EXERCISE – 1

1a) Write C program that use both recursive and non recursive functions to perform Linear search for a Key value in a given list.

AIM: To show a C program that use both recursive and non recursive functions to perform Linear search for a Key value in a given list.

Program:

```
#include<stdio.h>
void linear_search(int val,int n,int arr[*]);     //non-recursive function
int linear_search_re(int val,int n,int arr[*]);    //recursive function
void main()
{
 int n,i,val,pos,option;
 printf("enter the size of array: ");
 scanf("%d",&n);
 int arr[n];
         printf("enter the elements: ");
        for(i=0;i<n;i++)
                scanf("%d",&arr[i]);
        printf("enter the value to search: ");
        scanf("%d",&val);
        printf("enter 1 for Non-Recursive Function \n");
        printf("enter 2 for Recursive Function\n ");
        scanf("%d",&option);
        switch(option)
        {
        case 1:
          {
                linear_search(val,n,arr);
                break;
          }
        case 2:
          {
                pos=linear_search_re(val,n,arr);
                if(pos!=0)
                      printf("\nelement %d is found at position %d using Recursion",val, pos);
                else
                       printf("\nelement %d does not found using Recursion",val);
                break;
          }
        default:
                printf("enter correct option! ");
        }

}

void linear_search(int val,int n,int arr[n])     //Non-recursive function
{
        int i,found=0,pos=-1;
        for(i=0;i<n;i++)
        {
             if(arr[i]==val)
            {
                  found=1;
                  pos=i;
                  printf("\nelement %d is found at position %d using NonRecursion",val,i+1);
            break;
            }
        }
        if(found==0)
                printf("\nelement %d does not found using Non-Recursion",val
}

int linear_search_re(int val,int n,int arr[n]) //Recursive function
{
        if(n>0)
        {
                if(arr[n-1]==val)
                        return n;
                else
                        return linear_search_re(val,n-1,arr);
        }
        return 0;
}
```

OUTPUT:

Output 1:

```
enter the size of array: 5
enter the elements: 2 5 7 9 3
enter the value to search: 2
enter 1 for Non-Recursive Function
enter 2 for Recursive Function
 1
element 2 is found at position 1 using NonRecursion
```

Output 2:

```
enter the size of array: 5
enter the elements: 2 5 7 9 3
enter the value to search: 4
enter 1 for Non-Recursive Function
enter 2 for Recursive Function
 1
element 4 does not found using Non-Recursion
```

Output 3:

```
enter the size of array: 5
enter the elements: 5 4 7 1 6
enter the value to search: 7
enter 1 for Non-Recursive Function
enter 2 for Recursive Function
 2

element 7 is found at position 3 using Recursion
```

Output 4:

```
enter the size of array: 5
enter the elements: 5 4 7 1 6
enter the value to search: 8
enter 1 for Non-Recursive Function
enter 2 for Recursive Function
 2

element 8 does not found using Recursion
```

1b) Write C program that use both recursive and non recursive functions to perform Binary search for a Key value in a given list.

Aim: To show a C program that use both recursive and non recursive functions to perform Binary search for a Key value in a given list.

Program:

```
#include<stdio.h>
int binaryrecur(int n,int arr[*],int beg, int end, int val);        //Recursive function
void binarynonrecur(int n, int arr[*],int beg, int end, int val);          //Non-Recursive
void main()
{
 int n,i,val,pos,option;
 printf("enter the size of array: ");
 scanf("%d",&n);
 int arr[n];
        printf("enter the elements in ascending order: ");
        for(i=0;i<n;i++)
                scanf("%d",&arr[i]);
        printf("enter the value to search: ");
        scanf("%d",&val);
        printf("enter 1 for Non-Recursive Function \n");
        printf("enter 2 for Recursive Function\n");
        scanf("%d",&option);
        switch(option)
        {
        case 1:
          {
                binarynonrecur(n,arr,0,n-1,val);
                break;
          }
        case 2:
          {
              pos=binaryrecur(n,arr,0,n-1,val);
              if(pos!=-1)
                     printf("\nelement %d found at %d position using Recursion",val,pos+1);
              else
                     printf("\nelement %d not found using Recursion",val);
              break;
          }
    default:
                printf("enter correct option! ");
        }
}

int binaryrecur(int n,int arr[n],int beg,int end,int val)          //Recursion Function
{
        if(beg<=end)
        {
                int mid=(beg+end)/2;
                if(arr[mid]==val)
```

```
                    return mid;
            if(arr[mid]>val)
                    return binaryrecur(n,arr,beg,mid-1,val);
            else
                    return binaryrecur(n,arr,mid+1,end,val);
        }
        return -1;
}

void binarynonrecur(int n,int arr[n],int beg,int end,int val)  //Non-Recursion
{
        int mid,pos=-1;
        while(beg<=end)
        {
          mid=(beg+end)/2;
          if(arr[mid]==val)
           {
             pos=mid+1;
           printf("\nelement %d is found at position %d using Non-Recursion",val,pos);
          break;
           }
         else if(arr[mid]>val)
                end=mid-1;
         else
                beg=mid+1;
        }
        if(pos==-1)
          printf("\nelement %d does not found in the array using Non-Recursion",val);
}
```

OUTPUT:

Output 1:

```
enter the size of array: 5
enter the elements in ascending order: 2 4 6 8 10
enter the value to search: 4
enter 1 for Non-Recursive Function
enter 2 for Recursive Function
1
element 4 is found at position 2 using Non-Recursion
```

Output 2:

```
enter the size of array: 5
enter the elements in ascending order: 2 4 6 8 10
enter the value to search: 5
enter 1 for Non-Recursive Function
enter 2 for Recursive Function
1
element 5 does not found in the array using Non-Recursion
```

Output 3:

```
enter the size of array: 5
enter the elements in ascending order: 1 3 5 7 9
enter the value to search: 3
enter 1 for Non-Recursive Function
enter 2 for Recursive Function
2
element 3 found at 2 position using Recursion
```

Output 4:

```
enter the size of array: 5
enter the elements in ascending order: 1 3 5 7 9
enter the value to search: 4
enter 1 for Non-Recursive Function
enter 2 for Recursive Function
2
element 4 not found using Recursion
```

EXERCISE - 2

2 a) Write C program that implement Bubble sort, to sort a given list of integers in ascending order

Aim: To show a C program that implement Bubble sort, to sort a given list of integers in ascending order

Program:

```
#include<stdio.h>
int main()
{
 int i,n,temp,j;
 printf("enter the number of values: ");
 scanf("%d",&n);
 int arr[n];
 printf("enter the array elements: ");
 for(i=0;i<n;i++)
                scanf("%d",&arr[i]);
        for(i=0 ; i < n-1 ; i++)
        {
                for(j=0 ; j < n-1-i ; j++)
                {
                  if(arr[j] > arr[j+1])
                    {
                        temp=arr[j];
                        arr[j]=arr[j+1];
                        arr[j+1]=temp;
                    }
                }
        }
        printf("\nAfter sorting the elements of an array is : ");
        for(i=0; i<n ;i++)
                printf("%d ",arr[i]);
}
```

OUTPUT:

```
enter the number of values: 6
enter the array elements: 8 4 2 7 6 1
After sorting the elements of an array is : 1 2 4 6 7 8
```

2b) Write C program that implement Quick sort, to sort a given list of integers in ascending order

Aim: To show a C program that implement Quick sort, to sort a given list of integers in ascending order.

