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
{
```


Reg. No:

```
40:         int i,found=0,pos=-1;
41:         for(i=0;i<n;i++)
42:         {
43:              if(arr[i]==val)
44:             {
45:                       found=1;
46:                       pos=i;
47:                       printf("\nelement %d is found at position %d using NonRecursion",val,i+1);
48:               break;
49:              }
50:         }
51:         if(found==0)
52:                   printf("\nelement %d does not found using Non-Recursion",val
53: }

54: int linear_search_re(int val,int n,int arr[n]) //Recursive function
55: {
56:         if(n>0)
57:         {
58:                   if(arr[n-1]==val)
59:                             return n;
60:                   else
61:                             return linear_search_re(val,n-1,arr);
62:         }
63:         return 0;
64: }
```

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:

```
1: #include<stdio.h>
2: int binaryrecur(int n,int arr[*],int beg, int end, int val);        //Recursive function
3: void binarynonrecur(int n, int arr[*],int beg, int end, int val);          //Non-Recursive
4: void main()
5: {
6:  int n,i,val,pos,option;
7:  printf("enter the size of array: ");
8:  scanf("%d",&n);
9:  int arr[n];
10:         printf("enter the elements in ascending order: ");
11:         for(i=0;i<n;i++)
12:                 scanf("%d",&arr[i]);
13:         printf("enter the value to search: ");
14:         scanf("%d",&val);
15:         printf("enter 1 for Non-Recursive Function \n");
16:         printf("enter 2 for Recursive Function\n");
17:         scanf("%d",&option);
18:         switch(option)
19:         {
20:         case 1:
21:           {
22:                 binarynonrecur(n,arr,0,n-1,val);
23:                 break;
24:           }
25:         case 2:
26:           {
27:             pos=binaryrecur(n,arr,0,n-1,val);
28:             if(pos!=-1)
29:                     printf("\nelement %d found at %d position using Recursion",val,pos+1);
30:             else
31:                     printf("\nelement %d not found using Recursion",val);
32:             break;
33:           }
34:     default:
35:                 printf("enter correct option! ");
36:         }
37: }

38: int binaryrecur(int n,int arr[n],int beg,int end,int val)        //Recursion Function
39: {
40:         if(beg<=end)
41:         {
42:                 int mid=(beg+end)/2;
43:                 if(arr[mid]==val)
```

```
44:                         return mid;
45:                 if(arr[mid]>val)
46:                         return binaryrecur(n,arr,beg,mid-1,val);
47:                 else
48:                         return binaryrecur(n,arr,mid+1,end,val);
49:         }
50:         return -1;
51: }

52: void binarynonrecur(int n,int arr[n],int beg,int end,int val)  //Non-Recursion
53: {
54:         int mid,pos=-1;
55:         while(beg<=end)
56:         {
57:           mid=(beg+end)/2;
58:           if(arr[mid]==val)
59:           {
60:            pos=mid+1;
61:            printf("\nelement %d is found at position %d using Non-Recursion",val,pos);
62:           break;
63:           }
64:          else if(arr[mid]>val)
65:                 end=mid-1;
66:          else
67:                 beg=mid+1;
68:         }
69:         if(pos==-1)
70:           printf("\nelement %d does not found in the array using Non-Recursion",val);
71: }
```

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:

```
1: #include<stdio.h>
2: int main()
3: {
4:  int i,n,temp,j;
5:  printf("enter the number of values: ");
6:  scanf("%d",&n);
7:  int arr[n];
8:  printf("enter the array elements: ");
9:  for(i=0;i<n;i++)
10:                 scanf("%d",&arr[i]);
11:         for(i=0 ; i < n-1 ; i++)
12:         {
13:                 for(j=0 ; j < n-1-i ; j++)
14:                 {
15:                   if(arr[j] > arr[j+1])
16:                     {
17:                         temp=arr[j];
18:                         arr[j]=arr[j+1];
19:                         arr[j+1]=temp;
20:                     }
21:                 }
22:         }
23:         printf("\nAfter sorting the elements of an array is : ");
24:         for(i=0; i<n ;i++)
25:                 printf("%d ",arr[i]);
26: }
```

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
```


Reg. No:

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:

```
1: // c program to perform quick sort
2: #include <stdio.h>
3:
4: void quicksort (int [], int, int);
5:
6: void main()
7: {
8:  int arr[20];
9:  int n, i;
10:
11:         printf("Enter the number of elements: ");
12:                 scanf("%d", &n);
13:         printf("Enter the elements to be sorted: ");
14:         for (i = 0; i < n; i++)
15:                 scanf("%d", &arr[i]);
16:         quicksort(arr, 0, n - 1);
17:         printf("After applying quick sort: ");
18:         for (i = 0; i < n; i++)
19:                 printf("%d ", arr[i]);
20: }
21:
22: void quicksort(int arr[], int low, int high)
23: {
24:         int pivot, i, j, temp;
25:         if (low < high)
26:         {
27:                 pivot = low;
28:                 i = low;
29:                 j = high;
30:                 while (i < j)
31:                 {
32:                         while (arr[i] <= arr[pivot] && i <= high)
33:                         {
34:                                 i++;
35:                         }
36:                         while (arr[j] > arr[pivot] && j >= low)
37:                         {
38:                                 j--;
```

```
39:                     }
40:                     if (i < j)
41:                     {
42:                             temp = arr[i];
43:                             arr[i] = arr[j];
44:                             arr[j] = temp;
45:                     }
46:             }
47:             temp = arr[j];
48:             arr[j] = arr[pivot];
49:             arr[pivot] = temp;
50:             quicksort(arr, low, j - 1);
51:             quicksort(arr, j + 1, high);
52:     }
53: }
```

