INDEX

S. No.	Name of Practical	Date	Remark
1.1	Write a program in C to insert element in arra		
1.2	Write a program in C to insert and delete an element from array.		
1.3	Write a program in C to reverse an array.		
1.4	Write a program in C to merge two arrays.		
2.1	Write a program in C to implement bubble so in array.		
2.2	Write a program in C to implement Merge sort in array.		
2.3	Write a program in C to implement Insertion sort in array.		
2.4	Write a program in C to implement Selection sort in array.		
3.0	Write a program in C to implement Singly Linked List.		
4.0	Write a program in C to implement Circular Linked List.		
5.0	Write a program in C to implement Doubly Linked List.		
6.0	Write a program in C to implement Stack.		
7.0	Write a program in C to implement Circular Queue.		
8.0	Write a program in C to implement De-Queue.		
9.0	Write a program in C to implement Binary Search Tree Insertion.		
10.0	Write a program in C to implement Binary Search Tree Traversal (Inorder, Preorder, Postorder).		
11.1	Write a program in C to implement Linear Search Algorithm.		
11.2	Write a program in C to implement Binary Search Algorithm.		
12.0	Write a program in C to implement Matrix insert by user. Check either matrix is sparse or not?		
13.0	Write a program in C to implement addition of two polynomials		
14.0	Write a program in C to implement to calculate grade of a student by given marks.		

PROGRAM NO. 1.1

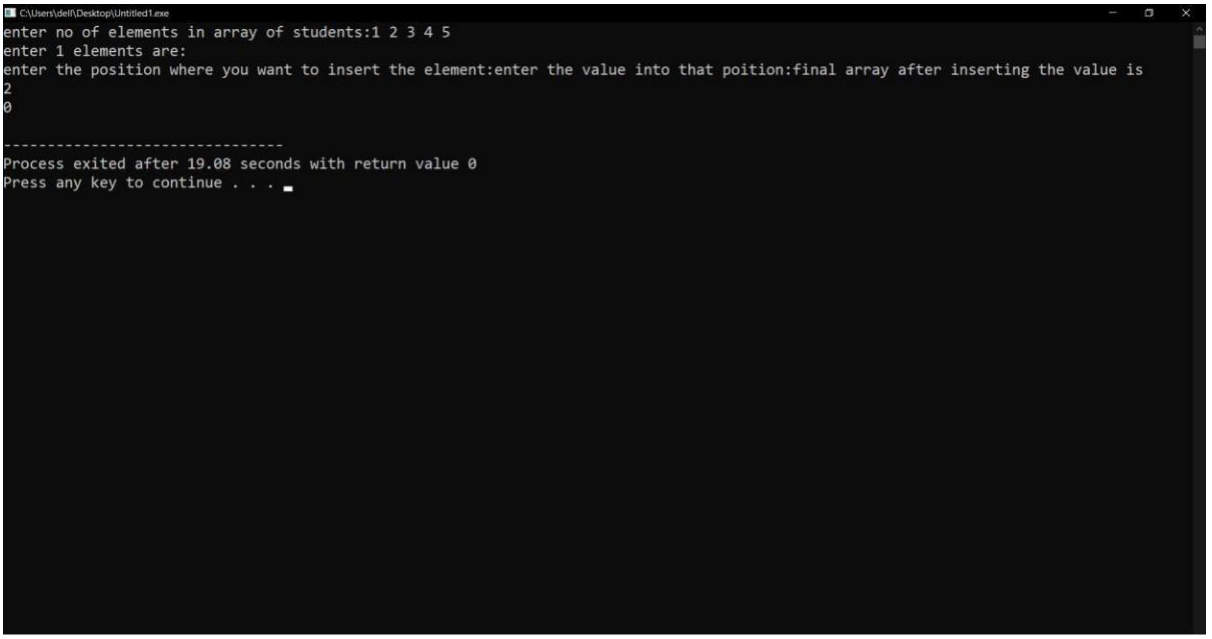
OBJECTIVE:

Write a program in C to insert element in array.

CODING:

```
#include<stdio.h>
int main(){
int student[40],pos,i,size,value;
printf("enter no of elements in array of
students:"); scanf("%d",&size);
printf("enter %d elements are:\n",size);
for(i=0;i<size;i++)
  scanf("%d",&student[i]);
printf("enter the position where you want to insert the element:");
scanf("%d",&pos);
printf("enter the value into that poition:");
scanf("%d",&value);
for(i=size-1;i>=pos-1;i--)
  student[i+1]=student[i];
student[pos-1]= value;
printf("final array after inserting the value
is\n"); for(i=0;i<=size;i++)
  printf("%d\n",student[i]);
return 0;
}
```

OUTPUT:


C:\Users\dell\Desktop\Untitled1.exe
enter no of elements in array of students:1 2 3 4 5
enter 1 elements are:
enter the position where you want to insert the element:enter the value into that poition:final array after inserting the value is
2
0

Process exited after 19.08 seconds with return value 0
Press any key to continue . . .

PROGRAM NO. 1.2

OBJECTIVE:

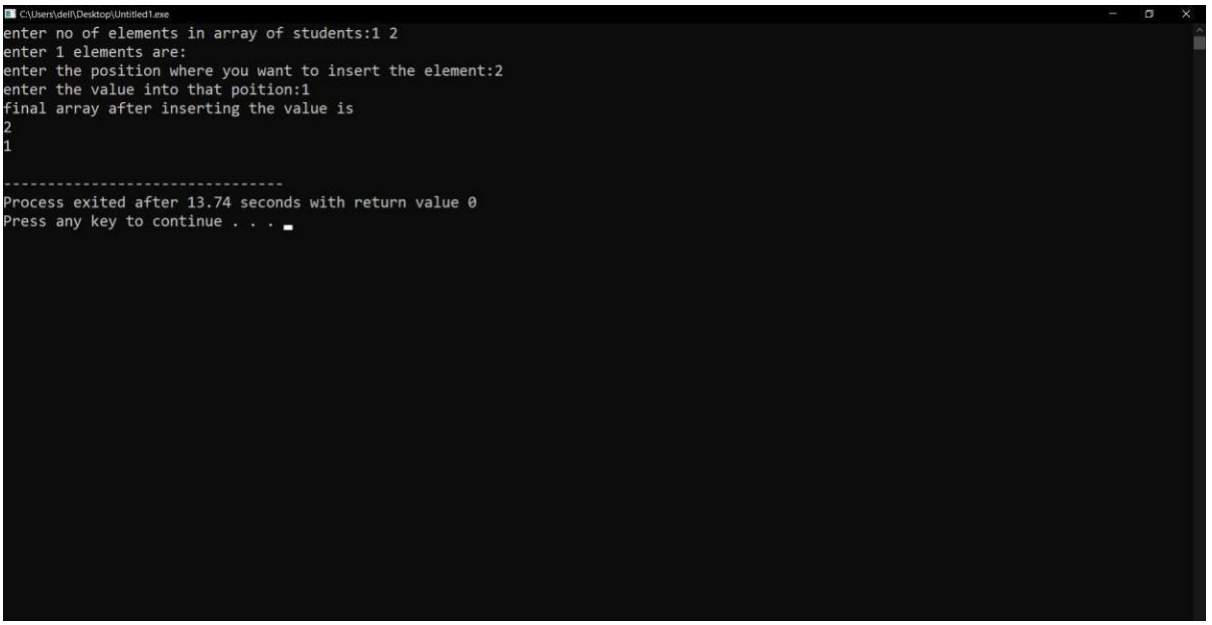
Write a program in C to insert and delete an element from array.

CODING:

```
#include <conio.h>
  int main ()
{
    // declaration of the int type
    variable int arr[50];
    int pos, i, num; // declare int type variable
    pr#include <stdio.h>
intf (" \n Enter the number of elements in an array: \n
    "); scanf (" %d", &num);

    printf (" \n Enter %d elements in array: \n ", num);

    // use for loop to insert elements one by one in
    array for (i = 0; i < num; i++ )
    { printf (" arr[%d] = ", i);
       scanf (" %d", &arr[i]);
    }
      // enter the position of the element to be deleted
    printf( " Define the position of the array element where you want to delete: \n
    "); scanf (" %d", &pos);

    // check whether the deletion is possible or
    not if (pos >= num+1)
    {
       printf (" \n Deletion is not possible in the array.");
    }
    else
    {
       for (i = pos - 1; i < num -1; i++)
       {
          arr[i] = arr[i+1]; // assign arr[i+1] to arr[i]
       }

       printf (" \n The resultant array is: \n");

       // display the final array
       for (i = 0; i< num - 1; i++)
       {
          printf (" arr[%d] = ", i);
          printf (" %d \n", arr[i]);
       }
```

```
    }
    return 0;
}
```

OUTPUT:



PROGRAM NO. 1.3

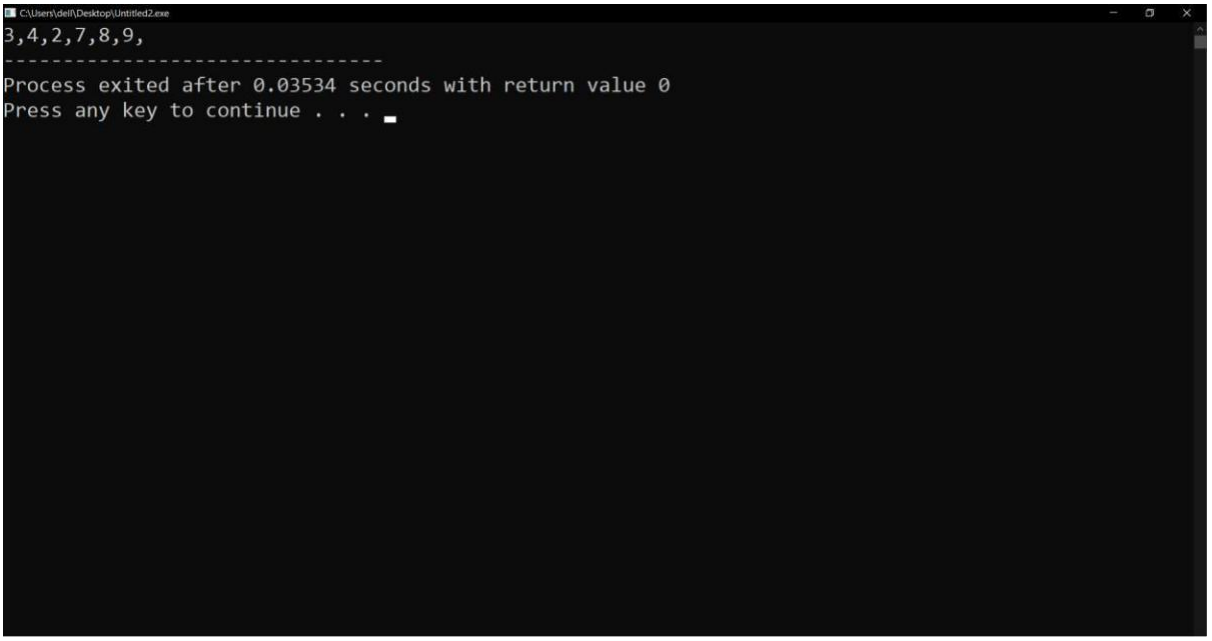
OBJECTIVE:

Write a program in C to reverse an array.

CODING:

```
#include  <stdio.h>
#include <stdlib.h>
#define n 6
int main(){
   int arr[n] = {9, 8, 7, 2, 4, 3};
   int temp;
   for(int i = 0; i<n/2; i++){
      temp = arr[i];
      arr[i] = arr[n-i-1];
      arr[n-i-1] = temp;
   }
   for(int i = 0; i < n; i++){
      printf("%d,", arr[i]);
   }
}
```

OUTPUT:


C:\Users\dell\Desktop\Untitled2.exe
3,4,2,7,8,9,

Process exited after 0.03534 seconds with return value 0
Press any key to continue . . .

PROGRAM NO. 1.4

OBJECTIVE:

Write a program in C to merge two arrays.

CODING:

```
#include<stdio.h>
#include<stdlib.h>
int main(){
  int a[10],b[10],c[20],m,n,o,i,j,k,temp;
  printf("Enter size of Array1\n");
  scanf("%d",&n);
  printf("Enter size of Array2\n");
  scanf("%d",&m);
  o=m+n; //size of third array
  printf("Enter Elements of Array1\n");
  for(i=0;i<n;i++){
    scanf("%d",&a[i]);
  }
  printf("Enter Elements of Array2\n");
  for(i=0;i<m;i++){
    scanf("%d",&b[i]);
  }
  //sorting first array
  for(i=0;i<n;i++){
    for(j=0;j<n-1-i;j++){
      if(a[j]>a[j+1]){
        temp=a[j];
        a[j]=a[j+1];
        a[j+1]=temp;
      }
    }
  }
  //sorting second array
  for(i=0;i<m;i++){
    for(j=0;j<m-1-i;j++){
      if(b[j]>b[j+1]){
        temp=b[j];
        b[j]=b[j+1];
        b[j+1]=temp;
      }
    }
  }
  printf("Elements of Array1\n");
  for(i=0;i<n;i++){
    printf("a[%d]=%d\n",i,a[i]);
  }
```

```
  printf("Elements of Array2\n");
  for(i=0;i<m;i++){
    printf("b[%d]=%d\n",i,b[i]);
  }
  j=0;
  k=0;
  for(i=0;i<o;i++){ // merging two arrays
    if(a[j]<=b[k]){
      c[i]=a[j];
      j++;
    }
    else{
      c[i]=b[k];
      k++;
    }
  }
  printf("Merged array is :\n");
  for(i=0;i<o;i++){
    printf("c[%d]=%d\n",i,c[i]);
  }
}
```

OUTPUT:


C:\Users\dell\Desktop\Untitled3.exe
4
Enter Elements of Array1
1 2 3 4 5
Enter Elements of Array2
3 4 5 6
Elements of Array1
a[0]=1
a[1]=2
a[2]=3
a[3]=4
a[4]=5
Elements of Array2
b[0]=3
b[1]=4
b[2]=5
b[3]=6
Merged array is :
c[0]=1
c[1]=2
c[2]=3
c[3]=3
c[4]=4
c[5]=4
c[6]=5
c[7]=0
c[8]=5

Process exited after 27.8 seconds with return value 0
Press any key to continue . . .

PROGRAM NO. 2.1

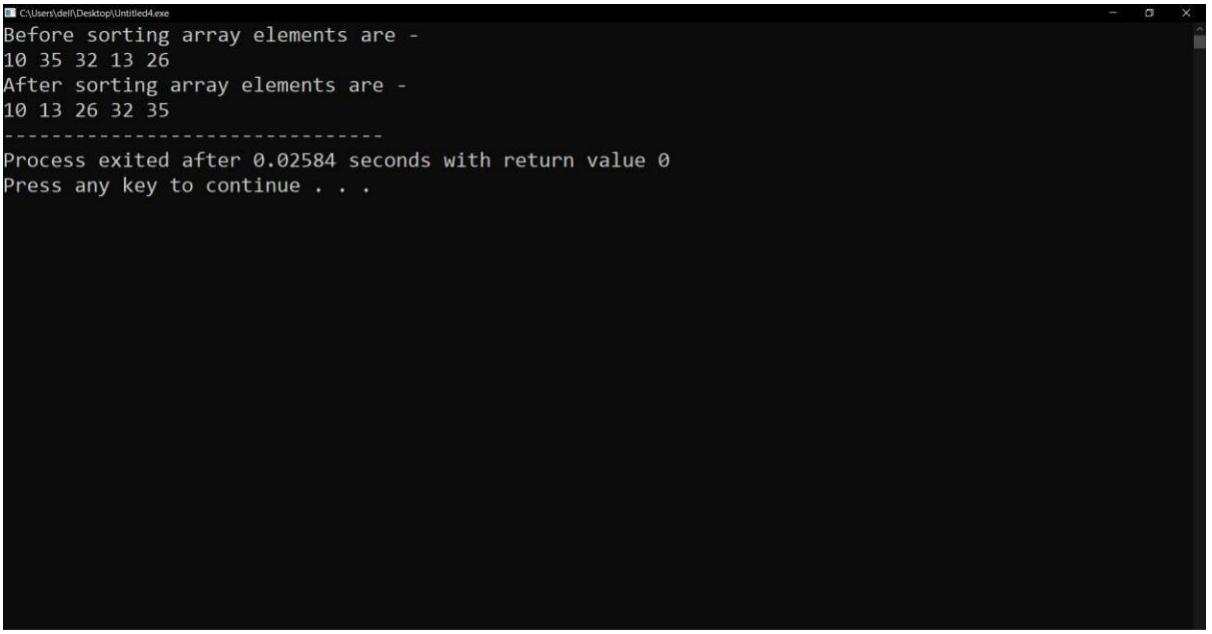
OBJECTIVE:

Write a program in C to implement bubble sort in array.

CODING:

```
#include<stdio.h>
 void print(int a[], int n) //function to print array elements
    {
    int i;
    for(i = 0; i < n; i++)
    {
        printf("%d ",a[i]);
    }
    }
 void bubble(int a[], int n) // function to implement bubble sort
 {
   int i, j, temp;
   for(i = 0; i < n; i++)
    {
      for(j = i+1; j < n; j++)
        {
           if(a[j] < a[i])
           {
              temp = a[i];
              a[i] = a[j];
              a[j] = temp;
           }
        }
    }
 }
void main ()
{
    int i, j,temp;
    int a[5] = { 10, 35, 32, 13, 26};
    int n = sizeof(a)/sizeof(a[0]);
    printf("Before sorting array elements are - \n");
    print(a, n);
    bubble(a, n);
    printf("\nAfter sorting array elements are -
    \n"); print(a, n);
}
```

OUTPUT:


C:\Users\dell\Desktop\Untitled4.exe
Before sorting array elements are -
10 35 32 13 26
After sorting array elements are -
10 13 26 32 35

Process exited after 0.02584 seconds with return value 0
Press any key to continue . . .

PROGRAM NO. 2.2

OBJECTIVE:

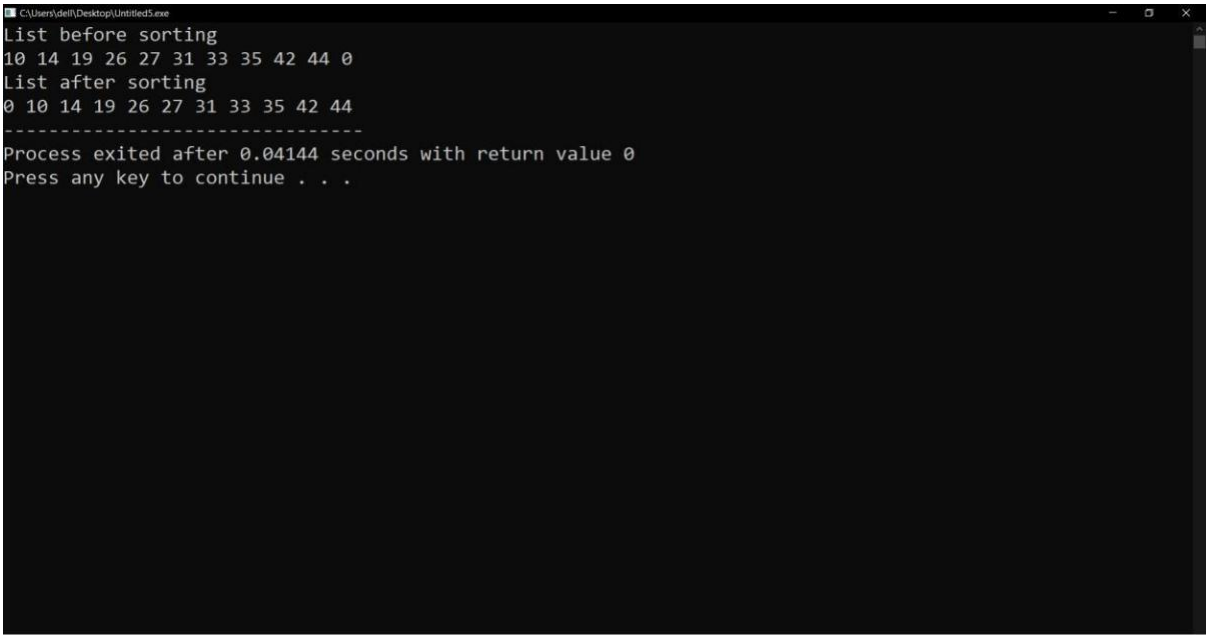
Write a program in C to implement Merge sort in array.