Program:

```
// c program to perform quick sort
#include <stdio.h>

void quicksort (int [], int, int);

void main()
{
 int arr[20];
 int n, i;

        printf("Enter the number of elements: ");
                scanf("%d", &n);
        printf("Enter the elements to be sorted: ");
        for (i = 0; i < n; i++)
                scanf("%d", &arr[i]);
        quicksort(arr, 0, n - 1);
        printf("After applying quick sort: ");
        for (i = 0; i < n; i++)
                printf("%d ", arr[i]);
}

void quicksort(int arr[], int low, int high)
{
        int pivot, i, j, temp;
        if (low < high)
        {
                pivot = low;
                i = low;
                j = high;
                while (i < j)
                {
                        while (arr[i] <= arr[pivot] && i <= high)
                        {
                                i++;
                        }
                        while (arr[j] > arr[pivot] && j >= low)
                        {
                                j--;
```

```
            }
            if (i < j)
            {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        temp = arr[j];
        arr[j] = arr[pivot];
        arr[pivot] = temp;
        quicksort(arr, low, j - 1);
        quicksort(arr, j + 1, high);
    }
}
```

OUTPUT:

```
Enter the number of elements: 6

Enter the elements to be sorted: 20 50 30 60 10 40

After applying quick sort: 10 20 30 40 50 60
```

2c) Write C program that implement Insertion sort, to sort a given list of integers in ascending order

Aim: To show a C program that implement Insertion sort, to sort a given list of integers in ascending order

Program:

```
#include<stdio.h>
void insertion_sort(int arr[*],int n);
void main()
{
 int i,n;
 printf("enter the size of an array: ");
 scanf("%d",&n);
 int arr[n];
 printf("enter the %d elements of an array: ",n);
        for(i=0 ; i<n ; i++)
                scanf("%d",&arr[i]);
        insertion_sort(arr,n);
}

void insertion_sort(int arr[], int n)
{
        int i,j,temp;
        for(i=1 ; i<n ; i++)
        {
                temp=arr[i];
                j=i-1;
                while((temp < arr[j]) && j>=0)
                {
                  arr[j+1]=arr[j];
                  j--;
                }
                arr[j+1]=temp;
        }
        printf("\nAfter sorting the elements of an array is : ");
        for(i=0 ; i<n ; i++)
                printf("%d ",arr[i]);
}
```

OUTPUT:

```
enter the size of an array: 6
enter the 6 elements of an array: 8 4 3 9 1 5
After sorting the elements of an array is : 1  3  4  5  8  9
```

EXERCISE – 3

3a) Write C program that implement radix sort, to sort a given list of integers in ascending order

Aim: To show C program that implement radix sort, to sort a given list of integers in ascending order

Program:

```
// c program to implement radix sort

#include <stdio.h>
#define size 10

int largest(int arr[], int n);
void radix_sort(int arr[], int n);

void main()
{
        int arr[size], i, n;
        printf("\n Enter the number of elements in the array: ");
                scanf("%d", &n);
        printf("\n Enter the elements of the array: ");
        for(i=0;i<n;i++)
                scanf("%d", &arr[i]);
        radix_sort(arr, n);
        printf("\n The sorted array is: ");
        for(i=0;i<n;i++)
                printf(" %d\t", arr[i]);
}

int largest(int arr[], int n)
{
        int large=arr[0], i;
        for(i=1;i<n;i++)
        {
                if(arr[i]>large)
                large = arr[i];
        }
        return large;
}

void radix_sort(int arr[], int n)
{
        int bucket[size][size], bucket_count[size];
        int i, j, k, remainder, NOP=0, divisor=1, large, pass;
        large = largest(arr, n);
```

```
    while(large>0)
    {
        NOP++;
        large/=size;
    }
    for(pass=0;pass<NOP;pass++) // Initialize the buckets
    {
        for(i=0;i<size;i++)
            bucket_count[i]=0;
        for(i=0;i<n;i++)
        {
            // sort the numbers according to the digit at passth place
            remainder = (arr[i]/divisor)%size;
            bucket[remainder][bucket_count[remainder]] = arr[i];
            bucket_count[remainder] += 1;
        }
        // collect the numbers after PASS pass
        i=0;
        for(k=0;k<size;k++)
        {
            for(j=0;j<bucket_count[k];j++)
            {
                arr[i] = bucket[k][j];
                i++;
            }
        }
        divisor *= size;
    }
}
```

OUTPUT:

```
Enter the number of elements in the array: 6

 Enter the elements of the array: 50 20 60 40 10 30

 The sorted array is:  10      20      30      40      50      60
```

3b) Write C program that implement merge sort, to sort a given list of integers in ascending order

Aim: To show C program that implement merge sort, to sort a given list of integers in ascending order

Program:

```
//c program to implement merge sort
#include <stdio.h>

void mergeSort(int [], int, int, int);
void partition(int [],int, int);

void main()
{
 int arr[20];
        int i, n;
        printf("Enter total number of elements:");
                scanf("%d", &n);
        printf("Enter the elements:");
        for(i = 0; i < n; i++)
                scanf("%d", &arr[i]);
        partition(arr, 0, n - 1);
        printf("After merge sort: ");
        for(i = 0;i < n; i++)
                printf("%d ",arr[i]);
}

void partition(int arr[],int low,int high)
{
        int mid;
        if(low < high)
        {
                mid = (low + high) / 2;
                partition(arr, low, mid);
                partition(arr, mid + 1, high);
                mergeSort(arr, low, mid, high);
        }
}

void mergeSort(int arr[],int low,int mid,int high)
{
        int i, mi, k, lo, temp[50];
        lo = low;
        i = low;
```

```
    mi = mid + 1;
    while ((lo <= mid) && (mi <= high))
    {
        if (arr[lo] <= arr[mi])
        {
            temp[i] = arr[lo];
            lo++;
        }
        else
        {
            temp[i] = arr[mi];
            mi++;
        }
        i++;
    }
    if (lo > mid)
        for (k = mi; k <= high; k++)
        {
            temp[i] = arr[k];
            i++;
        }
    else
        for (k = lo; k <= mid; k++)
        {
            temp[i] = arr[k];
            i++;
        }

    for (k = low; k <= high; k++)
        arr[k] = temp[k];
}
```

OUTPUT:

```
Enter total number of elements:6
Enter the elements:20 50 60 10 40 30
After merge sort: 10   20   30   40   50   60
```

EXERCISE – 4

4a) Write a C program that uses functions to create a singly linked list

Aim: To show a C program that uses functions to create a singly linked list

Program:

```
/* a) Write a C program that uses functions to create a singly linked list */
#include<stdio.h>
#include<stdlib.h>

struct node
{
 int data;
 struct node* next;
};

struct node* start = NULL;
struct node* createll(struct node *);
void display(struct node *);

void main()
{
        int option,val;
        do
        {
        printf("\n\n *****MAIN MENU *****");
        printf("\n 1: Create a list");
        printf("\n 2: Display the list");
        printf("\n 3: Exit");
        printf("\nenter your option: ");
        scanf("%d",&option);
        switch(option)
          {
        case 1:
                start = createll(start);
                printf("\n linked list is created");
                break;
        case 2:
                display (start);
                break;
          }
        }while(option != 3);
}

struct node* createll(struct node* start)
{
        struct node *newnode;
        struct node *ptr;
        int num;
        printf("\n enter the data or -1 to end: ");
        scanf("%d",&num);
```

```
    while(num!= -1)
    {
        newnode = (struct node *)malloc(sizeof(struct node));
        newnode->data=num;
        if(start == NULL)
        {
            newnode->next=NULL;
            start=newnode;
        }
        else
        {
            ptr = start;
            while(ptr->next != NULL)
            {
                ptr = ptr->next;
            }
            ptr->next = newnode;
            newnode->next=NULL;
        }
        printf("\n enter the data or -1 to end: ");
        scanf("%d",&num);
    }
    return start;
}

void display(struct node* start)
{
    struct node * ptr;
    ptr=start;
    while (ptr != NULL)
    {
        printf("%d\t ", ptr->data);
        ptr=ptr->next;
    }
}
```

OUTPUT:

```
*****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: Exit
enter your option: 1

enter the data or -1 to end: 10
 enter the data or -1 to end: 20
 enter the data or -1 to end: 30
 enter the data or -1 to end: 40
 enter the data or -1 to end: 50
 enter the data or -1 to end: -1
 linked list is created

 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: Exit
enter your option: 2
10      20      30      40      50
```