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:

```
1: #include<stdio.h>
2: void insertion_sort(int arr[*],int n);
3: void main()
4: {
5:  int i,n;
6:  printf("enter the size of an array: ");
7:  scanf("%d",&n);
8:  int arr[n];
9:  printf("enter the %d elements of an array: ",n);
10:         for(i=0 ; i<n ; i++)
11:                 scanf("%d",&arr[i]);
12:         insertion_sort(arr,n);
13: }
14:
15: void insertion_sort(int arr[], int n)
16: {
17:         int i,j,temp;
18:         for(i=1 ; i<n ; i++)
19:         {
20:                 temp=arr[i];
21:                 j=i-1;
22:                 while((temp < arr[j]) && j>=0)
23:                 {
24:                   arr[j+1]=arr[j];
25:                   j--;
26:                 }
27:                 arr[j+1]=temp;
28:         }
29:         printf("\nAfter sorting the elements of an array is : ");
30:         for(i=0 ; i<n ; i++)
31:                 printf("%d ",arr[i]);
32: }
```

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
```


Reg. No:

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:

```
1: // c program to implement radix sort
2:
3: #include <stdio.h>
4: #define size 10
5:
6: int largest(int arr[], int n);
7: void radix_sort(int arr[], int n);
8:
9: void main()
10: {
11:         int arr[size], i, n;
12:         printf("\n Enter the number of elements in the array: ");
13:                 scanf("%d", &n);
14:         printf("\n Enter the elements of the array: ");
15:         for(i=0;i<n;i++)
16:                 scanf("%d", &arr[i]);
17:         radix_sort(arr, n);
18:         printf("\n The sorted array is: ");
19:         for(i=0;i<n;i++)
20:                 printf(" %d\t", arr[i]);
21: }
22:
23: int largest(int arr[], int n)
24: {
25:         int large=arr[0], i;
26:         for(i=1;i<n;i++)
27:         {
28:                 if(arr[i]>large)
29:                 large = arr[i];
30:         }
31:         return large;
32: }
33:
34: void radix_sort(int arr[], int n)
35: {
36:         int bucket[size][size], bucket_count[size];
37:         int i, j, k, remainder, NOP=0, divisor=1, large, pass;
38:         large = largest(arr, n);
```

```
39:         while(large>0)
40:         {
41:                 NOP++;
42:                 large/=size;
43:         }
44:         for(pass=0;pass<NOP;pass++) // Initialize the buckets
45:         {
46:                 for(i=0;i<size;i++)
47:                         bucket_count[i]=0;
48:                 for(i=0;i<n;i++)
49:                 {
50:                         // sort the numbers according to the digit at passth place
51:                         remainder = (arr[i]/divisor)%size;
52:                         bucket[remainder][bucket_count[remainder]] = arr[i];
53:                         bucket_count[remainder] += 1;
54:                 }
55:                 // collect the numbers after PASS pass
56:                 i=0;
57:                 for(k=0;k<size;k++)
58:                 {
59:                         for(j=0;j<bucket_count[k];j++)
60:                         {
61:                                 arr[i] = bucket[k][j];
62:                                 i++;
63:                         }
64:                 }
65:                 divisor *= size;
66:         }
67: }
```

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:

```
1: //c program to implement merge sort
2: #include <stdio.h>
3:
4: void mergeSort(int [], int, int, int);
5: void partition(int [],int, int);
6:
7: void main()
8: {
9:  int arr[20];
10:         int i, n;
11:         printf("Enter total number of elements:");
12:                 scanf("%d", &n);
13:         printf("Enter the elements:");
14:         for(i = 0; i < n; i++)
15:                 scanf("%d", &arr[i]);
16:         partition(arr, 0, n - 1);
17:         printf("After merge sort: ");
18:         for(i = 0;i < n; i++)
19:                 printf("%d ",arr[i]);
20: }
21:
22: void partition(int arr[],int low,int high)
23: {
24:         int mid;
25:         if(low < high)
26:         {
27:                 mid = (low + high) / 2;
28:                 partition(arr, low, mid);
29:                 partition(arr, mid + 1, high);
30:                 mergeSort(arr, low, mid, high);
31:         }
32: }
33:
34: void mergeSort(int arr[],int low,int mid,int high)
35: {
36:         int i, mi, k, lo, temp[50];
37:         lo = low;
38:         i = low;
```

```
39:         mi = mid + 1;
40:         while ((lo <= mid) && (mi <= high))
41:         {
42:                 if (arr[lo] <= arr[mi])
43:                 {
44:                         temp[i] = arr[lo];
45:                         lo++;
46:                 }
47:                 else
48:                 {
49:                         temp[i] = arr[mi];
50:                         mi++;
51:                 }
52:                 i++;
53:         }
54:         if (lo > mid)
55:                 for (k = mi; k <= high; k++)
56:                 {
57:                         temp[i] = arr[k];
58:                         i++;
59:                 }
60:         else
61:                 for (k = lo; k <= mid; k++)
62:                 {
63:                         temp[i] = arr[k];
64:                         i++;
65:                 }
66:
67:         for (k = low; k <= high; k++)
68:                 arr[k] = temp[k];
69: }
```

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:

```
1: /* a) Write a C program that uses functions to create a singly linked list */
2: #include<stdio.h>
3: #include<stdlib.h>
4:
5: struct node
6: {
7:  int data;
8:  struct node* next;
9: };
10:
11: struct node* start = NULL;
12: struct node* createll(struct node *);
13: void display(struct node *);
14:
15: void main()
16: {
17:         int option,val;
18:         do
19:         {
20:         printf("\n\n *****MAIN MENU *****");
21:         printf("\n 1: Create a list");
22:         printf("\n 2: Display the list");
23:         printf("\n 3: Exit");
24:         printf("\nenter your option: ");
25:         scanf("%d",&option);
26:         switch(option)
27:           {
28:         case 1:
29:                 start = createll(start);
30:                 printf("\n linked list is created");
31:                 break;
32:         case 2:
33:                 display (start);
34:                 break;
35:           }
36:         }while(option != 3);
37: }
38:
39: struct node* createll(struct node* start)
40: {
41:         struct node *newnode;
42:         struct node *ptr;
43:         int num;
44:         printf("\n enter the data or -1 to end: ");
45:         scanf("%d",&num);
```