CODING:

```
#include <stdio.h>

#define max 10

int a[11] = { 10, 14, 19, 26, 27, 31, 33, 35, 42, 44, 0 };
int b[10];

void merging(int low, int mid, int high) {
  int l1, l2, i;

  for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high; i++)
    { if(a[l1] <= a[l2])
      b[i] = a[l1++];
    else
      b[i] = a[l2++];
  }

  while(l1 <= mid)
    b[i++] = a[l1++];

  while(l2 <= high)
    b[i++] = a[l2++];

  for(i = low; i <= high; i++)
    a[i] = b[i];
}

void sort(int low, int high) {
  int mid;

  if(low < high) {
    mid = (low + high) / 2;
    sort(low, mid);
    sort(mid+1, high);
    merging(low, mid, high);
  } else {
    return;
  }
}

int main() {
```

```
  int i;

  printf("List before sorting\n");

  for(i = 0; i <= max; i++)
    printf("%d ", a[i]);

  sort(0, max);

  printf("\nList after sorting\n");

  for(i = 0; i <= max; i++)
    printf("%d ", a[i]);
}
```

OUTPUT:


C:\Users\dell\Desktop\Untitled5.exe
List before sorting
10 14 19 26 27 31 33 35 42 44 0
List after sorting
0 10 14 19 26 27 31 33 35 42 44

Process exited after 0.04144 seconds with return value 0
Press any key to continue . . .

PROGRAM NO. 2.3

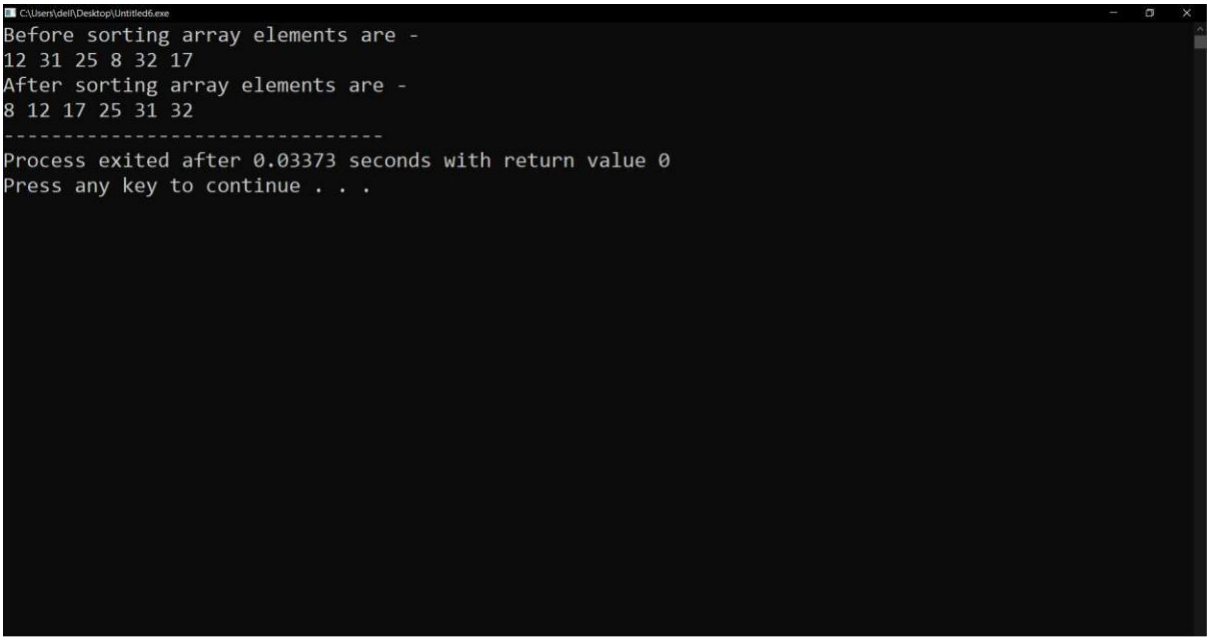
OBJECTIVE:

Write a program in C to implement Insertion sort in array.

CODING:

```
#include <stdio.h>

void insert(int a[], int n) /* function to sort an aay with insertion sort */
{
   int i, j, temp;
   for (i = 1; i < n; i++) {
      temp = a[i];
      j = i - 1;

      while(j>=0 && temp <= a[j]) /* Move the elements greater than temp to one position
ahead from their current position*/
      {
         a[j+1] = a[j];
         j = j-1;
      }
      a[j+1] = temp;
   }
}

void printArr(int a[], int n) /* function to print the array
*/ {
   int i;
   for (i = 0; i < n; i++)
      printf("%d ", a[i]);
}

int main()
{
   int a[] = { 12, 31, 25, 8, 32, 17 };
   int n = sizeof(a) / sizeof(a[0]);
   printf("Before sorting array elements are - \n");
   printArr(a, n);
   insert(a, n);
   printf("\nAfter sorting array elements are - \n");
   printArr(a, n);

   return 0;
}
```

OUTPUT:


C:\Users\dell\Desktop\Untitled6.exe
Before sorting array elements are -
12 31 25 8 32 17
After sorting array elements are -
8 12 17 25 31 32

Process exited after 0.03373 seconds with return value 0
Press any key to continue . . .

PROGRAM NO. 2.4

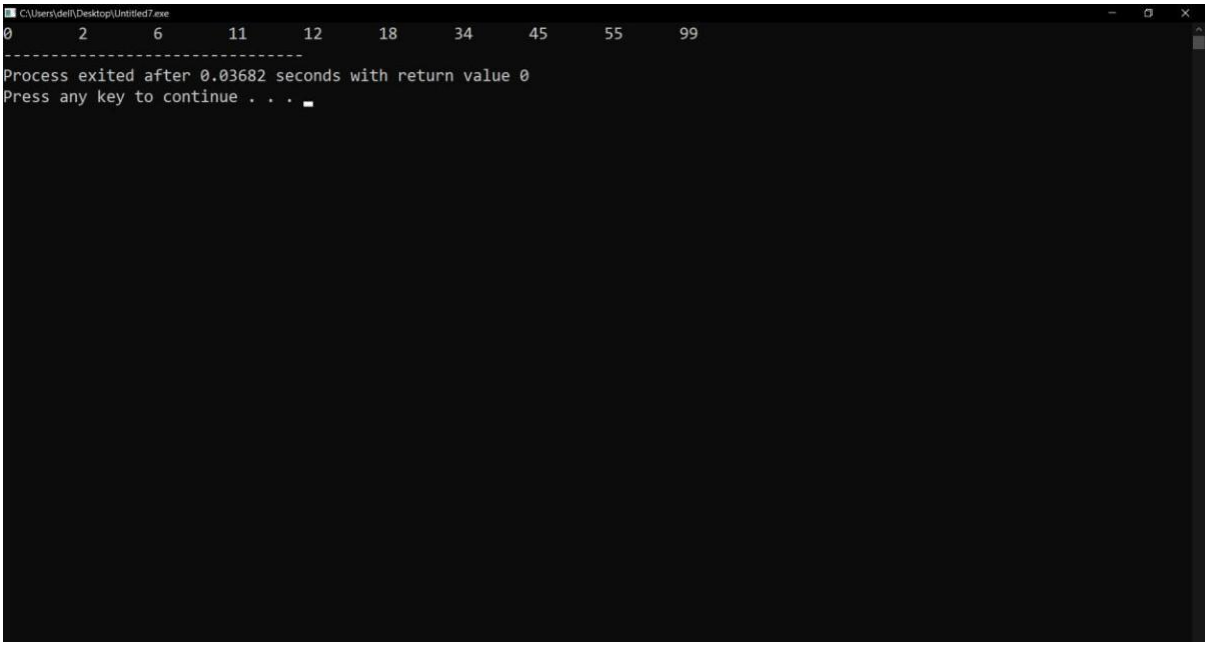
OBJECTIVE:

Write a program in C to implement Selection sort in array.

CODING:

```
#include <stdio.h>
int main() {
  int arr[10]={6,12,0,18,11,99,55,45,34,2};
  int n=10;
  int i, j, position, swap;
  for (i = 0; i < (n - 1); i++) {
    position = i;
    for (j = i + 1; j < n; j++) {
      if (arr[position] > arr[j])
        position = j;
    }
    if (position != i) {
      swap = arr[i];
      arr[i] = arr[position];
      arr[position] = swap;
    }
  }
  for (i = 0; i < n; i++)
    printf("%d\t", arr[i]);
  return 0;
}
```

OUTPUT:


C:\Users\dell\Desktop\Untitled7.exe
0 2 6 11 12 18 34 45 55 99
Process exited after 0.03682 seconds with return value 0
Press any key to continue . . .