4b) Write a C program that uses functions to perform insertion operation on a singly linked list

Aim: To show a C program that uses functions to perform insertion operation on a singly linked list

Program:

```
/* b)Use functions to perform insertion operation in a singly linked list*/
#include<stdio.h>
#include<stdlib.h>

struct node
{
 int data;
 struct noden* next;
};

struct node* start = NULL;
struct node* createll(struct node *);
void display(struct node *);
struct node* insert_beg(struct node*);
struct node* insert_end(struct node*);
struct node* insert_after(struct node*);

void main()
{
	int option;
	do
	{
	printf("\n\n *****MAIN MENU *****");
	printf("\n 1: Create a list");
	printf("\n 2: Display the list");
	printf("\n 3: inserting a node at beginning of the list");
	printf("\n 4: insert a node at the end of the list");
	printf("\n 5: insert a node after a particular node in the list");
	printf("\n 6: Exit");
	printf("\nenter your option: ");
	scanf("%d",&option);
	switch(option)
	{
	case 1:
		start = createll(start);
		printf("\n linked list is created");
		break;
	case 2:
		display (start);
		break;
	case 3:
		start = insert_beg(start);
		break;
	case 4:
		start= insert_end(start);
		break;
	case 5:
		start = insert_after(start);
```

```
                break;
          }
        }while(option != 6);
}

struct node* createll(struct node* start)
{
        struct node *newnode;
        struct node *ptr;
        int num;
        printf("\n enter the data or -1 to end: ");
        scanf("%d",&num);
        while(num!= -1)
        {
                newnode = (struct node *)malloc(sizeof(struct node));
                newnode->data=num;
                if(start == NULL)
                {
                        newnode->next=NULL;
                        start=newnode;
                }
                else
                {
                        ptr = start;
                        while(ptr->next != NULL)
                        {
                                ptr = ptr->next;
                        }
                        ptr->next = newnode;
                        newnode->next=NULL;
                }
                printf("\n enter the data or -1 to end: ");
                scanf("%d",&num);
        }
        return start;
}

void display(struct node* start)
{
        struct node * ptr;
        ptr=start;
        while (ptr != NULL)
        {
                printf("%d\t ", ptr->data);
                ptr=ptr->next;
        }
}

struct node * insert_beg(struct node * start)
{
        struct node * newnode;
        int num;
        printf("\n enter the data: ");
        scanf("%d",&num);
        newnode=(struct node *)malloc(sizeof(struct node));
        newnode->data=num;
        newnode->next=start;
        start=newnode;
        return start;
}

struct node* insert_end(struct node * start)
{
        int num;
        printf("\n enter the data: ");
        scanf("%d",&num);
        struct node* new_node, *ptr;
        new_node= (struct node*)malloc(sizeof(struct node));
        new_node->data=num;
        new_node-> next= NULL;
        ptr=start;
        while(ptr->next!=NULL)
                ptr= ptr->next;
        ptr->next=new_node;
        return start;
}

struct node* insert_after(struct node * start)
{
        int num, val;
        printf("\n enter the data: ");
        scanf("%d",&num);
        printf("\n enter a value after which the new node is inserted: ");
        scanf("%d",&val);
        struct node* new_node, *ptr, *preptr;
        new_node= (struct node*)malloc(sizeof(struct node));
        new_node->data=num;
        ptr=start;
        preptr=ptr;
        while(preptr->data!=val)
        {
                preptr=ptr;
                ptr=ptr->next;
        }
        preptr->next=new_node;
        new_node->next=ptr;
        return start;
}
```

OUTPUT:

```
*****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: inserting a node at beginning of the list
 4: insert a node at the end of the list
 5: insert a node after a particular node in the list
 6: Exit
enter your option: 1

 enter the data or -1 to end: 10
 enter the data or -1 to end: 20
 enter the data or -1 to end: 30
 enter the data or -1 to end: 40
 enter the data or -1 to end: 50
 enter the data or -1 to end: -1

 linked list is created

 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: inserting a node at beginning of the list
 4: insert a node at the end of the list
 5: insert a node after a particular node in the list
 6: Exit
enter your option: 2
10      20      30      40      50

 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: inserting a node at beginning of the list
 4: insert a node at the end of the list
 5: insert a node after a particular node in the list
 6: Exit
enter your option: 3

 enter the data: 15


 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: inserting a node at beginning of the list
 4: insert a node at the end of the list
 5: insert a node after a particular node in the list
 6: Exit
enter your option: 2
15      10      20      30      40      50

 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: inserting a node at beginning of the list
 4: insert a node at the end of the list
 5: insert a node after a particular node in the list
 6: Exit
enter your option: 4

 enter the data: 5


 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: inserting a node at beginning of the list
 4: insert a node at the end of the list
 5: insert a node after a particular node in the list
 6: Exit
enter your option: 2
15      10      20      30      40      50      5

 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: inserting a node at beginning of the list
 4: insert a node at the end of the list
 5: insert a node after a particular node in the list
 6: Exit
enter your option: 5

 enter the data: 65

 enter a value after which the new node is inserted: 30


 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: inserting a node at beginning of the list
 4: insert a node at the end of the list
 5: insert a node after a particular node in the list
 6: Exit
enter your option: 2
15      10      20      30      65      40      50      5
```

4c) Write a C program that uses functions to perform deletion operation on a singly linked list

Aim: To show a C program that uses functions to perform deletion operation on a singly linked list

Program:

```
/* c) Use functions to perform deletion operation on a singly linked list*/
#include<stdio.h>
#include<stdlib.h>

struct node
{
 int data;
 struct node* next;
};

struct node* start = NULL;
struct node* createll(struct node *);
void display(struct node *);
struct node* delete_beg(struct node*);
struct node* delete_end(struct node*);
struct node* delete_after(struct node*);

void main()
{
        int option;
        do
        {
        printf("\n\n *****MAIN MENU *****");
        printf("\n 1: Create a list");
        printf("\n 2: Display the list");
        printf("\n 3: Delete a node at beginning of the list");
        printf("\n 4: Delete a node at end of the list");
        printf("\n 5: Delete a node after a particular node in list");
        printf("\n 6: Exit");
        printf("\nenter your option: ");
        scanf("%d",&option);
        switch(option)
          {
                case 1:
                        start = createll(start);
                        printf("\n linked list is created");
                        break;
                case 2:
                        display (start);
                        break;
                case 3:
                        start = delete_beg(start);
                        printf("\n first node successfully deleted");
                        break;
                case 4:
                        start = delete_end(start);
                        printf("\n Last node successfully deleted");
                        break;
        case 5:
            start = delete_after(start);
            printf("\n Node successfully deleted");
            break;
      }
    }while(option != 6);
}

struct node* createll(struct node* start)
{
    struct node *newnode;
    struct node *ptr;
    int num;
    printf("\n enter the data or -1 to end: ");
    scanf("%d",&num);
    while(num!= -1)
    {
        newnode = (struct node *)malloc(sizeof(struct node));
        newnode->data=num;
        if(start == NULL)
        {
            newnode->next=NULL;
            start=newnode;
        }
        else
        {
            ptr = start;
            while(ptr->next != NULL)
            {
                ptr = ptr->next;
            }
            ptr->next = newnode;
            newnode->next=NULL;
        }
        printf("\n enter the data or -1 to end: ");
        scanf("%d",&num);
    }
    return start;
}

void display(struct node* start)
{
    struct node * ptr;
    ptr=start;
    while (ptr != NULL)
    {
        printf("%d\t ", ptr->data);
        ptr=ptr->next;
    }
}

struct node * delete_beg(struct node * start)
```

```
{
        struct node * ptr;
        ptr=start;
        start=start->next;
        free(ptr);
        return start;
}

struct node *delete_end(struct node *start)
{
        struct node *ptr, *preptr;
        ptr = start;
        while(ptr -> next != NULL)
        {
                preptr = ptr;
                ptr = ptr -> next;
        }
        preptr -> next = NULL;
        free(ptr);
        return start;
}

struct node *delete_after(struct node *start)
{
        struct node *ptr, *preptr;
        int val;
        printf("\n Enter the value after which the node has to deleted : ");
        scanf("%d", &val);
        ptr = start;
        preptr = start;
        if(preptr->data == val) / to delete a node after a first
                ptr=ptr->next;
        while(preptr -> data != val)
        {
                preptr = ptr;
                ptr = ptr -> next;
        }

        preptr -> next=ptr -> next;
        free(ptr);
        return start;
}
```