```
46:         while(num!= -1)
47:         {
48:                 newnode = (struct node *)malloc(sizeof(struct node));
49:                 newnode->data=num;
50:                 if(start == NULL)
51:                 {
52:                         newnode->next=NULL;
53:                         start=newnode;
54:                 }
55:                 else
56:                 {
57:                         ptr = start;
58:                         while(ptr->next != NULL)
59:                         {
60:                                 ptr = ptr->next;
61:                         }
62:                         ptr->next = newnode;
63:                         newnode->next=NULL;
64:                 }
65:                 printf("\n enter the data or -1 to end: ");
66:                 scanf("%d",&num);
67:         }
68:         return start;
69: }
70:
71: void display(struct node* start)
72: {
73:         struct node * ptr;
74:         ptr=start;
75:         while (ptr != NULL)
76:         {
77:                 printf("%d\t ", ptr->data);
78:                 ptr=ptr->next;
79:         }
80: }
```

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);
```

```
49:                     break;
50:                 }
51:         }while(option != 6);
52: }

53: struct node* createll(struct node* start)
54: {
55:         struct node *newnode;
56:         struct node *ptr;
57:         int num;
58:         printf("\n enter the data or -1 to end: ");
59:         scanf("%d",&num);
60:         while(num!= -1)
61:         {
62:                 newnode = (struct node *)malloc(sizeof(struct node));
63:                 newnode->data=num;
64:                 if(start == NULL)
65:                 {
66:                         newnode->next=NULL;
67:                         start=newnode;
68:                 }
69:                 else
70:                 {
71:                         ptr = start;
72:                         while(ptr->next != NULL)
73:                         {
74:                                 ptr = ptr->next;
75:                         }
76:                         ptr->next = newnode;
77:                         newnode->next=NULL;
78:                 }
79:                 printf("\n enter the data or -1 to end: ");
80:                 scanf("%d",&num);
81:         }
82:         return start;
83: }

84: void display(struct node* start)
85: {
86:         struct node * ptr;
87:         ptr=start;
88:         while (ptr != NULL)
89:         {
90:                 printf("%d\t ", ptr->data);
91:                 ptr=ptr->next;
92:         }
93: }

94: struct node * insert_beg(struct node * start)
95: {
96:         struct node * newnode;
97:         int num;
```

```
98:         printf("\n enter the data: ");
99:         scanf("%d",&num);
100:        newnode=(struct node *)malloc(sizeof(struct node));
101:        newnode->data=num;
102:        newnode->next=start;
103:        start=newnode;
104:        return start;
105: }

106: struct node* insert_end(struct node * start)
107: {
108:        int num;
109:        printf("\n enter the data: ");
110:        scanf("%d",&num);
111:        struct node* new_node, *ptr;
112:        new_node= (struct node*)malloc(sizeof(struct node));
113:        new_node->data=num;
114:        new_node-> next= NULL;
115:        ptr=start;
116:        while(ptr->next!=NULL)
117:                ptr= ptr->next;
118:        ptr->next=new_node;
119:        return start;
120: }

121: struct node* insert_after(struct node * start)
122: {
123:        int num, val;
124:        printf("\n enter the data: ");
125:        scanf("%d",&num);
126:        printf("\n enter a value after which the new node is inserted: ");
127:        scanf("%d",&val);
128:        struct node* new_node, *ptr, *preptr;
129:        new_node= (struct node*)malloc(sizeof(struct node));
130:        new_node->data=num;
131:        ptr=start;
132:        preptr=ptr;
133:        while(preptr->data!=val)
134:        {
135:                preptr=ptr;
136:                ptr=ptr->next;
137:        }
138:        preptr->next=new_node;
139:        new_node->next=ptr;
140:        return start;
141: }
```

