = new_data;
        new_node->next = (*head_ref);
        (*head_ref) = new_node;
}

/* Function to print linked list */
void printList(struct Node* head)
{
        struct Node* temp = head;
        while (temp != NULL) {
                printf("%d ", temp->data);
                temp = temp->next;
        }
}

/* Driver code*/
int main()
{
        /* Start with the empty list */
        struct Node* head = NULL;

        push(&head, 20);
        push(&head, 4);
        push(&head, 15);
        push(&head, 85);

        printf("Given linked list\n");
        printList(head);
        reverse(&head);
        printf("\nReversed Linked list \n");
        printList(head);
        getchar();
}
```

Output:

```
Given linked list
85 15 4 20
Reversed Linked list

20 4 15 85
```

Exercise-5(QUEUE)

Aim:write a C program to implement queue using arrays.Program:

```
#include <stdio.h>

#define MAX 5

void Enqueue();

void Dequeue();

void display();

int queue_array[MAX];

int rear = - 1;

int front = - 1;

int main()

{

   int choice;

   while(1)

   {

      printf("1.ENQUEUE \n");

      printf("2.DEQUEUE \n");

      printf("3.DISPLAY \n");

      printf("4.QUIT \n");

      printf("Enter your choice : ");
```

```
        scanf("%d", &choice);
        switch(choice)
        {
          case 1:
          insert();
          break;
          case 2:
          delete();
          break;
          case 3:
          display();
          break;
          case 4:
          printf("Exit Point");
          break;
          default:
          printf("Wrong choice \n");
        } /* End of switch */
      } /* End of while */
    } /* End of main() */
     void Enqueue(); ()
    {
      int add_item;
      if (rear == MAX - 1)
      printf("Queue Overflow \n");
    else
    {
       if (front == - 1)
       /*If queue is initially empty */
      {
          front = 0;
                     }
       printf("Inset the element in queue : ");
       scanf("%d", &add_item);
       rear = rear + 1;
       queue_array[rear] = add_item;
    }
} /* End of insert() */
 void Dequeue()
{
    if (front == - 1 || front > rear)
    {
       printf("Queue Underflow \n");
       return ;
    }
    else
    {
       printf("Element deleted from queue is : %d\n", queue_array[front]);
       front = front + 1;
    }
```

```
} /* End of delete() */

 void display()

{

    int i;

    if (front == - 1)

        printf("Queue is empty \n");

    else

    {

        printf("Queue is : \n");

        for (i = front; i <= rear; i++)

            printf("%d ", queue_array[i]);

        printf("\n");

    }

}
```

Output:

```
1.Insert element to queue 2.Delete
element from queue 3.Display all
elements of queue4.Quit
Enter your choice : 1
Inset the element in queue : 10
1.Insert element to queue 2.Delete
element from queue 3.Display all
elements of queue4.Quit
Enter your choice : 1
Inset the element in queue : 15
1.Insert element to queue 2.Delete
element from queue 3.Display all
elements of queue4.Quit
Enter your choice : 1
Inset the element in queue : 20
1.Insert element to queue 2.Delete
element from queue 3.Display all
elements of queue4.Quit
```

```
Enter your choice : 1
Inset the element in queue : 30
1.Insert element to queue 2.Delete
element from queue 3.Display all
elements of queue4.Quit
Enter your choice : 2
Element deleted from queue is : 10
1.Insert element to  queue 2.Delete
element from queue 3.Display all
elements of queue 4.Quit
Enter your choice : 3Queue
is :
15 20 30
1.Insert element to queue 2.Delete
element from queue 3.Display all
elements of queue4.Quit
Enter your choice : 4
```

Aim:b)Implementation of Queue using Linked List

Program:

```
#include<stdio.h>

struct Node

{

  int data;

  struct Node *next;

}*front = NULL,*rear = NULL;

void enqueue(int);

void dequeue();

void display();

int main()
```

```
{
  int choice, value;
  printf("\n:: Queue Implementation using Linked List ::\n");
  while(1){
    printf("\n****** MENU ******\n");
    printf("1. ENQUEUE\n2. DEQUEUE\n3. Display\n4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d",&choice);
    switch(choice){
         case 1: printf("Enter the value to be insert: ");
                 scanf("%d", &value);
                 enqueue(value);
                 break;
         case 2: dequeue(); break;
         case 3: display(); break;
         case 4: printf("Exit point"); break;
         default: printf("\nWrong selection!!! Please try again!!!\n");
    }
  }
}
void enqueue(int value)
{
  struct Node *newNode;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = value;
  newNode -> next = NULL;
  if(front == NULL)
    front = rear = newNode;
  else{
    rear -> next = newNode;
    rear = newNode;
  }
  printf("\nInsertion is Success!!!\n");
}
void dequeue()
{
  if(front == NULL)
    printf("\nQueue is Empty!!!\n");
  else{
    struct Node *temp = front;
    front = front -> next;
    printf("\nDeleted element: %d\n", temp->data);
    free(temp);
  }
}
void display()
{
  if(front == NULL)
    printf("\nQueue is Empty!!!\n");
  else{
    struct Node *temp = front;

    while(temp->next != NULL){

          printf("%d--->",temp->data);

          temp = temp -> next;