PROGRAM NO. 3.0

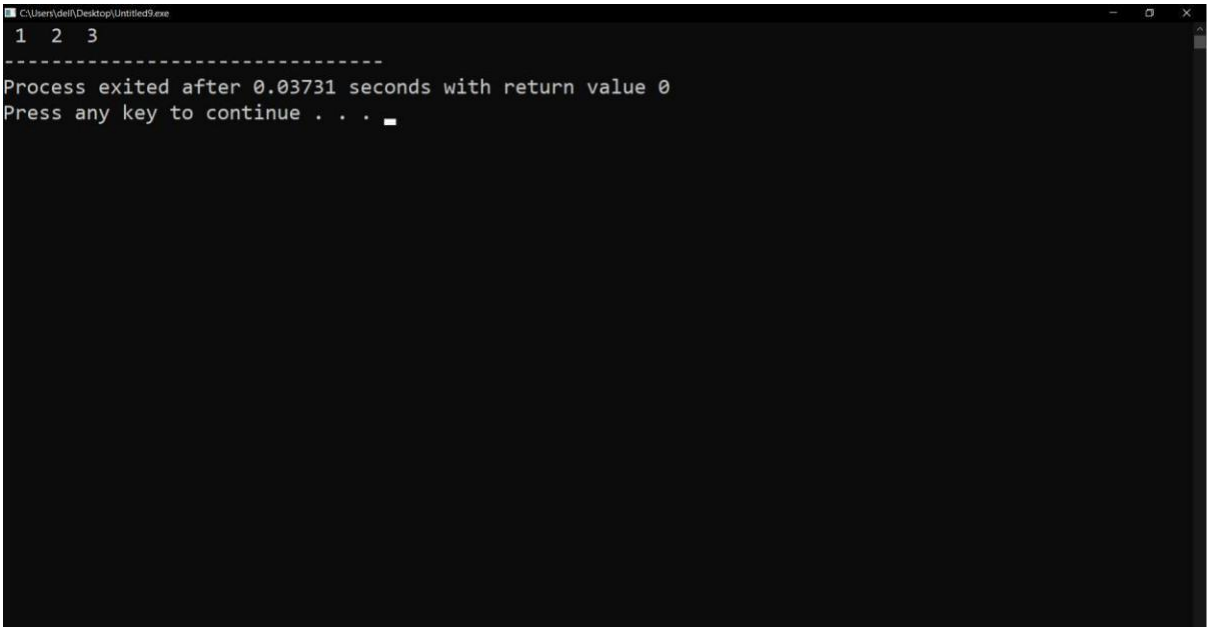
OBJECTIVE:

Write a program in C to implement Singly Linked List.

CODING:

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

struct Node {
   int data;
   struct Node* next;
};
// This function prints contents of linked list starting from
// the given node
void printList(struct Node* n)
{
   while (n != NULL) {
      printf(" %d ", n->data);
      n = n->next;
   }
}

int main()
{
   struct Node* head = NULL;
   struct Node* second = NULL;
   struct Node* third = NULL;

   // allocate 3 nodes in the heap
   head = (struct Node*)malloc(sizeof(struct Node));
   second = (struct Node*)malloc(sizeof(struct Node));
   third = (struct Node*)malloc(sizeof(struct Node));

   head->data = 1; // assign data in first node head-
   >next = second; // Link first node with second

   second->data = 2; // assign data to second node
   second->next = third;

   third->data = 3; // assign data to third node
   third->next = NULL;

   printList(head);

   return 0;
}
```

OUTPUT:


C:\Users\dell\Desktop\Untitled9.exe
1 2 3

Process exited after 0.03731 seconds with return value 0
Press any key to continue . . .

PROGRAM NO. 4.0

OBJECTIVE:

Write a program in C to implement Circular Linked List.

CODING:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node *head;

void beginsert  ();
void lastinsert ();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void search();
void main ()
{
   int choice =0;
   while(choice != 7)
   {
      printf("\n**********Main Menu**********\n"); printf("\nChoose one option from the
      following list ...\n");
      printf("\n===============================================\n");
      printf("\n1.Insert in begining\n2.Insert at last\n3.Delete from Beginning\n4.Delete from l
ast\n5.Search for an
      element\n6.Show\n7.Exit\n"); printf("\nEnter
      your choice?\n"); scanf("\n%d",&choice);
      switch(choice)
      {
         case 1:
         beginsert();
         break;
         case 2:
         lastinsert();
         break;
         case 3:
         begin_delete();
         break;
         case 4:
```

```
            last_delete();
            break;
            case 5:
            search();
            break;
            case 6:
            display();
            break;
            case 7:
            exit(0);
            break;
            default:
            printf("Please enter valid choice..");
        }
    }
}
void beginsert()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter the node data?");
        scanf("%d",&item);
        ptr -> data = item;
        if(head == NULL)
        {
            head = ptr;
            ptr -> next = head;
        }
        else
        {
            temp = head;
            while(temp->next != head)
                temp = temp->next;
            ptr->next = head;
            temp -> next = ptr;
            head = ptr;
        }
        printf("\nnode inserted\n");
    }

}
void lastinsert()
{
```

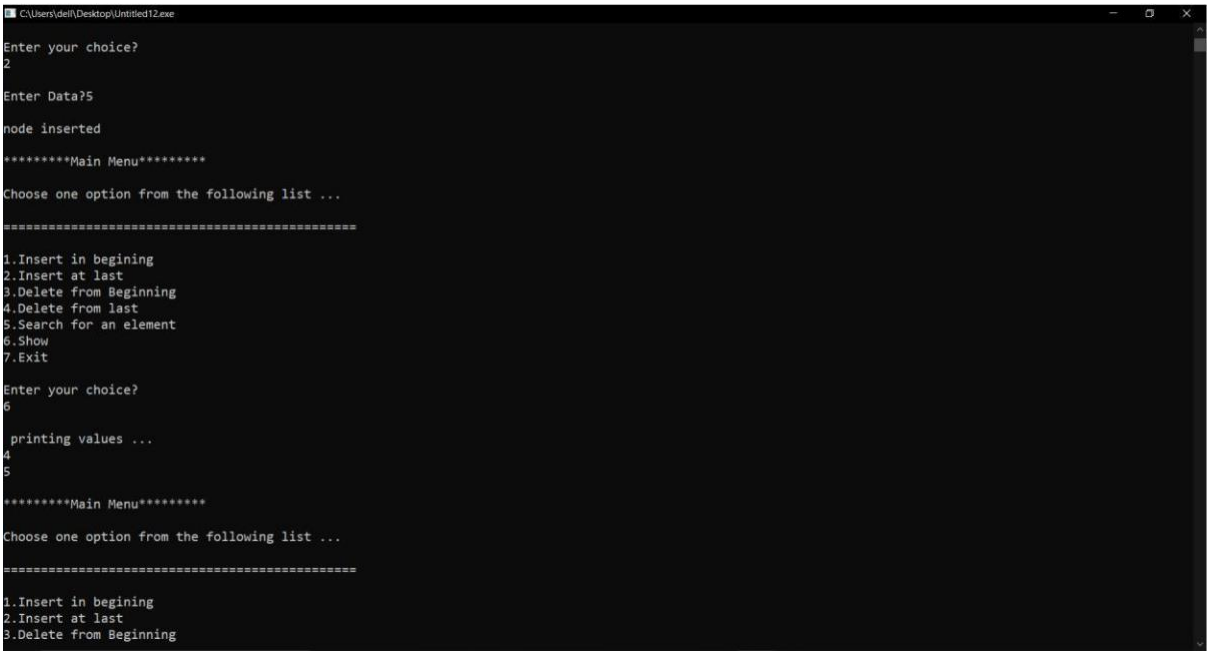
```
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW\n");
    }
    else
    {
        printf("\nEnter Data?");
        scanf("%d",&item);
        ptr->data = item;
        if(head == NULL)
        {
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

        printf("\nnode inserted\n");
    }

}

void begin_delete()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\nUNDERFLOW");
    }
    else if(head->next == head)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }

    else
    { ptr = head;
        while(ptr -> next != head)
```

```
            ptr = ptr -> next;
        ptr->next = head->next;
        free(head);
        head = ptr->next;
        printf("\nnode deleted\n");

    }
}
void last_delete()
{
    struct node *ptr, *preptr;
    if(head==NULL)
    {
        printf("\nUNDERFLOW");
    }
    else if (head ->next == head)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");

    }
    else
    {
        ptr = head;
        while(ptr ->next != head)
        {
            preptr=ptr;
            ptr = ptr->next;
        }
        preptr->next = ptr -> next;
        free(ptr);
        printf("\nnode deleted\n");
    }
}

void search()
{
    struct node *ptr;
    int item,i=0,flag=1;
    ptr = head;
    if(ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
    {
        printf("\nEnter item which you want to search?\n");
        scanf("%d",&item);
        if(head ->data == item)
```

```
        {
        printf("item found at location %d",i+1);
        flag=0;
        }
        else
        {
        while (ptr->next != head)
        {
           if(ptr->data == item)
           {
              printf("item found at location %d ",i+1);
              flag=0;
              break;
           }
           else
           {
              flag=1;
           }
           i++;
           ptr = ptr -> next;
        }
        }
        if(flag != 0)
        {
           printf("Item not found\n");
        }
     }

}
void display()
{
   struct node *ptr;
   ptr=head;
   if(head == NULL)
   {
      printf("\nnothing to print");
   }
   else
   {
      printf("\n printing values ... \n");

      while(ptr -> next != head)
      {

         printf("%d\n", ptr -> data);
         ptr = ptr -> next;
      }
      printf("%d\n", ptr -> data);
   }
}
```

OUTPUT:


C:\Users\dell\Desktop\Untitled12.exe
*********Main Menu*********
Choose one option from the following list ...
==========
1.Insert in begining
2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit
Enter your choice?
1
Enter the node data?4
node inserted
*********Main Menu*********
Choose one option from the following list ...
==========
1.Insert in begining
2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit
Enter your choice?
2


C:\Users\dell\Desktop\Untitled12.exe
Enter your choice?
2
Enter Data?5
node inserted
*********Main Menu*********
Choose one option from the following list ...
==========
1.Insert in begining
2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit
Enter your choice?
6
printing values ...
4
5
*********Main Menu*********
Choose one option from the following list ...
==========
1.Insert in begining
2.Insert at last
3.Delete from Beginning

PROGRAM NO. 5.0

OBJECTIVE:

Write a program in C to implement Doubly Linked List.