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:

```
1: /* c) Use functions to perform deletion operation on a singly linked list*/
2: #include<stdio.h>
3: #include<stdlib.h>
4:
5: struct node
6: {
7:  int data;
8:  struct node* next;
9: };
10:
11: struct node* start = NULL;
12: struct node* createll(struct node *);
13: void display(struct node *);
14: struct node* delete_beg(struct node*);
15: struct node* delete_end(struct node*);
16: struct node* delete_after(struct node*);
17:
18: void main()
19: {
20:         int option;
21:         do
22:         {
23:         printf("\n\n *****MAIN MENU *****");
24:         printf("\n 1: Create a list");
25:         printf("\n 2: Display the list");
26:         printf("\n 3: Delete a node at beginning of the list");
27:         printf("\n 4: Delete a node at end of the list");
28:         printf("\n 5: Delete a node after a particular node in list");
29:         printf("\n 6: Exit");
30:         printf("\nenter your option: ");
31:         scanf("%d",&option);
32:         switch(option)
33:           {
34:                 case 1:
35:                         start = createll(start);
36:                         printf("\n linked list is created");
37:                         break;
38:                 case 2:
39:                         display (start);
40:                         break;
41:                 case 3:
42:                         start = delete_beg(start);
43:                         printf("\n first node successfully deleted");
44:                         break;
45:                 case 4:
46:                         start = delete_end(start);
47:                         printf("\n Last node successfully deleted");
48:                         break;
```

```
49:                     case 5:
50:                             start = delete_after(start);
51:                             printf("\n Node successfully deleted");
52:                             break;
53:             }
54:         }while(option != 6);
55: }
56:
57: struct node* createll(struct node* start)
58: {
59:         struct node *newnode;
60:         struct node *ptr;
61:         int num;
62:         printf("\n enter the data or -1 to end: ");
63:         scanf("%d",&num);
64:         while(num!= -1)
65:         {
66:                 newnode = (struct node *)malloc(sizeof(struct node));
67:                 newnode->data=num;
68:                 if(start == NULL)
69:                 {
70:                         newnode->next=NULL;
71:                         start=newnode;
72:                 }
73:                 else
74:                 {
75:                         ptr = start;
76:                         while(ptr->next != NULL)
77:                         {
78:                                 ptr = ptr->next;
79:                         }
80:                         ptr->next = newnode;
81:                         newnode->next=NULL;
82:                 }
83:                 printf("\n enter the data or -1 to end: ");
84:                 scanf("%d",&num);
85:         }
86:         return start;
87: }
88:
89: void display(struct node* start)
90: {
91:         struct node * ptr;
92:         ptr=start;
93:         while (ptr != NULL)
94:         {
95:                 printf("%d\t ", ptr->data);
96:                 ptr=ptr->next;
97:         }
98: }
99:
100: struct node * delete_beg(struct node * start)
```

```
101: {
102:        struct node * ptr;
103:        ptr=start;
104:        start=start->next;
105:        free(ptr);
106:        return start;
107: }
108:
109: struct node *delete_end(struct node *start)
110: {
111:        struct node *ptr, *preptr;
112:        ptr = start;
113:        while(ptr -> next != NULL)
114:        {
115:                preptr = ptr;
116:                ptr = ptr -> next;
117:        }
118:        preptr -> next = NULL;
119:        free(ptr);
120:        return start;
121: }
122:
123: struct node *delete_after(struct node *start)
124: {
125:        struct node *ptr, *preptr;
126:        int val;
127:        printf("\n Enter the value after which the node has to deleted : ");
128:        scanf("%d", &val);
129:        ptr = start;
130:        preptr = start;
131:        if(preptr->data == val) / to delete a node after a first
132:                ptr=ptr->next;
133:        while(preptr -> data != val)
134:        {
135:                preptr = ptr;
136:                ptr = ptr -> next;
137:        }
138:
139:        preptr -> next=ptr -> next;
140:        free(ptr);
141:        return start;
142: }
```

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)
{
```

```
47:         struct node *newnode;
48:         struct node *ptr;
49:         int num;
50:         printf("\n enter the data or -1 to end: ");
51:         scanf("%d",&num);
52:         while(num!= -1)
53:         {
54:                 newnode = (struct node *)malloc(sizeof(struct node));
55:                 newnode->data=num;
56:                 if(start == NULL)
57:                 {
58:                         newnode->next=NULL;
59:                         start=newnode;
60:                 }
61:                 else
62:                 {
63:                         ptr = start;
64:                         while(ptr->next != NULL)
65:                         {
66:                                 ptr = ptr->next;
67:                         }
68:                         ptr->next = newnode;
69:                         newnode->next=NULL;
70:                 }
71:                 printf("\n enter the data or -1 to end: ");
72:                 scanf("%d",&num);
73:         }
74:         return start;
75: }
76:
77: void display(struct node* start)
78: {
79:         struct node * ptr;
80:         ptr=start;
81:         while (ptr != NULL)
82:         {
83:                 printf("%d\t ", ptr->data);
84:                 ptr=ptr->next;
85:         }
86: }
87:
88: struct node * reverse(struct node * start)
89: {
90:         struct node * ptr, *preptr=NULL, *postptr=NULL;
91:         ptr=start;
92:         while(ptr!=NULL)
93:         {
94:                 postptr=ptr->next;
95:                 ptr->next= preptr;
96:                 preptr=ptr;
97:                 ptr=postptr;
98:         }
```

```
99:         start=preptr;
100:        return start;
101: }
```

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:

```
1: /* C program that implement Queue (its operations) using arrays
2: #include<stdio.h>
3: #include<stdbool.h>
4:
5: int front=-1, rear=-1;
6:
7: void enqueue(int n, int arr[*]);
8: int dequeue(int n, int arr[*]);
9: int peek(int n, int arr[*]);
10: bool isEmpty(int n, int arr[*]);
11: bool isFull(int n, int arr[*]);
12: void display(int n, int arr[*]);
13:
14: void main()
15: {
16:         int option,val,MAX;
17:         printf("enter the size of array: ");
18:         scanf("%d",&MAX);
19:         int queue[MAX];
20:         bool x;
21:         do
22:         {
23:         printf("\n\n **** MAIN MENU ****");
24:         printf("\n 1. Insert an element into queue");
25:         printf("\n 2. Delete an element from queue");
26:         printf("\n 3. Peek value of queue");
27:         printf("\n 4. Is queue empty");
28:         printf("\n 5. Is queue full");
29:         printf("\n 6. Display the queue");
30:         printf("\n 7. EXIT");
31:         printf("\n Enter your option: ");
32:         scanf("%d",&option);
33:         switch(option)
34:         {
35:         case 1:
36:                 enqueue(MAX, queue);
37:                 break;
38:         case 2:
39:                 val=dequeue(MAX, queue);
40:                 if(val != -1)
41:                         printf("\n the number deleted is : %d",val);
42:                 break;
43:         case 3:
```


Reg. No:

```
44:                     val=peek(MAX, queue);
45:                     if(val != -1)
46:                             printf("\n the first value in queue is : %d",queue[val]);
47:                     break;
48:             case 4:
49:                     x=isEmpty(MAX, queue);
50:                     if(x==true)
51:                             printf("\n the queue is empty");
52:                     else
53:                             printf("\n the queue is not empty");
54:                     break;
55:             case 5:
56:                     x=isFull(MAX, queue);
57:                     if(x==true)
58:                             printf("\n the queue is Full");
59:                     else
60:                             printf("\n the queue is not Full");
61:                     break;
62:             case 6:
63:                     display(MAX, queue);
64:                     break;
65:             default:
66:                     printf("enter correct option!");
67:               }
68:             }while(option != 7);
69: }
70:
71: //this function is used to insert an element into queue.
72: void enqueue(int MAX, int queue[MAX])
73: {
74:             int num;
75:             printf("\n enter the number to be inserted in the queue: ");
76:             scanf("%d",&num);
77:             if(rear== MAX-1)
78:                     printf("\n Overflow");
79:             else if(front == -1 && rear == -1)
80:                     front = rear = 0;
81:             else
82:                     rear++;
83:             queue[rear]=num;
84: }
85:
86: // this function is used to delete an element from a queue.
87: int dequeue(int MAX, int queue[MAX])
88: {
89:             int val;
90:             if(front == -1 || front > rear)
91:             {
92:                     printf("\n underflow");
93:                     return -1;
94:             }
95:             else
```