    }

    printf("%d--->NULL\n",temp->data);

  }

}
```

Output:

```
1.Insert an element
2.Delete an element
3.Display the element
4.Exit
Enter your choice:1
Enter value:10

1.Insert an element
2.Delete an element
3.Display the element
4.Exit
Enter your choice:1
Enter value:20

1.Insert an element
2.Delete an element
3.Display the element
4.Exit
Enter your choice:1
Enter value:30

1.Insert an element
2.Delete an element
3.Display the element
4.Exit
Enter your choice:2
```

```
1.Insert an element
2.Delete an element
3.Display the element
4.Exit
Enter your choice:3
Elements of Queue: 20 30

1.Insert an element
2.Delete an element
3.Display the element
4.Exit
Enter your choice:4
```

Exercise-6 (stack)
Aim: a) write a C program to implement stack using arrays.

Program:

```
#include<stdio.h>

int stack[100],choice,n,top,x,i;

void push(void);

void pop(void);

void display(void);

int main()

{

    top=-1;

    printf("\n Enter the size of STACK[MAX=100]:");

    scanf("%d",&n);

    printf("\n\t STACK OPERATIONS USING ARRAY");

    printf("\n\t --------------------------------");

    printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");do

    {

        printf("\n Enter the Choice:");

        scanf("%d",&choice);

        switch(choice)

        {

            case 1:

            {
```

```
            push();
            break;
        }
        case 2:
        {
            pop();
            break;
        }
        case 3:
        {
            display();
            break;
        }
        case 4:
        {
            printf("\n\t EXIT POINT ");
            break;
        }
        default:
        {
            printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");
        }
       }
   }
   while(choice!=4);
```

```
    return 0;
}
void push()
{
    if(top==n-1)
    {
        printf("\n\tSTACK is over flow");


    }
    else
    {
        printf(" Enter a value to be pushed:");
        scanf("%d",&x);
        top++;
        stack[top]=x;
    }
}
void pop()
{
    if(top==-1)
    {
        printf("\n\t Stack is under flow");
    }
    else
    {
            printf("\n\t The popped elements is %d",stack[top]);top-
            -;
        }
    }
    void display()
    {
        if(top>=0)
        {
            printf("\n The elements in STACK \n");
            for(i=top; i>=0; i--)
                printf("\n%d",stack[i]);
            printf("\n Press Next Choice");
        }
        else
        {
            printf("\n The STACK is empty");
        }
      }
```

Output:

```
Enter the size of STACK[MAX=100]:10

STACK OPERATIONS USING ARRAY

         1.PUSH
         2.POP
         3.DISPLAY
         4.EXIT
 Enter the Choice:1
 Enter a value to be pushed:12
Enter the Choice:1
Enter a value to be pushed:24

Enter the Choice:1
Enter a value to be pushed:98Enter the

Choice:3

The elements in STACK

98
24
12
Press Next ChoiceEnter
the Choice:2

        The popped elements is 98
        Enter the Choice:3

The elements in STACK

24
12

Press Next ChoiceEnter
the Choice:4
```

Aim:b)write a C program to implement stack using linked list

Program:

```
#include<stdio.h>
#include<conio.h>
struct Node
{
  int data;
  struct Node *next;
}*top = NULL;


void push(int);
void pop();
void display();
void main()
{
  int choice, value;
  clrscr();
  printf("\n:: Stack using Linked List ::\n");
  while(1){
    printf("\n****** MENU ******\n");
    printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
    printf("Enter your choice: "); scanf("%d",&choice);
    switch(choice){
           case 1: printf("Enter the value to be insert: ");
                      scanf("%d", &value);
                      push(value);
                      break;
           case 2: pop(); break;
           case 3: display(); break;
           case 4: exit(0);
           default: printf("\nWrong selection!!! Please try again!!!\n");
    }
  }
}
void push(int value)
{
  struct Node *newNode;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = value;
  if(top == NULL)
    newNode->next = NULL;
```