OUTPUT:

```
*****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: Delete a node at beginning of the list
 4: Delete a node at end of the list
 5: Delete a node after a particular node in list
 6: Exit
enter your option: 1

 enter the data or -1 to end: 10
 enter the data or -1 to end: 20
 enter the data or -1 to end: 30
 enter the data or -1 to end: 40
 enter the data or -1 to end: 50
 enter the data or -1 to end: 60
 enter the data or -1 to end: -1

 linked list is created

 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: Delete a node at beginning of the list
 4: Delete a node at end of the list
 5: Delete a node after a particular node in list
 6: Exit
enter your option: 2
10      20      30      40      50      60

 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: Delete a node at beginning of the list
 4: Delete a node at end of the list
 5: Delete a node after a particular node in list
 6: Exit
enter your option: 3

 first node successfully deleted

 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: Delete a node at beginning of the list
 4: Delete a node at end of the list
 5: Delete a node after a particular node in list
 6: Exit
enter your option: 2
20      30      40      50      60

 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: Delete a node at beginning of the list
 4: Delete a node at end of the list
 5: Delete a node after a particular node in list
 6: Exit
enter your option: 4
```

```
 Last node successfully deleted

 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: Delete a node at beginning of the list
 4: Delete a node at end of the list
 5: Delete a node after a particular node in list
 6: Exit
enter your option: 2
20      30      40      50

 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: Delete a node at beginning of the list
 4: Delete a node at end of the list
 5: Delete a node after a particular node in list
 6: Exit
enter your option: 5

 Enter the value after which the node has to deleted : 30

 Node successfully deleted

 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: Delete a node at beginning of the list
 4: Delete a node at end of the list
 5: Delete a node after a particular node in list
 6: Exit
enter your option: 2
20      30      50
```

4d) Write a C program to reverse elements of a single linked list.

Aim: To show a C program to reverse elements of a single linked list.

Program:

```
/* d) Write a C program to reverse elements of a singly linked list*/
#include<stdio.h>
#include<stdlib.h>

struct node
{
 int data;
 struct node* next;
};

struct node* start = NULL;
struct node* createll(struct node *);
void display(struct node *);
struct node* reverse(struct node*);

void main()
{
        int option;
        do
        {
        printf("\n\n *****MAIN MENU *****");
        printf("\n 1: Create a list");
        printf("\n 2: Display the list");
        printf("\n 3: Reverse the list");
        printf("\n 4: Exit");
        printf("\nenter your option: ");
        scanf("%d",&option);
        switch(option)
          {
                case 1:
                        start = createll(start);
                        printf("\n linked list is created");
                        break;
                case 2:
                        display (start);
                        break;
                case 3:
                        start = reverse(start);
                        printf("\n Linked list successfully reversed");
                        break;
          }
        }while(option != 4);
}

struct node* createll(struct node* start)
{
    struct node *newnode;
    struct node *ptr;
    int num;
    printf("\n enter the data or -1 to end: ");
    scanf("%d",&num);
    while(num!= -1)
    {
        newnode = (struct node *)malloc(sizeof(struct node));
        newnode->data=num;
        if(start == NULL)
        {
            newnode->next=NULL;
            start=newnode;
        }
        else
        {
            ptr = start;
            while(ptr->next != NULL)
            {
                ptr = ptr->next;
            }
            ptr->next = newnode;
            newnode->next=NULL;
        }
        printf("\n enter the data or -1 to end: ");
        scanf("%d",&num);
    }
    return start;
}

void display(struct node* start)
{
    struct node * ptr;
    ptr=start;
    while (ptr != NULL)
    {
        printf("%d\t ", ptr->data);
        ptr=ptr->next;
    }
}

struct node * reverse(struct node * start)
{
    struct node * ptr, *preptr=NULL, *postptr=NULL;
    ptr=start;
    while(ptr!=NULL)
    {
        postptr=ptr->next;
        ptr->next= preptr;
        preptr=ptr;
        ptr=postptr;
    }
```

```
        start=preptr;
        return start;
}
```

OUTPUT:

```
*****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: Reverse the list
 4: Exit
enter your option: 1

 enter the data or -1 to end: 10
 enter the data or -1 to end: 20
 enter the data or -1 to end: 30
 enter the data or -1 to end: 40
 enter the data or -1 to end: -1

 linked list is created

 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: Reverse the list
 4: Exit
enter your option: 2
10      20      30      40

 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: Reverse the list
 4: Exit
enter your option: 3

 Linked list successfully reversed

 *****MAIN MENU *****
 1: Create a list
 2: Display the list
 3: Reverse the list
 4: Exit
enter your option: 2
40      30      20      10
```

EXERCISE – 5

5a) Write C program that implement Queue (its operations) using Array.

Aim: To Show a C program that implement Queue (its operations) using Array.

Program:

```
/* C program that implement Queue (its operations) using arrays
#include<stdio.h>
#include<stdbool.h>

int front=-1, rear=-1;

void enqueue(int n, int arr[*]);
int dequeue(int n, int arr[*]);
int peek(int n, int arr[*]);
bool isEmpty(int n, int arr[*]);
bool isFull(int n, int arr[*]);
void display(int n, int arr[*]);

void main()
{
        int option,val,MAX;
        printf("enter the size of array: ");
        scanf("%d",&MAX);
        int queue[MAX];
        bool x;
        do
        {
        printf("\n\n **** MAIN MENU ****");
        printf("\n 1. Insert an element into queue");
        printf("\n 2. Delete an element from queue");
        printf("\n 3. Peek value of queue");
        printf("\n 4. Is queue empty");
        printf("\n 5. Is queue full");
        printf("\n 6. Display the queue");
        printf("\n 7. EXIT");
        printf("\n Enter your option: ");
        scanf("%d",&option);
        switch(option)
        {
        case 1:
                enqueue(MAX, queue);
                break;
        case 2:
                val=dequeue(MAX, queue);
                if(val != -1)
                        printf("\n the number deleted is : %d",val);
                break;
        case 3:
```

```
        val=peek(MAX, queue);
        if(val != -1)
            printf("\n the first value in queue is : %d",queue[val]);
        break;
    case 4:
        x=isEmpty(MAX, queue);
        if(x==true)
            printf("\n the queue is empty");
        else
            printf("\n the queue is not empty");
        break;
    case 5:
        x=isFull(MAX, queue);
        if(x==true)
            printf("\n the queue is Full");
        else
            printf("\n the queue is not Full");
        break;
    case 6:
        display(MAX, queue);
        break;
    default:
        printf("enter correct option!");
      }
    }while(option != 7);
}

//this function is used to insert an element into queue.
void enqueue(int MAX, int queue[MAX])
{
    int num;
    printf("\n enter the number to be inserted in the queue: ");
    scanf("%d",&num);
    if(rear== MAX-1)
        printf("\n Overflow");
    else if(front == -1 && rear == -1)
        front = rear = 0;
    else
        rear++;
    queue[rear]=num;
}

// this function is used to delete an element from a queue.
int dequeue(int MAX, int queue[MAX])
{
    int val;
    if(front == -1 || front > rear)
    {
        printf("\n underflow");
        return -1;
    }
    else
```

```
        {
                val = queue[front];
                front++;
                if(front > rear)
                        front = rear = -1;
                return val;
        }
}

//this function is used to print the front element of the queue
int peek(int MAX, int queue[MAX])
{
        if(front == -1 || front > rear)
        {
                printf("\n queue is empty");
                return -1;
        }
        else
                return front;
}

//this function is used to check the queue is empty or not.
bool isEmpty(int MAX, int queue[MAX])
{
        if(front==-1 || front > rear)
                return true;
        else
                return false;
}

// this function is used to check the queue is full or not.
bool isFull(int MAX, int queue[MAX])
{
        if(rear == MAX-1)
                return true;
        else
                return false;
}

// this function is used to display the element of the queue
void display(int MAX, int queue[MAX])
{
        int i;
        if(front == -1 || front >rear)
                printf("\n queue is empty");
        else
        {
        for(i=front; i<= rear; i++)
                printf("\n%d \t",queue[i]);
        }
}
```