```
96:         {
97:                 val = queue[front];
98:                 front++;
99:                 if(front > rear)
100:                        front = rear = -1;
101:                return val;
102:        }
103: }
104:
105: //this function is used to print the front element of the queue
106: int peek(int MAX, int queue[MAX])
107: {
108:        if(front == -1 || front > rear)
109:        {
110:                printf("\n queue is empty");
111:                return -1;
112:        }
113:        else
114:                return front;
115: }
116:
117: //this function is used to check the queue is empty or not.
118: bool isEmpty(int MAX, int queue[MAX])
119: {
120:        if(front==-1 || front > rear)
121:                return true;
122:        else
123:                return false;
124: }
125:
126: // this function is used to check the queue is full or not.
127: bool isFull(int MAX, int queue[MAX])
128: {
129:        if(rear == MAX-1)
130:                return true;
131:        else
132:                return false;
133: }
134:
135: // this function is used to display the element of the queue
136: void display(int MAX, int queue[MAX])
137: {
138:        int i;
139:        if(front == -1 || front >rear)
140:                printf("\n queue is empty");
141:        else
142:        {
143:        for(i=front; i<= rear; i++)
144:                printf("\n%d \t",queue[i]);
145:        }
146: }
```

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:
```

```
47:                     peek(front);
48:                     break;
49:             case 4:
50:                     x=isEmpty(front);
51:                     if(x==true)
52:                             printf("\n the queue is empty");
53:                     else
54:                             printf("\n the queue is not empty");
55:                     break;
56:             case 5:
57:                     front = display(front);
58:                     break;
59:             default:
60:                     printf("enter correct option!");
61:                 }
62:             }while(option !=6);
63: }
64:
65: //this function is used to insert an element in to a queue
66: struct node *insert(struct node * front,int val)
67: {
68:             struct node *ptr;
69:             ptr = (struct node*)malloc(sizeof(struct node));
70:             ptr -> data = val;
71:             if( front == NULL)
72:             {
73:                     front = ptr;
74:                     rear = ptr;
75:                     front -> next = rear -> next = NULL;
76:             }
77:             else
78:             {
79:                     rear -> next = ptr;
80:                     rear = ptr;
81:                     rear -> next = NULL;
82:             }
83:             return front;
84: }
85:
86: //this function is used to delete an element an element from a queue.
87: struct node *delete_element(struct node *front)
88: {
89:             struct node *ptr;
90:             ptr = front;
91:             if(front == NULL)
92:                     printf("\n UNDERFLOW");
93:             else
94:             {
95:                     front = front -> next;
96:                     printf("\n The value being deleted is : %d", ptr -> data);
97:                     free(ptr);
98:             }
```

```
99:         return front;
100: }
101:
102: //this function displays front value.
103: void peek(struct node *front)
104: {
105:         if(front==NULL)
106:                 printf("\n QUEUE IS EMPTY");
107:         else
108:                 printf("\nPeek = %d",front->data);
109: }
110:
111: //this function checks the queue is empty or not.
112: bool isEmpty(struct node * front)
113: {
114:         if(front==NULL)
115:                 return true;
116:         else
117:                 return false;
118: }
119:
120: // this function is used to display the elements of queue
121: struct node *display(struct node *front)
122: {
123:         struct node *ptr;
124:         ptr = front;
125:         if(ptr == NULL)
126:                 printf("\n QUEUE IS EMPTY");
127:         else
128:         {
129:                 printf("\n");
130:                 while(ptr!=rear)
131:                 {
132:                         printf("%d\t", ptr -> data);
133:                         ptr = ptr -> next;
134:                 }
135:                 printf("%d\t", ptr -> data);
136:         }
137:         return front;
138: }
```