```
   else
      newNode->next = top;
   top = newNode;
   printf("\nInsertion is Success!!!\n");
}
void pop()
{
   if(top == NULL)
      printf("\nStack is Empty!!!\n");
   else{
      struct Node *temp = top;
      printf("\nDeleted element: %d", temp->data);top
      = temp->next;
      free(temp);
   }
}
```

Output:

```
1 - Push
2 - Pop
3 - Top
4 - Empty
5 - Exit
6 - Dipslay
7 - Stack Count
8 - Destroy stackEnter
choice : 1 Enter data :
56

Enter choice : 1Enter
data : 80

Enter choice : 2

Popped value : 80Enter
choice : 3

Top element : 56Enter
```

```
choice : 1
Enter data : 78

Enter choice : 1Enter
data : 90

Enter choice : 690 78
56
Enter choice : 7

No. of elements in stack : 3Enter
choice : 8

All stack elements destroyedEnter
choice : 4

Stack is empty Enter
choice : 5
```

Aim:c)write a C program that uses stack operations to evaluate postfix expression

Program:

```
#include<stdio.h>

int stack[20]; int top

= -1;


void push(int x)

{

    stack[++top] = x;

}


int pop()

{

    return stack[top--];
```

```
}
int main()
{
        char exp[20];char
        *e;
        int n1,n2,n3,num;
        printf("Enter the expression :: ");
        scanf("%s",exp);
        e =   exp; while(*e !=
        '\0')
        {
                if(isdigit(*e))
                {
                        num = *e - 48;
                        push(num);
                }
                else
                {
                        n1 = pop();
                        n2 = pop();
                        switch(*e)
                        {
                        case '+':
                        {
```

```
                n3 = n1 + n2;
                break;
```

```
            }
            case '-':
            {
                n3 = n2 - n1;
                break;
            }
            case '*':
            {
                n3 = n1 * n2;
                break;
            }
            case '/':
            {
                n3 = n2 / n1;
                break;
            }
            }
            push(n3);
        }
        e++;
    }
    printf("\nThe result of expression %s  =  %d\n\n",exp,pop());return 0;
}
```

OUTPUT:

```
Enter the expression :: 245+*

The result of expression 245+*  =  18
```

Exercise-7(binary tree)

Aim:d)write a c program for traversing binary tree in preorder,inorder and postorder.

Program:

```
 #include <stdio.h>

#include <stdlib.h>


/* A binary tree node has data, pointer to left childand a

pointer to right child */

struct node {

        int data;

        struct node* left;

        struct node* right;

};


/* Helper function that allocates a new node with thegiven

data and NULL left and right pointers. */

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);
```

```
}

/* Given a binary tree, print its nodes according to the
"bottom-up" postorder traversal. */
void printPostorder(struct node* node)
{
        if (node == NULL)
                return;

        // first recur on left subtree
        printPostorder(node->left);

        // then recur on right subtree
        printPostorder(node->right);

        // now deal with the node
        printf("%d ", node->data);
}

/* Given a binary tree, print its nodes in inorder*/
void printInorder(struct node* node)
{
        if (node == NULL)
                return;

        /* first recur on left child */
        printInorder(node->left);

        /* then print the data of node */
        printf("%d ", node->data);

        /* now recur on right child */
        printInorder(node->right);
}

/* Given a binary tree, print its nodes in preorder*/void
printPreorder(struct node* node)
{
        if (node == NULL)
                return;

        /* first print data of node */

        printf("%d ", node->data);

        /* then recur on left subtree */
        printPreorder(node->left);

        /* now recur on right subtree */
        printPreorder(node->right);
}
```

```
/* Driver program to test above functions*/
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("\nPreorder traversal of binary tree is \n");
        printPreorder(root);

        printf("\nInorder traversal of binary tree is \n");
        printInorder(root);
```

```
    printf("\nPostorder traversal of binary tree is \n");

    printPostorder(root);


    getchar();

    return 0;

}
```

Output:

```
Preorder traversal of binary tree is1 2 4 5 3
Inorder traversal of binary tree is4 2 5 1 3
Postorder traversal of binary tree is4 5 2 3 1
```

Exercise-8(binary search tree)

Aim:a)write a C program to create a BST b)write a C program to insert a node into a BSTprogram:

```
#include<stdio.h>

#include<stdlib.h>


struct node {

    int key;

    struct node *left, *right;

};

// A utility function to create a new BST node

struct node*create newNode(int item)