CODING:

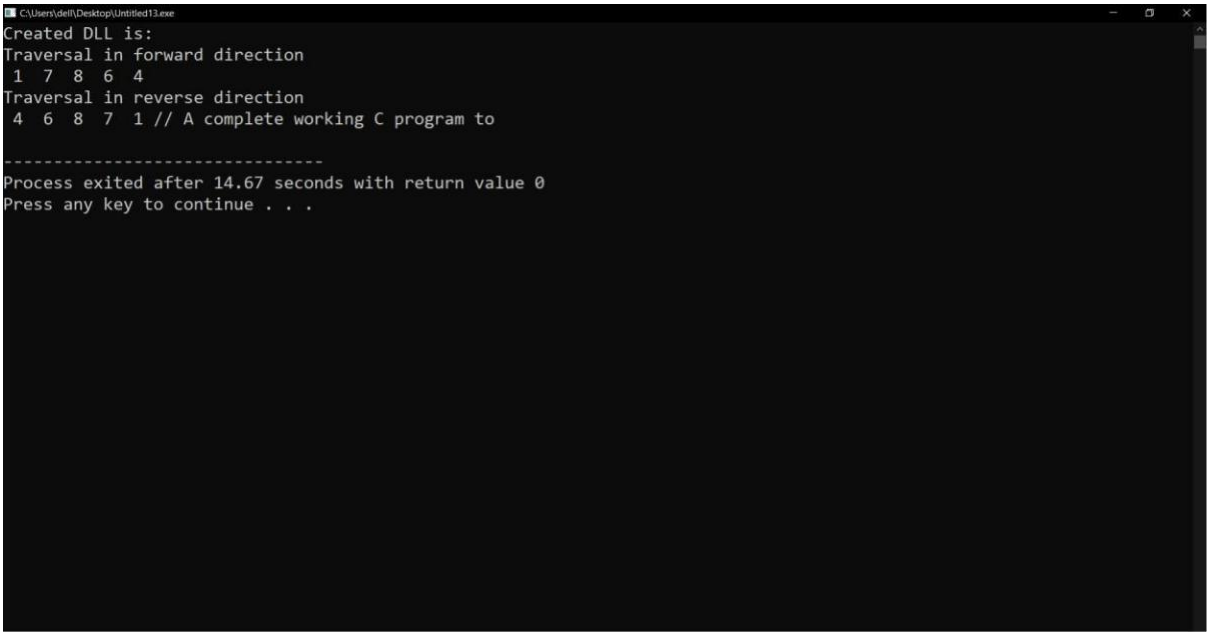
```
// A complete working C program to
// demonstrate all insertion
//   methods
#include <stdio.h>
#include <stdlib.h>

// A linked list node
struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
};

/* Given a reference (pointer to pointer) to the head of a
   list and an int, inserts a new node on the front of the
   list. */
void push(struct Node** head_ref, int new_data)
{
    /* 1. allocate node */
    struct Node* new_node
        = (struct Node*)malloc(sizeof(struct Node));

    /* 2. put in the data */
    new_node->data = new_data;

    /* 3. Make next of new node as head and previous as NULL
     */
    new_node->next = (*head_ref);
    new_node->prev = NULL;

    /* 4. change prev of head node to new node */
    if ((*head_ref) != NULL)
        (*head_ref)->prev = new_node;

    /* 5. move the head to point to the new node */
    (*head_ref) = new_node;
}

/* Given a node as prev_node, insert a new node after the
 * given node */
void insertAfter(struct Node* prev_node, int new_data)
```

```
{
    /*1. check if the given prev_node is NULL */
    if (prev_node == NULL) {
        printf("the given previous node cannot be NULL");
        return;
    }

    /* 2. allocate new node */
    struct Node* new_node
        = (struct Node*)malloc(sizeof(struct Node));

    /* 3. put in the data */
    new_node->data = new_data;

    /* 4. Make next of new node as next of prev_node */
    new_node->next = prev_node->next;

    /* 5. Make the next of prev_node as new_node */
    prev_node->next = new_node;

    /* 6. Make prev_node as previous of new_node */
    new_node->prev = prev_node;

    /* 7. Change previous of new_node's next node */
    if (new_node->next != NULL)
        new_node->next->prev = new_node;
}

/* Given a reference (pointer to pointer) to the head
   of a DLL and an int, appends a new node at the end */
void append(struct Node** head_ref, int new_data) {

    /* 1. allocate node */
    struct Node* new_node
        = (struct Node*)malloc(sizeof(struct Node));

    struct Node* last = *head_ref; /* used in step 5*/

    /* 2. put in the data */
    new_node->data = new_data;

    /* 3. This new node is going to be the last node, so
          make next of it as NULL*/
    new_node->next = NULL;

    /* 4. If the Linked List is empty, then make the new
          node as head */
    if (*head_ref == NULL) {
        new_node->prev = NULL;
        *head_ref = new_node;
```

```
        return;
    }

    /* 5. Else traverse till the last node */
    while (last->next != NULL)
        last = last->next;

    /* 6. Change the next of last node */
    last->next = new_node;

    /* 7. Make last node as previous of new node */
    new_node->prev = last;

    return;
}

// This function prints contents of linked list starting
// from the given node
void printList(struct Node* node)
{
    struct Node* last;
    printf("\nTraversal in forward direction \n");
    while (node != NULL) {
        printf(" %d ", node->data);
        last = node;
        node = node->next;
    }

    printf("\nTraversal in reverse direction \n");
    while (last != NULL) {
        printf(" %d ", last->data);
        last = last->prev;
    }
}

/* Driver program to test above functions*/
int main()
{
    /* Start with the empty list */
    struct Node* head = NULL;

    // Insert 6. So linked list becomes 6->NULL
    append(&head, 6);

    // Insert 7 at the beginning. So linked list becomes
    // 7->6->NULL
    push(&head, 7);

    // Insert 1 at the beginning. So linked list becomes
    // 1->7->6->NULL
```

```
    push(&head, 1);

    // Insert 4 at the end. So linked list becomes
    // 1->7->6->4->NULL
    append(&head, 4);

    // Insert 8, after 7. So linked list becomes
    //  1->7->8->6->4->NULL
    insertAfter(head->next, 8);

    printf("Created DLL is: ");
    printList(head);

    getchar();
    return 0;
}
```

OUTPUT:


C:\Users\dell\Desktop\Untitled13.exe
Created DLL is:
Traversal in forward direction
1 7 8 6 4
Traversal in reverse direction
4 6 8 7 1 // A complete working C program to

Process exited after 14.67 seconds with return value 0
Press any key to continue . . .

PROGRAM NO. 6.0

OBJECTIVE:

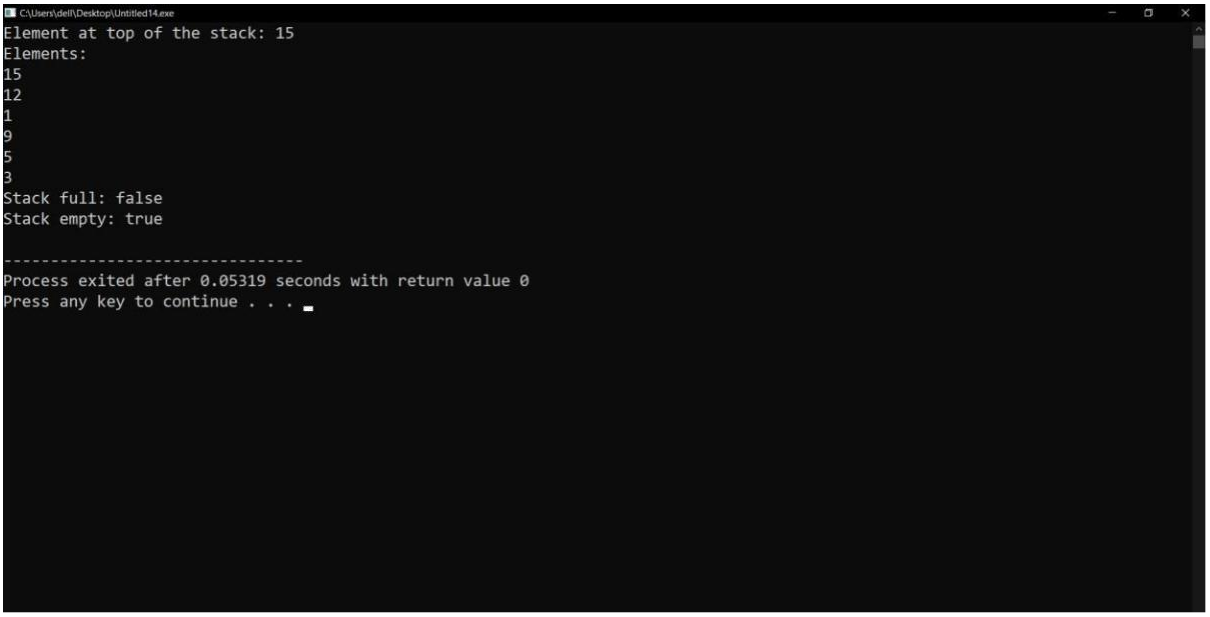
Write a program in C to implement Stack.