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:

```
1: // C program that implement stack (its operations) using arrays
2: #include <stdio.h>
3: int top=-1;
4:
5: void push(int MAX, int stack[*]);
6: int pop(int MAX, int stack[*]);
7: void peek(int MAX, int stack[*]);
8: void display(int MAX, int stack[*]);
9: void isEmpty(int MAX, int stack[*]);
10: void isFull(int MAX, int stack[*]);
11:
12: void main( )
13: {
14:         int val, option,MAX;
15:         printf("enter the size of stack: ");
16:         scanf("%d",&MAX);
17:         int stack[MAX];
18:         do
19:         {
20:         printf("\n *****MAIN MENU*****");
21:         printf("\n 1. PUSH");
22:         printf("\n 2. POP");
23:         printf("\n 3. PEEK");
24:         printf("\n 4. DISPLAY");
25:         printf("\n 5. To check the stack is Empty");
26:         printf("\n 6. To check the stack is Full");
27:         printf("\n 7. EXIT");
28:         printf("\n Enter your option: ");
29:         scanf("%d", &option);
30:         switch(option)
31:         {
32:         case 1:
33:                 push(MAX, stack);
34:                 break;
35:         case 2:
36:                 val = pop(MAX, stack);
37:                 if(val != -1)
38:                         printf("\n The value deleted from stack is: %d", val);
39:                         break;
40:         case 3:
41:                 peek(MAX, stack);
42:                 break;
43:         case 4:
44:                 display(MAX, stack);
```

```
45:                     break;
46:             case 5:
47:                     isEmpty(MAX, stack);
48:                     break;
49:             case 6:
50:                     isFull(MAX, stack);
51:                     break;
52:             default:
53:                     printf("enter correct option!!!");
54:             }
55:     }while(option != 7);
56: }
57:
58:
59: void push(int MAX, int stack[MAX])
60: {
61:             int val;
62:             printf("\n Enter the number to be pushed on stack: ");
63:             scanf("%d", &val);
64:             if(top == MAX-1)
65:                     printf("\n STACK OVERFLOW");
66:             else
67:             {
68:                     top++;
69:                     stack[top] = val;
70:             }
71: }
72:
73: int pop(int MAX, int stack[MAX])
74: {
75:             int val;
76:             if(top == -1)
77:             {
78:                     printf("\n STACK UNDERFLOW");
79:                     return -1;
80:             }
81:             else
82:             {
83:                     val = stack[top];
84:                     top--;
85:                     return val;
86:             }
87: }
88:
89: void peek(int MAX, int stack[MAX])
90: {
91:             if(top == -1)
92:             {
93:                     printf("\n STACK IS EMPTY");
94:             }
95:             else
96:                     printf("\n The value stored at top of stack is: %d", stack[top
```

```
97: }
98:
99: void display(int MAX, int stack[MAX])
100: {
101:        int i;
102:        if(top == -1)
103:                printf("\n STACK IS EMPTY");
104:        else
105:        {
106:                for(i=0;i<=top;i++)
107:                        printf("\t %d",stack[i]);
108:        }
109: }
110:
111: void isEmpty(int MAX, int stack[MAX])
112: {
113:        if(top == -1)
114:                printf("Stack is empty");
115:        else
116:                printf("Stack is not empty");
117: }
118:
119: void isFull(int MAX, int stack[MAX])
120: {
121:        if(top == MAX-1)
122:                printf("\n the stack is Full");
123:        else
124:                printf("\n the stack is not Full");
125: }
```

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:

```
1: //C program that implement stack (its operations) using LinkedList
2: #include <stdio.h>
3: #include <stdlib.h>
4:
5: struct node
6: {
7:  int data;
8:  struct node *next;
9: };
10:
11: struct node *top = NULL;
12:
13: struct node *push(struct node *, int);
14: struct node *display(struct node *);
15: struct node *pop(struct node *);
16: int peek(struct node *);
17:
18: void main()
19: {
20:         int val, option;
21:         do
22:         {
23:         printf("\n *****MAIN MENU*****");
24:         printf("\n 1. PUSH");
25:         printf("\n 2. POP");
26:         printf("\n 3. PEEK");
27:         printf("\n 4. DISPLAY");
28:         printf("\n 5. EXIT");
29:         printf("\n Enter your option: ");
30:         scanf("%d", &option);
31:         switch(option)
32:         {
33:         case 1:
34:                 printf("\n Enter the number to be pushed on stack: ");
35:                 scanf("%d", &val);
36:                 top = push(top, val);
37:                 break;
38:         case 2:
39:                 top = pop(top);
40:                 break;
41:         case 3:
42:                 val = peek(top);
43:                 if (val != -1)
44:                         printf("\n The value at the top of stack is: %d", val);
45:                 else
46:                         printf("\n STACK IS EMPTY");
47:                 break;
48:         case 4:
```

```
49:                     top = display(top);
50:                     break;
51:               }
52:         }while(option != 5);
53: }
54:
55:
56: struct node *push(struct node *top, int val)
57: {
58:         struct node* ptr;
59:         ptr=(struct node*)malloc(sizeof(struct node));
60:         ptr->data=val;
61:         if(top==NULL)
62:         {
63:                 ptr->next=NULL;
64:                 top=ptr;
65:         }
66:         else
67:         {
68:                 ptr->next=top;
69:                 top=ptr;
70:         }
71:         return top;
72: }
73:
74:
75: struct node *pop(struct node *top)
76: {
77:         struct node* ptr = top;
78:         if(top==NULL)
79:                 printf("\n UNDERFLOW");
80:         else
81:         {
82:                 top=top->next;
83:                 printf("\n the value deleted is %d",ptr->data);
84:                 free(ptr);
85:         }
86:         return top;
87: }
88:
89:
90: int peek(struct node *top)
91: {
92:         if(top == NULL)
93:                 return -1;
94:         else
95:                 return top->data;
96: }
97:
98:
99: struct node *display(struct node *top)
100: {
```

```
101:        struct node* ptr=top;
102:        if(top==NULL)
103:                printf("\nUNDERFLOW");
104:        else
105:        {
106:                while(ptr!=NULL)
107:                {
108:                        printf("\t %d",ptr->data);
109:                        ptr=ptr->next;
110:                }
111:        }
112:        return top;
113: }
```

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:

```
1: // c program to evaluate postfix expression
2: #include <stdio.h>
3: #include <ctype.h>
4: #define MAX 100
5: float st[MAX];
6: int top=-1;
7:
8: void push(float st[], float val);
9: float pop(float st[]);
10: float evaluatePostfixExp(char exp[]);
11:
12: int main()
13: {
14:         float val;
15:         char exp[100];
16:         printf("\n Enter any postfix expression : ");
17:         gets(exp);
18:         val = evaluatePostfixExp(exp);
19:         printf("\n Value of the postfix expression = %.2f", val);
20:         return 0;
21: }
22:
23: float evaluatePostfixExp(char exp[])
24: {
25:         int i=0;
26:         float op1, op2, value;
27:         while(exp[i] != '\0')
28:         {
29:                 if(isdigit(exp[i]))
30:                         push(st, (float)(exp[i]-'0'));
31:                 else
32:                 {
33:                         op2 = pop(st);
34:                         op1 = pop(st);
35:                         switch(exp[i])
36:                         {
37:                         case '+':
38:                                 value = op1 + op2;
39:                                 break;
40:                         case '-':
41:                                 value = op1 - op2;
42:                                 break;
43:                         case '/':
44:                                 value = op1 / op2;
45:                                 break;
46:                         case '*':
```

```
47:                                     value = op1 * op2;
48:                                     break;
49:                             case '%':
50:                                     value = (int)op1 % (int)op2;
51:                                     break;
52:                     }
53:                     push(st, value);
54:             }
55:             i++;
56:         }
57:         return(pop(st));
58: }
59:
60: void push(float st[], float val)
61: {
62:         if(top==MAX-1)
63:                 printf("\n STACK OVERFLOW");
64:         else
65:         {
66:                 top++;
67:                 st[top]=val;
68:         }
69: }
70:
71: float pop(float st[])
72: {
73:         float val=-1;
74:         if(top==-1)
75:                 printf("\n STACK UNDERFLOW");
76:         else
77:         {
78:                 val=st[top];
79:                 top--;
80:         }
81:         return val;
82: }
```