OUTPUT:

```
enter the size of array: 4


 **** MAIN MENU ****
 1. Insert an element into queue
 2. Delete an element from queue
 3. Peek value of queue
 4. Is queue empty
 5. Is queue full
 6. Display the queue
 7. EXIT
 Enter your option: 1
enter the number to be inserted in the queue: 10


 **** MAIN MENU ****
 1. Insert an element into queue
 2. Delete an element from queue
 3. Peek value of queue
 4. Is queue empty
 5. Is queue full
 6. Display the queue
 7. EXIT
 Enter your option: 1
enter the number to be inserted in the queue: 20


 **** MAIN MENU ****
 1. Insert an element into queue
 2. Delete an element from queue
 3. Peek value of queue
 4. Is queue empty
 5. Is queue full
 6. Display the queue
 7. EXIT
 Enter your option: 1
 enter the number to be inserted in the queue: 30


 **** MAIN MENU ****
 1. Insert an element into queue
 2. Delete an element from queue
 3. Peek value of queue
 4. Is queue empty
 5. Is queue full
 6. Display the queue
 7. EXIT
 Enter your option: 1
 enter the number to be inserted in the queue: 40
 **** MAIN MENU ****
 1. Insert an element into queue
 2. Delete an element from queue
 3. Peek value of queue
 4. Is queue empty
 5. Is queue full
 6. Display the queue
 7. EXIT
 Enter your option: 6
10
20
30
40

 **** MAIN MENU ****
 1. Insert an element into queue
 2. Delete an element from queue
 3. Peek value of queue
 4. Is queue empty
 5. Is queue full
 6. Display the queue
 7. EXIT
 Enter your option: 3
 the first value in queue is : 10

 **** MAIN MENU ****
 1. Insert an element into queue
 2. Delete an element from queue
 3. Peek value of queue
 4. Is queue empty
 5. Is queue full
 6. Display the queue
 7. EXIT
 Enter your option: 2
 the number deleted is : 10


**** MAIN MENU ****
 1. Insert an element into queue
 2. Delete an element from queue
 3. Peek value of queue
 4. Is queue empty
 5. Is queue full
 6. Display the queue
 7. EXIT
 Enter your option: 6
20
30
40
```

```
**** MAIN MENU ****
1. Insert an element into queue
2. Delete an element from queue
3. Peek value of queue
4. Is queue empty
5. Is queue full
6. Display the queue
7. EXIT
Enter your option: 4

the queue is not empty

**** MAIN MENU ****
1. Insert an element into queue
2. Delete an element from queue
3. Peek value of queue
4. Is queue empty
5. Is queue full
6. Display the queue
7. EXIT
Enter your option: 5
the queue is Full
```

5b) Write C program that implement Queue (its operations) using Linkedlist.

Aim: To Show a C program that implement Queue (its operations) using Linkedlist.

Program:

```
/* C program that implement Queue (its operations) using Linkedlist
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
struct node
{
 int data;
 struct node *next;
};

struct node *front=NULL;
struct node *rear=NULL;

struct node *insert(struct node *,int);
struct node *delete_element(struct node *);
struct node *display(struct node *);
void peek(struct node*);
bool isEmpty(struct node*);


void main()
{
        int val, option;
        bool x;
        do
        {
        printf("\n\n *****MAIN MENU*****");
        printf("\n 1. Insert an element ");
        printf("\n 2. Delete an element");
        printf("\n 3. Peek value");
        printf("\n 4. Check queue is empty");
        printf("\n 5. Display the queue");
        printf("\n 6. EXIT");
        printf("\n Enter your option : ");
        scanf("%d", &option);
        switch(option)
        {
        case 1:
                printf("\n Enter the number to insert in the queue:");
                scanf("%d", &val);
                front = insert(front,val);
                break;
        case 2:
                front = delete_element(front);
                break;
        case 3:
                peek(front);
                break;
        case 4:
                x=isEmpty(front);
                if(x==true)
                    printf("\n the queue is empty");
                else
                    printf("\n the queue is not empty");
                break;
        case 5:
                front = display(front);
                break;
        default:
                printf("enter correct option!");
              }
        }while(option !=6);
}

//this function is used to insert an element in to a queue
struct node *insert(struct node * front,int val)
{
        struct node *ptr;
        ptr = (struct node*)malloc(sizeof(struct node));
        ptr -> data = val;
        if( front == NULL)
        {
                front = ptr;
                rear = ptr;
                front -> next = rear -> next = NULL;
        }
        else
        {
                rear -> next = ptr;
                rear = ptr;
                rear -> next = NULL;
        }
        return front;
}

//this function is used to delete an element an element from a queue.
struct node *delete_element(struct node *front)
{
        struct node *ptr;
        ptr = front;
        if(front == NULL)
                printf("\n UNDERFLOW");
        else
        {
                front = front -> next;
                printf("\n The value being deleted is : %d", ptr -> data);
                free(ptr);
        }
```

```c
    return front;
}

//this function displays front value.
void peek(struct node *front)
{
    if(front==NULL)
        printf("\n QUEUE IS EMPTY");
    else
        printf("\nPeek = %d",front->data);
}

//this function checks the queue is empty or not.
bool isEmpty(struct node * front)
{
    if(front==NULL)
        return true;
    else
        return false;
}

// this function is used to display the elements of queue
struct node *display(struct node *front)
{
    struct node *ptr;
    ptr = front;
    if(ptr == NULL)
        printf("\n QUEUE IS EMPTY");
    else
    {
        printf("\n");
        while(ptr!=rear)
        {
            printf("%d\t", ptr -> data);
            ptr = ptr -> next;
        }
        printf("%d\t", ptr -> data);
    }
    return front;
}
```

OUTPUT:

```
*****MAIN MENU*****
 1. Insert an element
 2. Delete an element
 3. Peek value
 4. Check queue is empty
 5. Display the queue
 6. EXIT
 Enter your option : 1
 Enter the number to insert in the queue:10
```

```
*****MAIN MENU*****
1. Insert an element
2. Delete an element
3. Peek value
4. Check queue is empty
5. Display the queue
6. EXIT
Enter your option : 1
Enter the number to insert in the queue:20


*****MAIN MENU*****
1. Insert an element
2. Delete an element
3. Peek value
4. Check queue is empty
5. Display the queue
6. EXIT
Enter your option : 1
Enter the number to insert in the queue:30


*****MAIN MENU*****
1. Insert an element
2. Delete an element
3. Peek value
4. Check queue is empty
5. Display the queue
6. EXIT
Enter your option : 1
Enter the number to insert in the queue:40


*****MAIN MENU*****
1. Insert an element
2. Delete an element
3. Peek value
4. Check queue is empty
5. Display the queue
6. EXIT
Enter your option : 5
10      20      30      40

*****MAIN MENU*****
1. Insert an element
2. Delete an element
3. Peek value
4. Check queue is empty
5. Display the queue
6. EXIT
```

```
 Enter your option : 2
 The value being deleted is : 10

 *****MAIN MENU*****
 1. Insert an element
 2. Delete an element
 3. Peek value
 4. Check queue is empty
 5. Display the queue
 6. EXIT
 Enter your option : 5
20      30      40

 *****MAIN MENU*****
 1. Insert an element
 2. Delete an element
 3. Peek value
 4. Check queue is empty
 5. Display the queue
 6. EXIT
 Enter your option : 3
Peek = 20

 *****MAIN MENU*****
 1. Insert an element
 2. Delete an element
 3. Peek value
 4. Check queue is empty
 5. Display the queue
 6. EXIT
 Enter your option : 4
 the queue is not empty
```