{

    struct node* temp
```

```
                = (struct node*)malloc(sizeof(struct node));

        temp->key = item;

        temp->left = temp->right =

        NULL;return temp;

}


// A utility function to do inorder traversal of BST


/* A utility function to inserta

new node with given key in

* BST */

struct node* insert(struct node* node, int key)

{

        /* If the tree is empty, return a new node */if

        (node == NULL)

                return newNode(key);


        /* Otherwise, recur down the tree */if

        (key < node->key)

                node->left = insert(node->left, key);

        else if (key > node->key)

                node->right = insert(node->right, key);


        /* return the (unchanged) node pointer */

        return node;
```

```
}
int main()
{
        struct node* root =
        NULL;root = insert(root,
        50); insert(root, 30);
        insert(root, 20);
        insert(root, 40);
        insert(root, 70);
        insert(root, 60);
        insert(root, 80);

        return 0;
}
```

Output:

```
20
30
40
50
60
70
80
```

Aim:c)write a C program to delete a node from a BSTProgram:

```
#include  <stdio.h>

#include<stdlib.h>

struct node

{

        int key;

        struct node *left, *right;

};


// A utility function to create a new BST node

struct node* newNode(int item)

{

        struct node* temp

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

        temp->key = item;

        temp->left = temp->right =

        NULL;return temp;

}
// A utility function to do inorder traversal of BST

void inorder(struct node* root)

{

        if (root != NULL) {

                inorder(root->left);

                printf("%d ", root-

                >key);inorder(root-

                >right);
        }
}

/* A utility function to
insert a new node with given key in
* BST */
struct node* insert(struct node* node, int key)
{
        /* If the tree is empty, return a new node */if
        (node == NULL)
                return newNode(key);

        /* Otherwise, recur down the tree */if
        (key < node->key)
                node->left = insert(node->left, key);
        else
                node->right = insert(node->right, key);

        /* return the (unchanged) node pointer */
        return node;
}

/* Given a non-empty binary search
tree, return the node
with minimum key value found in
that tree. Note that the
```

```
entire tree does not need to be searched. */ struct
node* minValueNode(struct node* node)
{
        struct node* current = node;

        /* loop down to find the leftmost leaf */
        while (current && current->left !=
        NULL)
                current = current->left;

        return current;
}

/* Given a binary search tree
and a key, this function
deletes the key and
returns the new root */
struct node* deleteNode(struct node* root, int key)

{
        // base case
        if (root == NULL)
                return root;

         // If the key to be deleted
        // is smaller than the root's
        // key, then it lies in left subtree
        if (key < root->key)
                root->left = deleteNode(root->left, key);

         // If the key to be deleted
        // is greater than the root's
        // key, then it lies in right subtree
        else if (key > root->key)
                root->right = deleteNode(root->right, key);

        // if key is same as root's key,
        // then This is the node
        // to be deleted
        else {
                // node with only one child or no child
                if (root->left == NULL) {
                        struct node* temp = root->right;

                        free(root);
                        return temp;
                }
                else if (root->right == NULL) {
                        struct node* temp = root->left;
                        free(root);
                        return temp;
                }
```

```
                // node with two children:
                // Get the inorder successor
                // (smallest in the right subtree)
                struct node* temp = minValueNode(root->right);

                // Copy the inorder
                // successor's content to this node
                root->key = temp->key;

                // Delete the inorder successor
                root->right = deleteNode(root->right, temp->key);
        }
        return root;
}
// Driver Code
int main()
{
        /* Let us create following BST
                        50
                /        \
                30       70
                / \ / \
        20 40 60 80 */
        struct node* root =
        NULL;root = insert(root,
    50); root = insert(root,
    30); root = insert(root,
    20); root = insert(root,
    40); root = insert(root,
    70); root = insert(root,
    60); root = insert(root,
    80);

    printf("Inorder traversal of the given tree \n");
    inorder(root);

    printf("\nDelete  20\n"); root
    = deleteNode(root, 20);
    printf("Inorder traversal of the modified tree \n");
    inorder(root);
   printf("\nDelete  30\n"); root
  = deleteNode(root, 30);
    printf("Inorder traversal of the modified tree \n");
    inorder(root);

    printf("\nDelete  50\n"); root
    = deleteNode(root, 50);
    printf("Inorder traversal of the modified tree \n");
    inorder(root);
   return 0;
}
```

Ouput:

```
Inorder traversal of the given tree
20304050607080
Delete 20
Inorder traversal of the modified tree304050607080
Delete 30
Inorder traversal of the modified tree4050607080
Delete 50
Inorder traversal of the modified tree40607080
```