OUTPUT:

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


Reg. No:

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:

```
1: //A recursive C program for traversing a binary tree in preorder,inorder, postorder.
2: #include<stdio.h>
3: #include<stdlib.h>
4:
5: struct node
6: {
7:  char data;
8:  struct node *left;
9:  struct node *right;
10: };
11:
12: struct node *root=NULL;
13:
14: struct node *create();
15: void inorder(struct node*);
16: void preorder(struct node*);
17: void postorder(struct node*);
18:
19: void main()
20: {
21:
22:         int option;
23:         do
24:         {
25:         printf("\n\n *****MAIN MENU*****");
26:         printf("\n 1. create a binary tree ");
27:         printf("\n 2. In-order traversal");
28:         printf("\n 3. Pre-order traversal");
29:         printf("\n 4. Post-order traversal");
30:         printf("\n 5. EXIT");
31:         printf("\n Enter your option : ");
32:         scanf("%d", &option);
33:         switch(option)
34:           {
35:         case 1:
36:                 root = create();
37:                 break;
38:         case 2:
39:                 printf("\n the elements after inorder traversal: ");
40:                 inorder(root);
41:                 break;
42:         case 3:
43:                 printf("\n the elements after preorder traversal: ");
```

```
44:                     preorder(root);
45:                     break;
46:             case 4:
47:                     printf("\n the elements after postorder traversal: ");
48:                     postorder(root);
49:                     break;
50:               }
51:             }while(option != 5);
52: }
53:
54: struct node *create()
55: {
56:             struct node *p;
57:             char x;
58:             printf("Enter data(0 for no data):");
59:             scanf(" %c",&x);
60:             if(x=='0')
61:                     return NULL;
62:             p=(struct node*)malloc(sizeof(struct node));
63:             p->data=x;
64:             printf("Enter left child of %c:\n",x);
65:             p->left=create();
66:             printf("Enter right child of %c:\n",x);
67:             p->right=create();
68:             return p;
69: }
70:
71: void inorder(struct node *root)
72: {
73:             if(root!=NULL)
74:             {
75:                     inorder(root->left);
76:                     printf(" %2c",root->data);
77:                     inorder(root->right);
78:             }
79: }
80:
81: void preorder(struct node *root)
82: {
83:             if(root!=NULL)
84:             {
85:                     printf(" %2c",root->data);
86:                     preorder(root->left);
87:                     preorder(root->right);
88:             }
89: }
90:
91: void postorder(struct node *root)
92: {
93:             if(root!=NULL)
94:             {
95:                     postorder(root->left);
```

```
96:                 postorder(root->right);
97:                 printf(" %2c",root->data);
98:         }
99: }
```

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:

```
1: // C program to create a binary search tree
2: #include<stdio.h>
3: #include<stdlib.h>
4:
5: struct node
6: {
7:  int data;
8:  struct node* left, *right;
9: };
10:
11: struct node* root = NULL;
12:
13: struct node *create(struct node* );
14: void display(struct node* );
15:
16: void main()
17: {
18:         int option, val;
19:         do
20:         {
21:         printf("\n\n *** MAIN MENU ***");
22:         printf("\n 1. create");
23:         printf("\n 2. display");
24:         printf("\n 3. exit");
25:         printf("\n enter your choice: ");
26:         scanf("%d",&option);
27:         switch(option)
28:           {
29:                 case 1:
30:                         root = create(root);
31:                         break;
32:                 case 2:
33:                         printf("\n the elements in the BST: ");
34:                         display(root);
35:                         break;
36:           }
37:         }while(option<3);
38: }
39:
40: struct node* create(struct node* root)
41: {
42:         int val;
43:         printf("\n enter data (-1 to stop): ");
```

```
44:         scanf("%d",&val);
45:         while(val != -1)
46:         {
47:                 struct node *ptr, *nodeptr, *parentptr;
48:                 ptr=(struct node*)malloc(sizeof(struct node));
49:                 ptr->data=val;
50:                 ptr->left=NULL;
51:                 ptr->right=NULL;
52:                 if(root==NULL)
53:                         root=ptr;
54:                 else
55:                 {
56:                         parentptr=NULL;
57:                         nodeptr=root;
58:                         while(nodeptr!=NULL)
59:                         {
60:                                 parentptr=nodeptr;
61:                                 if(val<nodeptr->data)
62:                                         nodeptr=nodeptr->left;
63:                                 else
64:                                         nodeptr=nodeptr->right;
65:                         }
66:                         if(val<parentptr->data)
67:                                 parentptr->left=ptr;
68:                         else
69:                                 parentptr->right=ptr;
70:                 }
71:         printf("\n enter data (-1 to stop): ");
72:         scanf("%d",&val);
73:         }
74:         return root;
75: }
76:
77: void display(struct node* root)
78: {
79:         if(root != NULL)
80:         {
81:                 display(root->left);
82:                 printf("%4d", root->data);
83:                 display(root->right);
84:         }
85: }
```