CODING:

```
#include <stdio.h>

int MAXSIZE = 8;
int stack[8];
int top = -1;

int isempty() {

   if(top == -1)
      return 1;
   else
      return 0;
}

int isfull() {

   if(top == MAXSIZE)
      return 1;
   else
      return 0;
}

int peek() {
   return stack[top];
}

 int pop() {
   int data;

   if(!isempty()) {
      data = stack[top];
      top = top - 1;
      return data;
   } else {
      printf("Could not retrieve data, Stack is empty.\n");
   }
}

int push(int data) {
```

```
   if(!isfull()) {
      top = top + 1;
      stack[top] = data;
   } else {
      printf("Could not insert data, Stack is full.\n");
   }
}

int main() {
   // push items on to the
   stack push(3);
   push(5);
   push(9);
   push(1);
   push(12);
   push(15);

   printf("Element at top of the stack: %d\n" ,peek());
   printf("Elements: \n");

   // print stack data
   while(!isempty()) { int
      data = pop();
      printf("%d\n",data);
   }

   printf("Stack full: %s\n" , isfull()?"true":"false");
   printf("Stack empty: %s\n" , isempty()?"true":"false");

   return 0;
}
```

OUTPUT:


C:\Users\dell\Desktop\Untitled14.exe
Element at top of the stack: 15
Elements:
15
12
1
9
5
3
Stack full: false
Stack empty: true

Process exited after 0.05319 seconds with return value 0
Press any key to continue . . .

PROGRAM NO. 7.0

OBJECTIVE:

Write a program in C to implement Circular Queue.

CODING:

```
#include <stdio.h>
# define max 6
int queue[max]; // array declaration
int front=-1;
int rear=-1;
// function to insert an element in a circular queue
void enqueue(int element)
{
   if(front==-1 && rear==-1) // condition to check queue is empty
   {
      front=0;
      rear=0;
      queue[rear]=element;
   }
   else if((rear+1)%max==front) // condition to check queue is full
   {
      printf("Queue is overflow..");
   }
   else
   {
      rear=(rear+1)%max;        // rear is incremented
      queue[rear]=element;      // assigning a value to the queue at the rear position.
   }
}
// function to delete the element from the queue
int dequeue()
{
   if((front==-1) && (rear==-1)) // condition to check queue is empty
   {
      printf("\nQueue is underflow..");
   }
 else if(front==rear)
{
  printf("\nThe dequeued element is %d", queue[front]);
  front=-1;
  rear=-1;
}
else
{
   printf("\nThe dequeued element is %d", queue[front]);
  front=(front+1)%max;
```

```
}
}
// function to display the elements of a queue
void display()
{
   int i=front; if(front==-1
   && rear==-1)
   {
      printf("\n Queue is empty..");
   }
   else
   {
      printf("\nElements in a Queue are :");
      while(i<=rear)
      {
         printf("%d,", queue[i]);
         i=(i+1)%max;
      }
   }
}
int main()
{
   int choice=1,x; // variables declaration

   while(choice<4 && choice!=0) // while loop
   {
   printf("\n Press 1: Insert an element");
   printf("\nPress 2: Delete an element");
   printf("\nPress 3: Display the element");
   printf("\nEnter your choice");
   scanf("%d", &choice);

   switch(choice)
   {

      case 1:

      printf("Enter the element which is to be inserted");
      scanf("%d", &x);
      enqueue(x);
      break;
      case 2:
      dequeue();
      break;
      case 3:
      display();
      break;
   }}
   return 0;
}
```

OUTPUT:


C:\Users\dell\Desktop\Untitled15.exe
Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice1
Enter the element which is to be inserted20
Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice1
Enter the element which is to be inserted40
Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice1
Enter the element which is to be inserted50
Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice3
Elements in a Queue are :20,40,50,
Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice

PROGRAM NO. 8.0

OBJECTIVE:

Write a program in C to implement De-Queue.

CODING:

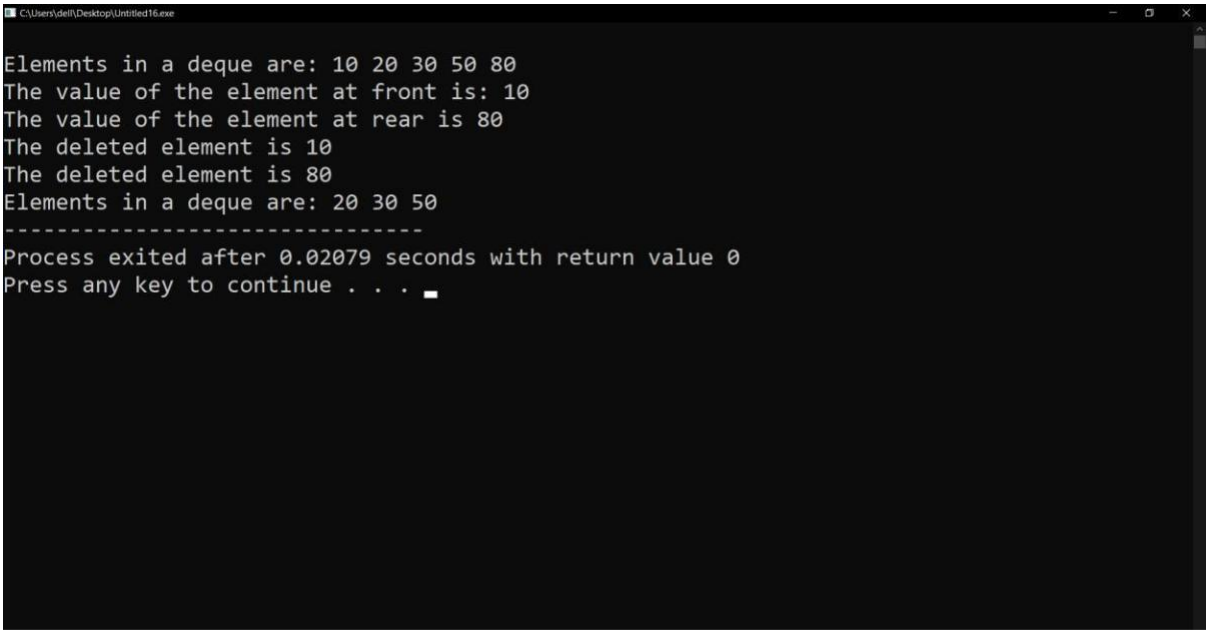
```
#include <stdio.h>
#define size 5
int deque[size];
int f = -1, r = -1;
// insert_front function will insert the value from the front
void insert_front(int x)
{
   if((f==0 && r==size-1) || (f==r+1))
   {
      printf("Overflow");
   }
   else if((f==-1) && (r==-1))
   {
      f=r=0;
      deque[f]=x;
   }
   else if(f==0)
   {
      f=size-1;
      deque[f]=x;
   }
   else
   {
      f=f-1;
      deque[f]=x;
   }
}

// insert_rear function will insert the value from the
rear void insert_rear(int x)
{
   if((f==0 && r==size-1) || (f==r+1))
   {
      printf("Overflow");
   }
   else if((f==-1) && (r==-1))
   {
      r=0;
      deque[r]=x;
   }
   else if(r==size-1)
```

```
    {
        r=0;
        deque[r]=x;
    }
    else
    {
        r++;
        deque[r]=x;
    }

}

// display function prints all the value of deque.
void display()
{
    int i=f;
    printf("\nElements in a deque are: ");

    while(i!=r)
    {
        printf("%d ",deque[i]);
        i=(i+1)%size;
    }
     printf("%d",deque[r]);
}

// getfront function retrieves the first value of the
deque. void getfront()
{
    if((f==-1) && (r==-1))
    {
        printf("Deque is empty");
    }
    else
    {
        printf("\nThe value of the element at front is: %d", deque[f]);
    }

}

// getrear function retrieves the last value of the deque.
void getrear()
{
    if((f==-1) && (r==-1))
    {
        printf("Deque is empty");
    }
    else
    {
        printf("\nThe value of the element at rear is %d", deque[r]);
```

```
    }

}

// delete_front() function deletes the element from the front
void delete_front()
{
    if((f==-1) && (r==-1))
    {
        printf("Deque is empty");
    }
    else if(f==r)
    {
        printf("\nThe deleted element is %d",
        deque[f]); f=-1;
        r=-1;

    }
     else if(f==(size-1))
     {
         printf("\nThe deleted element is %d",
         deque[f]); f=0;
     }
     else
     {
          printf("\nThe deleted element is %d",
          deque[f]); f=f+1;
     }
}

// delete_rear() function deletes the element from the rear
void delete_rear()
{
    if((f==-1) && (r==-1))
    {
        printf("Deque is empty");
    }
    else if(f==r)
    {
        printf("\nThe deleted element is %d",
        deque[r]); f=-1;
        r=-1;

    }
     else if(r==0)
     {
         printf("\nThe deleted element is %d",
         deque[r]); r=size-1;
     }
     else
```

```
    {
        printf("\nThe deleted element is %d",
        deque[r]); r=r-1;
    }
}

int main()
{
   insert_front(20);
   insert_front(10);
   insert_rear(30);
   insert_rear(50);
   insert_rear(80);
   display(); // Calling the display function to retrieve the values of
   deque getfront(); // Retrieve the value at front-end getrear(); //
   Retrieve the value at rear-end
   delete_front();
   delete_rear();
   display(); // calling display function to retrieve values after
   deletion return 0;
}
```

OUTPUT:


C:\Users\dell\Desktop\Untitled16.exe
Elements in a deque are: 10 20 30 50 80
The value of the element at front is: 10
The value of the element at rear is 80
The deleted element is 10
The deleted element is 80
Elements in a deque are: 20 30 50

Process exited after 0.02079 seconds with return value 0
Press any key to continue . . .