EXERCISE – 6

6a) Write C program that implement stack (its operations) using arrays

Aim: To show a C program that implement stack (its operations) using arrays

Program:

```
// C program that implement stack (its operations) using arrays
#include <stdio.h>
int top=-1;

void push(int MAX, int stack[*]);
int pop(int MAX, int stack[*]);
void peek(int MAX, int stack[*]);
void display(int MAX, int stack[*]);
void isEmpty(int MAX, int stack[*]);
void isFull(int MAX, int stack[*]);

void main( )
{
        int val, option,MAX;
        printf("enter the size of stack: ");
        scanf("%d",&MAX);
        int stack[MAX];
        do
        {
        printf("\n *****MAIN MENU*****");
        printf("\n 1. PUSH");
        printf("\n 2. POP");
        printf("\n 3. PEEK");
        printf("\n 4. DISPLAY");
        printf("\n 5. To check the stack is Empty");
        printf("\n 6. To check the stack is Full");
        printf("\n 7. EXIT");
        printf("\n Enter your option: ");
        scanf("%d", &option);
        switch(option)
        {
        case 1:
                push(MAX, stack);
                break;
        case 2:
                val = pop(MAX, stack);
                if(val != -1)
                        printf("\n The value deleted from stack is: %d", val);
                        break;
        case 3:
                peek(MAX, stack);
                break;
        case 4:
                display(MAX, stack);
                break;
        case 5:
                isEmpty(MAX, stack);
                break;
        case 6:
                isFull(MAX, stack);
                break;
        default:
                printf("enter correct option!!!");
        }
  }while(option != 7);
 }


void push(int MAX, int stack[MAX])
{
        int val;
        printf("\n Enter the number to be pushed on stack: ");
        scanf("%d", &val);
        if(top == MAX-1)
                printf("\n STACK OVERFLOW");
        else
        {
                top++;
                stack[top] = val;
        }
 }

int pop(int MAX, int stack[MAX])
{
        int val;
        if(top == -1)
        {
                printf("\n STACK UNDERFLOW");
                return -1;
        }
        else
        {
                val = stack[top];
                top--;
                return val;
        }
}

void peek(int MAX, int stack[MAX])
{
        if(top == -1)
        {
                printf("\n STACK IS EMPTY");
        }
        else
                printf("\n The value stored at top of stack is: %d", stack[top
}

void display(int MAX, int stack[MAX])
{
        int i;
        if(top == -1)
                printf("\n STACK IS EMPTY");
        else
        {
                for(i=0;i<=top;i++)
                        printf("\t %d",stack[i]);
        }
}

void isEmpty(int MAX, int stack[MAX])
{
        if(top == -1)
                printf("Stack is empty");
        else
                printf("Stack is not empty");
}

void isFull(int MAX, int stack[MAX])
{
        if(top == MAX-1)
                printf("\n the stack is Full");
        else
                printf("\n the stack is not Full");
}
```

OUTPUT:

```
*****MAIN MENU*****
 1. PUSH
 2. POP
 3. PEEK
 4. DISPLAY
 5. EXIT
 Enter your option: 1
 Enter the number to be pushed on stack: 10
 *****MAIN MENU*****
 1. PUSH
 2. POP
 3. PEEK
 4. DISPLAY
 5. EXIT
 Enter your option: 1
 Enter the number to be pushed on stack: 20

 *****MAIN MENU*****
 1. PUSH
 2. POP
 3. PEEK
 4. DISPLAY
```

```
5. EXIT
 Enter your option: 1
 Enter the number to be pushed on stack: 30

 *****MAIN MENU*****
 1. PUSH
 2. POP
 3. PEEK
 4. DISPLAY
 5. EXIT
 Enter your option: 1
 Enter the number to be pushed on stack: 40

 *****MAIN MENU*****
 1. PUSH
 2. POP
 3. PEEK
 4. DISPLAY
 5. EXIT
 Enter your option: 4
         10      20      30      40

 *****MAIN MENU*****
 1. PUSH
 2. POP
 3. PEEK
 4. DISPLAY
 5. EXIT
 Enter your option: 3

 The value stored at top of stack is: 40
 *****MAIN MENU*****
 1. PUSH
 2. POP
 3. PEEK
 4. DISPLAY
 5. EXIT
 Enter your option: 2
 The value deleted from stack is: 40

 *****MAIN MENU*****
 1. PUSH
 2. POP
 3. PEEK
 4. DISPLAY
 5. EXIT
 Enter your option: 4
         10      20      30
```

6b) Write C program that implement stack (its operations) using Linked list

Aim: To show a C program that implement stack (its operations) using Linked list

Program:

```
//C program that implement stack (its operations) using LinkedList
#include <stdio.h>
#include <stdlib.h>

struct node
{
 int data;
 struct node *next;
};

struct node *top = NULL;

struct node *push(struct node *, int);
struct node *display(struct node *);
struct node *pop(struct node *);
int peek(struct node *);

void main()
{
        int val, option;
        do
        {
        printf("\n *****MAIN MENU*****");
        printf("\n 1. PUSH");
        printf("\n 2. POP");
        printf("\n 3. PEEK");
        printf("\n 4. DISPLAY");
        printf("\n 5. EXIT");
        printf("\n Enter your option: ");
        scanf("%d", &option);
        switch(option)
        {
        case 1:
                printf("\n Enter the number to be pushed on stack: ");
                scanf("%d", &val);
                top = push(top, val);
                break;
        case 2:
                top = pop(top);
                break;
        case 3:
                val = peek(top);
                if (val != -1)
                        printf("\n The value at the top of stack is: %d", val);
                else
                        printf("\n STACK IS EMPTY");
                break;
        case 4:
                top = display(top);
                break;
         }
        }while(option != 5);
}


struct node *push(struct node *top, int val)
{
        struct node* ptr;
        ptr=(struct node*)malloc(sizeof(struct node));
        ptr->data=val;
        if(top==NULL)
        {
                ptr->next=NULL;
                top=ptr;
        }
        else
        {
                ptr->next=top;
                top=ptr;
        }
        return top;
}


struct node *pop(struct node *top)
{
        struct node* ptr = top;
        if(top==NULL)
                printf("\n UNDERFLOW");
        else
        {
                top=top->next;
                printf("\n the value deleted is %d",ptr->data);
                free(ptr);
        }
        return top;
}


int peek(struct node *top)
{
        if(top == NULL)
                return -1;
        else
                return top->data;
}


struct node *display(struct node *top)
{
        struct node* ptr=top;
        if(top==NULL)
                printf("\nUNDERFLOW");
        else
        {
                while(ptr!=NULL)
                {
                    printf("\t %d",ptr->data);
                    ptr=ptr->next;
                }
        }
        return top;
}
```

OUTPUT:

```
*****MAIN MENU*****
 1. PUSH
 2. POP
 3. PEEK
 4. DISPLAY
 5. EXIT
 Enter your option: 1
 Enter the number to be pushed on stack: 10

 *****MAIN MENU*****
 1. PUSH
 2. POP
 3. PEEK
 4. DISPLAY
 5. EXIT
 Enter your option: 1
 Enter the number to be pushed on stack: 20

 *****MAIN MENU*****
 1. PUSH
 2. POP
 3. PEEK
 4. DISPLAY
 5. EXIT
 Enter your option: 1
 Enter the number to be pushed on stack: 30

 *****MAIN MENU*****
 1. PUSH
 2. POP
 3. PEEK
 4. DISPLAY
 5. EXIT
 Enter your option: 1
 Enter the number to be pushed on stack: 40
```

```
*****MAIN MENU*****
1. PUSH
2. POP
3. PEEK
4. DISPLAY
5. EXIT
Enter your option: 4
40        30      20      10

*****MAIN MENU*****
1. PUSH
2. POP
3. PEEK
4. DISPLAY
5. EXIT
Enter your option: 3
The value at the top of stack is: 40

*****MAIN MENU*****
1. PUSH
2. POP
3. PEEK
4. DISPLAY
5. EXIT
Enter your option: 2

the value deleted is 40

*****MAIN MENU*****
1. PUSH
2. POP
3. PEEK
4. DISPLAY
5. EXIT
Enter your option: 4
30        20      10
```