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:

```
1: // C program to insert a new node in a BST
2: #include<stdio.h>
3: #include<stdlib.h>
4:
5: struct node
6: {
7:  int data;
8:  struct node *left,*right;
9: };
10:
11: struct node *root=NULL;
12:
13: struct node *create(struct node *);
14: void display(struct node *);
15: struct node* insert(struct node*, int);
16:
17: void main()
18: {
19:         int option,val;
20:         do
21:         {
22:         printf("\n\n *** MAIN MENU *** ");
23:         printf("\n 1. create");
24:         printf("\n 2. display");
25:         printf("\n 3. insert a new node");
26:         printf("\n 4. Quit");
27:         printf("\n Enter your choice: ");
28:         scanf("%d",&option);
29:         switch (option)
30:           {
31:                 case 1:
32:                         root = create(root);
33:                         break;
34:                 case 2:
35:                         printf("\n The elements in the tree are\n");
36:                         display(root);
37:                         break;
38:                 case 3:
39:                         printf("\n Enter an element to insert in BST: ");
40:                         scanf("%d",&val);
41:                         root= insert(root, val);
42:                         break;
43:                 default:
44:                         printf("enter correct option!");
45:           }
46:         }while(option<4);
47: }
48:
```

```
49:
50: struct node* create(struct node* root)
51: {
52:         int val;
53:         printf("\n enter data (-1 to stop): ");
54:         scanf("%d",&val);
55:         while(val != -1)
56:         {
57:                 struct node *ptr, *nodeptr, *parentptr;
58:                 ptr = (struct node*)malloc(sizeof(struct node));
59:                 ptr->data = val;
60:                 ptr->left = NULL;
61:                 ptr->right = NULL;
62:                 if(root==NULL)
63:                 {
64:                         root=ptr;
65:                 }
66:                 else
67:                 {
68:                         parentptr=NULL;
69:                         nodeptr=root;
70:                         while(nodeptr!=NULL)
71:                         {
72:                                 parentptr=nodeptr;
73:                                 if(val<nodeptr->data)
74:                                         nodeptr=nodeptr->left;
75:                                 else
76:                                         nodeptr = nodeptr->right;
77:                         }
78:                         if(val<parentptr->data)
79:                                 parentptr->left = ptr;
80:                         else
81:                                 parentptr->right = ptr;
82:                 }
83:                 printf("\n enter data (-1 to stop): ");
84:                 scanf("%d",&val);
85:         }
86:         return root;
87: }
88:
89: void display(struct node *root)
90: {
91:         if(root!=NULL)
92:         {
93:                 display(root->left);
94:                 printf(" %5d",root->data);
95:                 display (root->right);
96:         }
97: }
98:
99: struct node* insert(struct node* root, int val)
100: {
```

```
101:        if(root==NULL)
102:        {
103:                root=(struct node*)malloc(sizeof(struct node));
104:                root->data=val;
105:                root->right=root->left=NULL;
106:                return root;
107:        }
108:        else if (val < root->data)
109:                root->left = insert(root->left, val);
110:        else
111:                root->right = insert(root->right, val);
112:        return root;
113: }
```

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);
```

```
47:                             root=delete_node(root, val);
48:                             break;
49:                     default:
50:                             printf("\nenter valid option!");
51:             }
52:         }while(option!=4);
53: }
54:
55: struct node* create(struct node* root)
56: {
57:         int val;
58:         printf("\n enter data (-1 to stop): ");
59:         scanf("%d",&val);
60:         while(val != -1)
61:         {
62:                 struct node *ptr, *nodeptr, *parentptr;
63:                 ptr=(struct node*)malloc(sizeof(struct node));
64:                 ptr->data=val;
65:                 ptr->left=NULL;
66:                 ptr->right=NULL;
67:                 if(root==NULL)
68:                         root=ptr;
69:                 else
70:                 {
71:                         parentptr=NULL;
72:                         nodeptr=root;
73:                         while(nodeptr!=NULL)
74:                         {
75:                                 parentptr=nodeptr;
76:                                 if(val<nodeptr->data)
77:                                         nodeptr=nodeptr->left;
78:                                 else
79:                                         nodeptr=nodeptr->right;
80:                         }
81:                         if(val<parentptr->data)
82:                                 parentptr->left=ptr;
83:                         else
84:                                 parentptr->right=ptr;
85:                 }
86:                 printf("\n enter data (-1 to stop): ");
87:                 scanf("%d",&val);
88:         }
89:         return root;
90: }
91:
92: void display(struct node* root)
93: {
94:         if(root != NULL)
95:         {
96:                 display(root->left);
97:                 printf("%4d", root->data);
98:                 display(root->right);
```

```
99:         }
100: }
101:
102: struct node* delete_node(struct node* root, int val)
103: {
104:         if(root==NULL)
105:                 return root;
106:         //If the val to be deleted is smaller than the root go to left subtree
107:         if(val<root->data)
108:                 root->left=delete_node(root->left, val);
109:         //If the val to be deleted is greater than the root go to right subtree
110:         else if(val>root->data)
111:                 root->right=delete_node(root->right, val);
112:         else
113:         {
114:         //a node with one child or null
115:                 if(root->left==NULL)
116:                 {
117:                         struct node* temp=root->right;
118:                         free(root);
119:                         return temp;
120:                 }
121:                 else if(root->right==NULL)
122:                 {
123:                         struct node* temp=root->left;
124:                         free(root);
125:                         return temp;
126:                 }
127: // a node with two child
128:                 struct node* temp=findminvalue(root->right);
129:                 root->data=temp->data;
130:                 root->right=delete_node(root->right,temp->data);
131:         }
132:         return root;
133: }

134: struct node* findminvalue(struct node* ptr)
135: {
136:         struct node* current=ptr;
137:         while(current->left!=NULL)
138:                 current = current->left;
139:         return current;
140: }
```

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
```