PROGRAM NO. 9.0

OBJECTIVE:

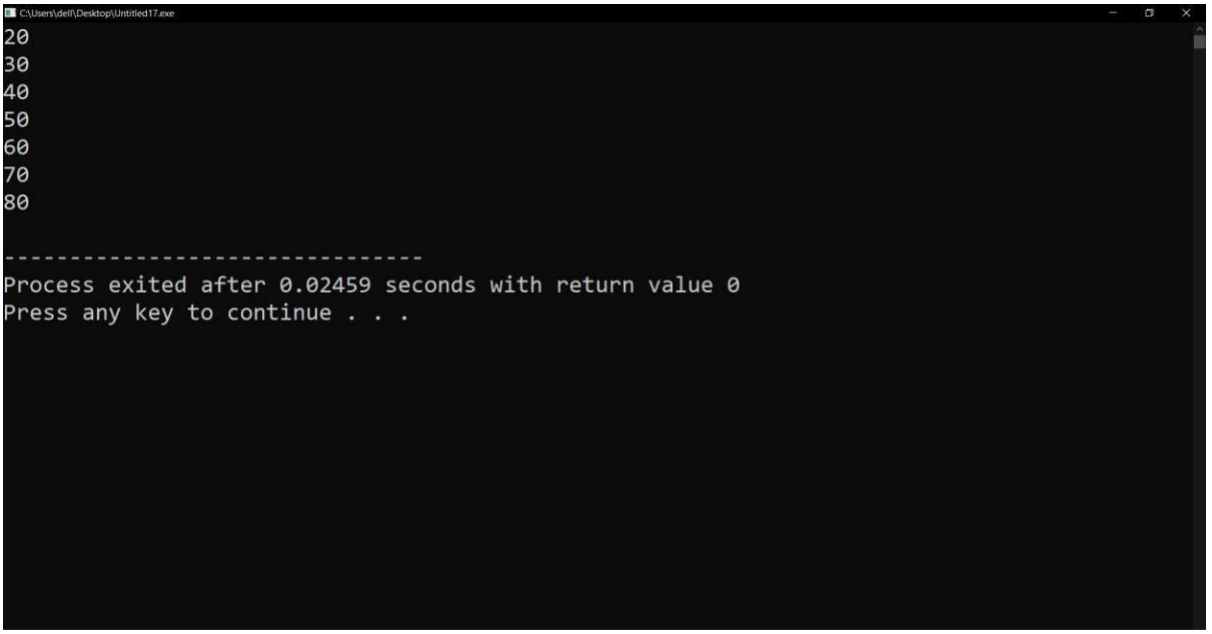
Write a program in C to implement Binary Search Tree Insertion.

CODING:

```
// C program to demonstrate insert
// operation in binary
// search tree.
#include <stdio.h>
#include <stdlib.h>

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

// A utility function to create a new BST
node struct node* newNode(int item)
{
    struct node* temp
        = (struct node*)malloc(sizeof(struct node));
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}

// A utility function to do inorder traversal of
BST void inorder(struct node* root)
{
    if (root != NULL) {
        inorder(root->left);
        printf("%d \n", root->key);
        inorder(root->right);
    }
}

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

    /* Otherwise, recur down the tree */
```

```
    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

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

// Driver Code
int main()
{
    /* Let us create following BST
              50
           /     \
          30      70
         / \   / \
       20 40 60    80 */
    struct node* root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);

    // print inoder traversal of the BST
    inorder(root);

    return 0;
}
```

OUTPUT:


C:\Users\dell\Desktop\Untitled17.exe
20
30
40
50
60
70
80

Process exited after 0.02459 seconds with return value 0
Press any key to continue . . .

PROGRAM NO. 10.0

OBJECTIVE:

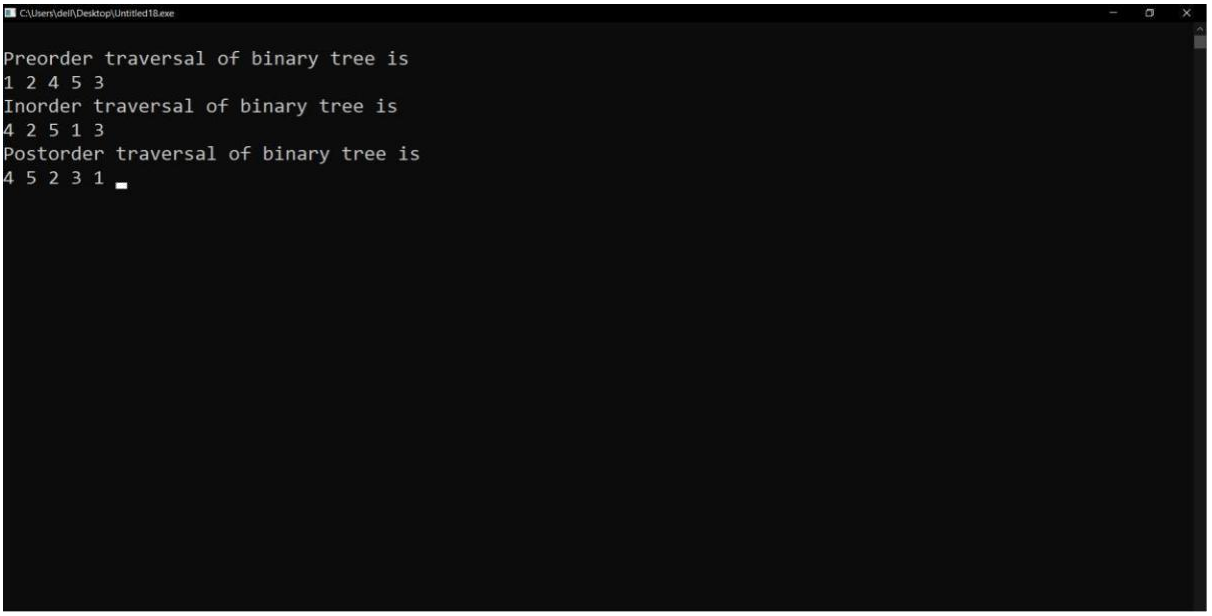
Write a program in C to implement Binary Search Tree Traversal (Inorder, Preorder, Postorder).

CODING:

```
// C program for different tree
traversals #include <stdio.h>
#include <stdlib.h>
/* A binary tree node has data, pointer to left
   child and a pointer to right child */
struct node {
    int data;
    struct node* left;
    struct node* right;
};
/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
struct node* newNode(int data)
{
    struct node* node
        = (struct node*)malloc(sizeof(struct
    node)); node->data = data;
    node->left = NULL;
    node->right = NULL;

    return (node);
}

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

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

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

    // now deal with the node
    printf("%d ", node->data);
}
/* Given a binary tree, print its nodes in inorder*/
void printInorder(struct node* node)
```

```
{
    if (node == NULL)
        return;

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

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

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

/* Given a binary tree, print its nodes in preorder*/
void printPreorder(struct node* node) {

    if (node == NULL)
        return;

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

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

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

/* Driver program to test above functions*/
int main()
{
    struct node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);

    printf("\nPreorder traversal of binary tree is \n");
    printPreorder(root);

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

    printf("\nPostorder traversal of binary tree is \n");
    printPostorder(root);
    getchar();
    return 0;
}
```

OUTPUT:


C:\Users\dell\Desktop\Untitled18.exe
Preorder traversal of binary tree is
1 2 4 5 3
Inorder traversal of binary tree is
4 2 5 1 3
Postorder traversal of binary tree is
4 5 2 3 1

PROGRAM NO. 11.1

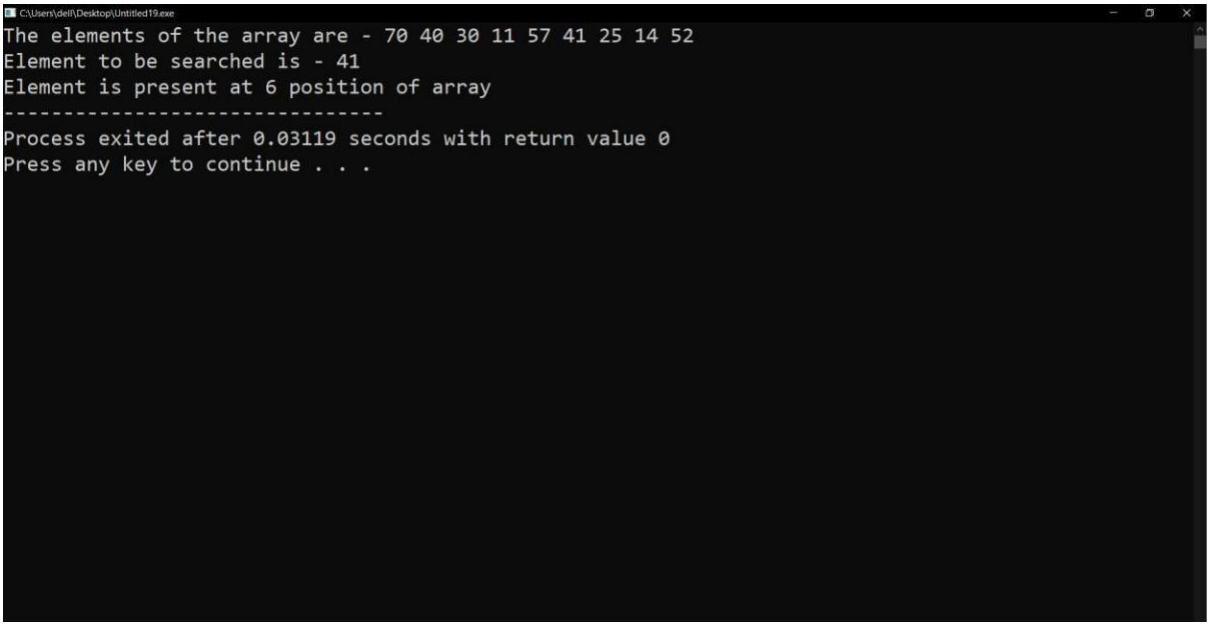
OBJECTIVE:

Write a program in C to implement Linear Search Algorithm.