6c) Write a C program that uses Stack operations to evaluate postfix expression

Aim: To show a C program that uses Stack operations to evaluate postfix expression

Program:

```
// c program to evaluate postfix expression
#include <stdio.h>
#include <ctype.h>
#define MAX 100
float st[MAX];
int top=-1;

void push(float st[], float val);
float pop(float st[]);
float evaluatePostfixExp(char exp[]);

int main()
{
    float val;
    char exp[100];
    printf("\n Enter any postfix expression : ");
    gets(exp);
    val = evaluatePostfixExp(exp);
    printf("\n Value of the postfix expression = %.2f", val);
    return 0;
}

float evaluatePostfixExp(char exp[])
{
    int i=0;
    float op1, op2, value;
    while(exp[i] != '\0')
    {
        if(isdigit(exp[i]))
            push(st, (float)(exp[i]-'0'));
        else
        {
            op2 = pop(st);
            op1 = pop(st);
            switch(exp[i])
            {
            case '+':
                value = op1 + op2;
                break;
            case '-':
                value = op1 - op2;
                break;
            case '/':
                value = op1 / op2;
                break;
            case '*':
                value = op1 * op2;
                break;
            case '%':
                value = (int)op1 % (int)op2;
                break;
        }
        push(st, value);
      }
      i++;
    }
    return(pop(st));
}

void push(float st[], float val)
{
    if(top==MAX-1)
        printf("\n STACK OVERFLOW");
    else
    {
        top++;
        st[top]=val;
    }
}

float pop(float st[])
{
    float val=-1;
    if(top==-1)
        printf("\n STACK UNDERFLOW");
    else
    {
        val=st[top];
        top--;
    }
    return val;
}
```

OUTPUT:

```
Enter any postfix expression : 934*8+4/-
 Value of the postfix expression = 4.00
```

EXERCISE – 7

7a) Write a recursive C program for traversing a binary tree in preorder, inorder and postorder.

Aim: To show a recursive C program for traversing a binary tree in preorder, inorder and postorder.

Program:

```
//A recursive C program for traversing a binary tree in preorder,inorder, postorder.
#include<stdio.h>
#include<stdlib.h>

struct node
{
 char data;
 struct node *left;
 struct node *right;
};

struct node *root=NULL;

struct node *create();
void inorder(struct node*);
void preorder(struct node*);
void postorder(struct node*);

void main()
{

        int option;
        do
        {
        printf("\n\n *****MAIN MENU*****");
        printf("\n 1. create a binary tree ");
        printf("\n 2. In-order traversal");
        printf("\n 3. Pre-order traversal");
        printf("\n 4. Post-order traversal");
        printf("\n 5. EXIT");
        printf("\n Enter your option : ");
        scanf("%d", &option);
        switch(option)
          {
        case 1:
                root = create();
                break;
        case 2:
                printf("\n the elements after inorder traversal: ");
                inorder(root);
                break;
        case 3:
                printf("\n the elements after preorder traversal: ");
                preorder(root);
                break;
        case 4:
                printf("\n the elements after postorder traversal: ");
                postorder(root);
                break;
          }
        }while(option != 5);
}

struct node *create()
{
        struct node *p;
        char x;
        printf("Enter data(0 for no data):");
        scanf(" %c",&x);
        if(x=='0')
                return NULL;
        p=(struct node*)malloc(sizeof(struct node));
        p->data=x;
        printf("Enter left child of %c:\n",x);
        p->left=create();
        printf("Enter right child of %c:\n",x);
        p->right=create();
        return p;
}

void inorder(struct node *root)
{
        if(root!=NULL)
        {
                inorder(root->left);
                printf(" %2c",root->data);
                inorder(root->right);
        }
}

void preorder(struct node *root)
{
        if(root!=NULL)
        {
                printf(" %2c",root->data);
                preorder(root->left);
                preorder(root->right);
        }
}

void postorder(struct node *root)
{
        if(root!=NULL)
        {
                postorder(root->left);
                postorder(root->right);
                printf(" %2c",root->data);
        }
}
```

OUTPUT:

```
*****MAIN MENU*****
 1. create a binary tree
 2. In-order traversal
 3. Pre-order traversal
 4. Post-order traversal
 5. EXIT
 Enter your option : 1
Enter data(0 for no data):a
Enter left child of a:
Enter data(0 for no data):b
Enter left child of b:
Enter data(0 for no data):d
Enter left child of d:
Enter data(0 for no data):0
Enter right child of d:
Enter data(0 for no data):0
Enter right child of b:
Enter data(0 for no data):e
Enter left child of e:
Enter data(0 for no data):0
Enter right child of e:
Enter data(0 for no data):0
Enter right child of a:
Enter data(0 for no data):c
Enter left child of c:
Enter data(0 for no data):f
Enter left child of f:
Enter data(0 for no data):0
Enter right child of f:
Enter data(0 for no data):0
Enter right child of c:
Enter data(0 for no data):g
Enter left child of g:
Enter data(0 for no data):0
Enter right child of g:
Enter data(0 for no data):0

 *****MAIN MENU*****
 1. create a binary tree
 2. In-order traversal
 3. Pre-order traversal
 4. Post-order traversal
 5. EXIT
 Enter your option : 2
```

```
the elements after inorder traversal:    d  b  e  a  f  c  g

*****MAIN MENU*****
1. create a binary tree
2. In-order traversal
3. Pre-order traversal
4. Post-order traversal
5. EXIT
Enter your option : 3

the elements after preorder traversal:    a  b  d  e  c  f  g

*****MAIN MENU*****
1. create a binary tree
2. In-order traversal
3. Pre-order traversal
4. Post-order traversal
5. EXIT
Enter your option : 4

the elements after postorder traversal:    d  e  b  f  g  c  a
```

EXERCISE – 8

8a) Write a C program to Create a BST

Aim: To show a C program to Create a BST

Program:

```
// C program to create a binary search tree
#include<stdio.h>
#include<stdlib.h>

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

struct node* root = NULL;

struct node *create(struct node* );
void display(struct node* );

void main()
{
        int option, val;
        do
        {
        printf("\n\n *** MAIN MENU ***");
        printf("\n 1. create");
        printf("\n 2. display");
        printf("\n 3. exit");
        printf("\n enter your choice: ");
        scanf("%d",&option);
        switch(option)
          {
                case 1:
                        root = create(root);
                        break;
                case 2:
                        printf("\n the elements in the BST: ");
                        display(root);
                        break;
          }
        }while(option<3);
}

struct node* create(struct node* root)
{
        int val;
        printf("\n enter data (-1 to stop): ");
```

```
    scanf("%d",&val);
    while(val != -1)
    {
        struct node *ptr, *nodeptr, *parentptr;
        ptr=(struct node*)malloc(sizeof(struct node));
        ptr->data=val;
        ptr->left=NULL;
        ptr->right=NULL;
        if(root==NULL)
            root=ptr;
        else
        {
            parentptr=NULL;
            nodeptr=root;
            while(nodeptr!=NULL)
            {
                parentptr=nodeptr;
                if(val<nodeptr->data)
                    nodeptr=nodeptr->left;
                else
                    nodeptr=nodeptr->right;
            }
            if(val<parentptr->data)
                parentptr->left=ptr;
            else
                parentptr->right=ptr;
        }
    printf("\n enter data (-1 to stop): ");
    scanf("%d",&val);
    }
    return root;
}

void display(struct node* root)
{
    if(root != NULL)
    {
        display(root->left);
        printf("%4d", root->data);
        display(root->right);
    }
}
```

OUTPUT:

```
*** MAIN MENU ***
 1. create
 2. display
 3. exit
 enter your choice: 1

 enter data (-1 to stop): 10
 enter data (-1 to stop): 20

 enter data (-1 to stop): 30

 enter data (-1 to stop): 50

 enter data (-1 to stop): 40

 enter data (-1 to stop): 45

 enter data (-1 to stop): -1


 *** MAIN MENU ***
 1. create
 2. display
 3. exit
 enter your choice: 2

 the elements in the BST:   10  20  30  40  45  50
```

8b) Write a C program to insert a node into a BST.

Aim: To show C program to insert a node into a BST.

Program:

```
// C program to insert a new node in a BST
#include<stdio.h>
#include<stdlib.h>

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

struct node *root=NULL;

struct node *create(struct node *);
void display(struct node *);
struct node* insert(struct node*, int);

void main()
{
        int option,val;
        do
        {
        printf("\n\n *** MAIN MENU *** ");
        printf("\n 1. create");
        printf("\n 2. display");
        printf("\n 3. insert a new node");
        printf("\n 4. Quit");
        printf("\n Enter your choice: ");
        scanf("%d",&option);
        switch (option)
          {
                case 1:
                        root = create(root);
                        break;
                case 2:
                        printf("\n The elements in the tree are\n");
                        display(root);
                        break;
                case 3:
                        printf("\n Enter an element to insert in BST: ");
                        scanf("%d",&val);
                        root= insert(root, val);
                        break;
                default:
                        printf("enter correct option!");
          }
        }while(option<4);
}

```

```

struct node* create(struct node* root)
{
    int val;
    printf("\n enter data (-1 to stop): ");
    scanf("%d",&val);
    while(val != -1)
    {
        struct node *ptr, *nodeptr, *parentptr;
        ptr = (struct node*)malloc(sizeof(struct node));
        ptr->data = val;
        ptr->left = NULL;
        ptr->right = NULL;
        if(root==NULL)
        {
            root=ptr;
        }
        else
        {
            parentptr=NULL;
            nodeptr=root;
            while(nodeptr!=NULL)
            {
                parentptr=nodeptr;
                if(val<nodeptr->data)
                    nodeptr=nodeptr->left;
                else
                    nodeptr = nodeptr->right;
            }
            if(val<parentptr->data)
                parentptr->left = ptr;
            else
                parentptr->right = ptr;
        }
        printf("\n enter data (-1 to stop): ");
        scanf("%d",&val);
    }
    return root;
}

void display(struct node *root)
{
    if(root!=NULL)
    {
        display(root->left);
        printf(" %5d",root->data);
        display (root->right);
    }
}

struct node* insert(struct node* root, int val)
{
	if(root==NULL)
	{
		root=(struct node*)malloc(sizeof(struct node));
		root->data=val;
		root->right=root->left=NULL;
		return root;
	}
	else if (val < root->data)
		root->left = insert(root->left, val);
	else
		root->right = insert(root->right, val);
	return root;
}
```

OUTPUT:

```
*** MAIN MENU ***
 1. create
 2. display
 3. insert a new node
 4. Quit
 Enter your choice: 1

 enter data (-1 to stop): 20

 enter data (-1 to stop): 10

 enter data (-1 to stop): 30

 enter data (-1 to stop): 40

 enter data (-1 to stop): 25

 enter data (-1 to stop): 50

 enter data (-1 to stop): -1


 *** MAIN MENU ***
 1. create
 2. display
 3. insert a new node
 4. Quit
 Enter your choice: 2
 The elements in the tree are
10    20    25    30    40    50

 *** MAIN MENU ***
 1. create
 2. display
 3. insert a new node
 4. Quit
```

```
Enter your choice: 3
Enter an element to insert in BST: 35

*** MAIN MENU ***
1. create
2. display
3. insert a new node
4. Quit
Enter your choice: 2

The elements in the tree are
   10    20    25    30    35    40    50
```

8c) Write a C program to delete a node from a BST.

Aim: To show a C program to delete a node from a BST.

Program:

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

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

struct node* root = NULL;

struct node *create(struct node* );
void display(struct node* );
struct node* delete_node(struct node*, int);
struct node* findminvalue(struct node*);

void main()
{
	int option, val;
	struct node* ptr;
	do
	{
	printf("\n\n *** MAIN MENU ***");
	printf("\n 1. create a BST");
	printf("\n 2. display a BST");
	printf("\n 3. Delete a node from a BST");
	printf("\n 4. Quit");
	printf("\n enter your choice: ");
	scanf("%d",&option);
	switch(option)
	  {
		case 1:
			root = create(root);
			break;
		case 2:
			if(root == NULL)
				printf("\n NO elements in the BST ");
			else
			{
				printf("\n the elements in the BST: ");
				display(root);
			}
			break;
		case 3:
			printf("\n enter a value to delete from a BST: ");
			scanf("%d", &val);
```

```
            root=delete_node(root, val);
            break;
        default:
            printf("\nenter valid option!");
      }
    }while(option!=4);
}

struct node* create(struct node* root)
{
    int val;
    printf("\n enter data (-1 to stop): ");
    scanf("%d",&val);
    while(val != -1)
    {
        struct node *ptr, *nodeptr, *parentptr;
        ptr=(struct node*)malloc(sizeof(struct node));
        ptr->data=val;
        ptr->left=NULL;
        ptr->right=NULL;
        if(root==NULL)
            root=ptr;
        else
        {
            parentptr=NULL;
            nodeptr=root;
            while(nodeptr!=NULL)
            {
                parentptr=nodeptr;
                if(val<nodeptr->data)
                    nodeptr=nodeptr->left;
                else
                    nodeptr=nodeptr->right;
            }
            if(val<parentptr->data)
                parentptr->left=ptr;
            else
                parentptr->right=ptr;
        }
        printf("\n enter data (-1 to stop): ");
        scanf("%d",&val);
    }
    return root;
}

void display(struct node* root)
{
    if(root != NULL)
    {
        display(root->left);
        printf("%4d", root->data);
        display(root->right);
        }
}

struct node* delete_node(struct node* root, int val)
{
        if(root==NULL)
                return root;
        //If the val to be deleted is smaller than the root go to left subtree
        if(val<root->data)
                root->left=delete_node(root->left, val);
        //If the val to be deleted is greater than the root go to right subtree
        else if(val>root->data)
                root->right=delete_node(root->right, val);
        else
        {
        //a node with one child or null
                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;
                }
// a node with two child
                struct node* temp=findminvalue(root->right);
                root->data=temp->data;
                root->right=delete_node(root->right,temp->data);
        }
        return root;
}

struct node* findminvalue(struct node* ptr)
{
        struct node* current=ptr;
        while(current->left!=NULL)
                current = current->left;
        return current;
}
```

OUTPUT:

```
*** MAIN MENU ***
 1. create a BST
 2. display a BST
```

```
3. Delete a node from a BST
4. Quit
enter your choice: 1

enter data (-1 to stop): 20
enter data (-1 to stop): 30
enter data (-1 to stop): 40
enter data (-1 to stop): 60
enter data (-1 to stop): 50
enter data (-1 to stop): 25
enter data (-1 to stop): -1

*** MAIN MENU ***
1. create a BST
2. display a BST
3. Delete a node from a BST
4. Quit
enter your choice: 2
the elements in the BST:   20  25  30  40  50  60

*** MAIN MENU ***
1. create a BST
2. display a BST
3. Delete a node from a BST
4. Quit
enter your choice: 3
enter a value to delete from a BST: 40


*** MAIN MENU ***
1. create a BST
2. display a BST
3. Delete a node from a BST
4. Quit
enter your choice: 2
the elements in the BST:   20  25  30  50  60
```