CODING:

```
#include <stdio.h>
int linearSearch(int a[], int n, int val) {
  // Going through array sequencially
  for (int i = 0; i < n; i++)
    {
       if (a[i] == val)
       return i+1;
    }
  return -1;
}
int main() {
  int a[] = {70, 40, 30, 11, 57, 41, 25, 14, 52}; // given array
  int val = 41; // value to be searched
  int n = sizeof(a) / sizeof(a[0]); // size of array
  int res = linearSearch(a, n, val); // Store result
  printf("The elements of the array are - ");
  for (int i = 0; i < n; i++)
  printf("%d ", a[i]);
  printf("\nElement to be searched is - %d",
  val); if (res == -1)
  printf("\nElement is not present in the array");
  else
  printf("\nElement is present at %d position of array", res);
  return 0;
}
```

OUTPUT:


C:\Users\dell\Desktop\Untitled19.exe
The elements of the array are - 70 40 30 11 57 41 25 14 52
Element to be searched is - 41
Element is present at 6 position of array

Process exited after 0.03119 seconds with return value 0
Press any key to continue . . .

PROGRAM NO. 11.2

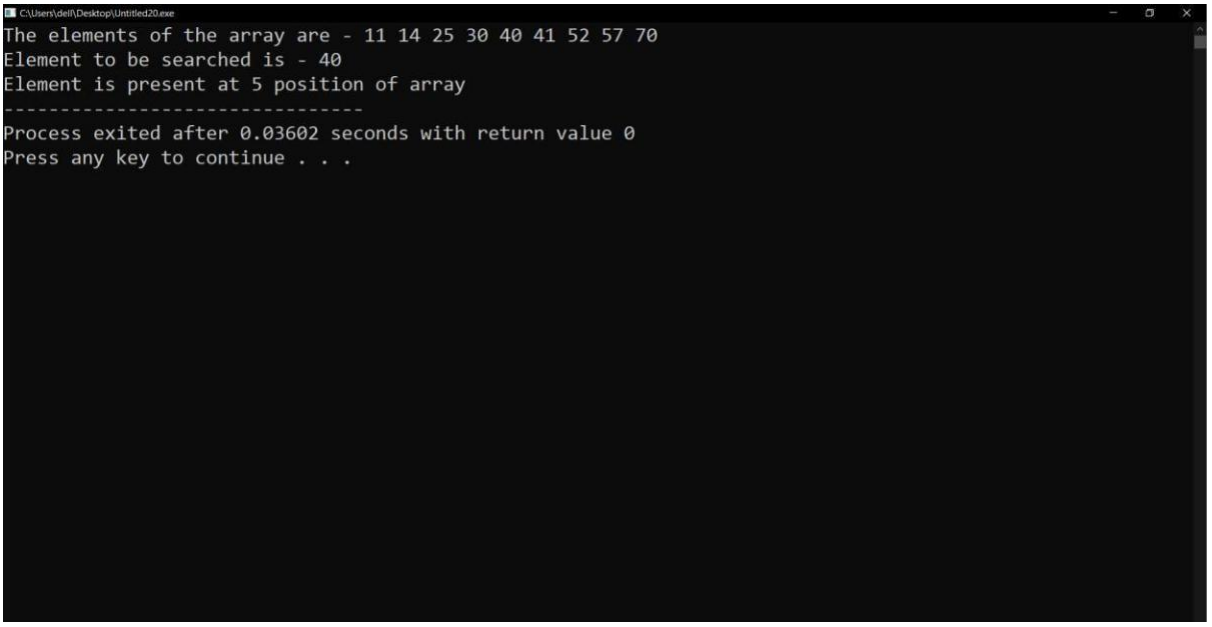
OBJECTIVE:

Write a program in C to implement Binary Search Algorithm.

CODING:

```
#include <stdio.h>
int binarySearch(int a[], int beg, int end, int val)
{
   int mid;
   if(end >= beg)
   {mid = (beg + end)/2;
/* if the item to be searched is present at middle
      */ if(a[mid] == val)
      {
         return mid+1;
      }
         /* if the item to be searched is smaller than middle, then it can only be in left subarray
 */
      else if(a[mid] < val)
      {
         return binarySearch(a, mid+1, end, val);
      }
         /* if the item to be searched is greater than middle, then it can only be in right subarra
y */
      else
      {
         return binarySearch(a, beg, mid-1, val);
      }
   }
   return -1;
}
int main() {
  int a[] = {11, 14, 25, 30, 40, 41, 52, 57, 70}; // given array
  int val = 40; // value to be searched
  int n = sizeof(a) / sizeof(a[0]); // size of array
  int res = binarySearch(a, 0, n-1, val); // Store
  result printf("The elements of the array are - ");
  for (int i = 0; i < n; i++)
  printf("%d ", a[i]);
  printf("\nElement to be searched is - %d",
  val); if (res == -1)
  printf("\nElement is not present in the array");
  else
  printf("\nElement is present at %d position of array",
  res); return 0;
}
```

OUTPUT:


C:\Users\dell\Desktop\Untitled20.exe
The elements of the array are - 11 14 25 30 40 41 52 57 70
Element to be searched is - 40
Element is present at 5 position of array

Process exited after 0.03602 seconds with return value 0
Press any key to continue . . .

PROGRAM NO. 12

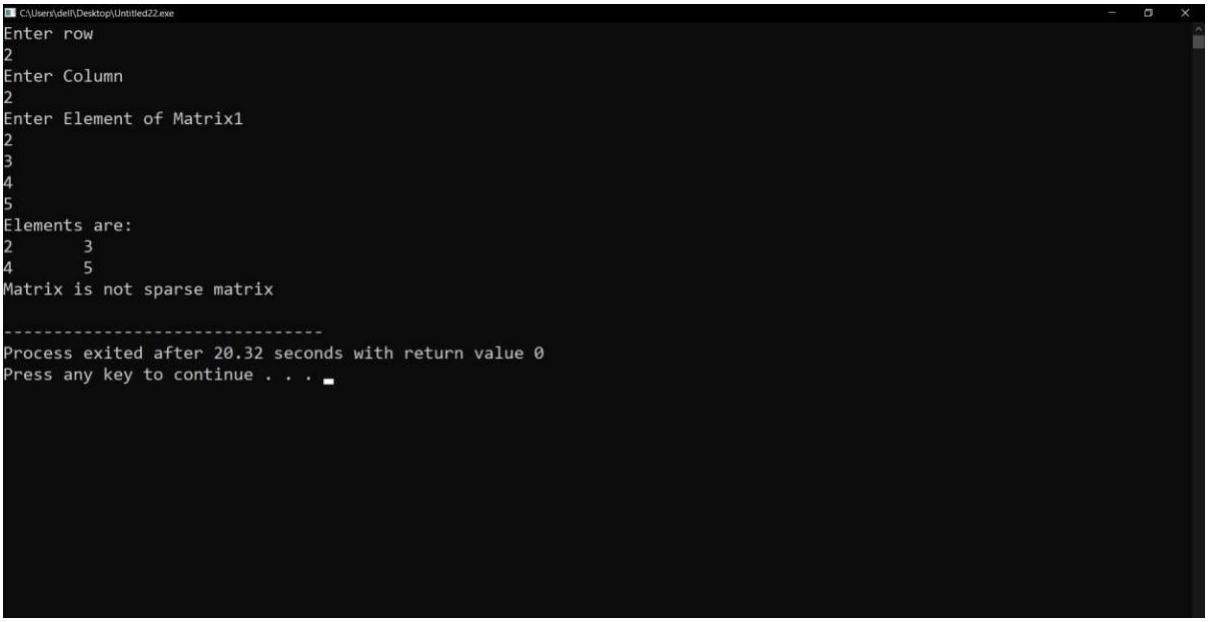
OBJECTIVE:

Write a program in C to implement Matrix insert by user. Check either matrix is sparse or not?

CODING:

```
#include<stdio.h>
#include<stdlib.h>
int main(){
  int row,col,i,j,a[10][10],count = 0;
  printf("Enter row\n");
  scanf("%d",&row);
  printf("Enter Column\n");
  scanf("%d",&col);
  printf("Enter Element of Matrix1\n");
  for(i = 0; i < row; i++){
    for(j = 0; j < col; j++){
      scanf("%d",&a[i][j]);
    }
  }
  printf("Elements are:\n");
  for(i = 0; i < row; i++){
    for(j = 0; j < col; j++){
      printf("%d\t",a[i][j]);
    }
    printf("\n");
  }
  /*checking sparse of matrix*/
  for(i = 0; i < row; i++){
    for(j = 0; j < col; j++){
      if(a[i][j] == 0)
        count++;
    }
  }
  if(count > ((row * col)/2))
    printf("Matrix is a sparse matrix \n");
  else
    printf("Matrix is not sparse matrix\n");
}
```

OUTPUT:


C:\Users\dell\Desktop\Untitled22.exe
Enter row
2
Enter Column
2
Enter Element of Matrix1
2
3
4
5
Elements are:
2 3
4 5
Matrix is not sparse matrix

Process exited after 20.32 seconds with return value 0
Press any key to continue . . .

PROGRAM NO. 13.0

OBJECTIVE:

Write a program in C to implement addition of two polynomials

CODING:

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

/*
  This structure is used to store a polynomial term. An array of such terms represents a
  polynomial.
  The "coeff" element stores the coefficient of a term in the polynomial,while
  the "exp" element stores the exponent.

*/
struct poly
{
  float coeff;
  int exp;
};


//declaration of polynomials
struct poly a[50],b[50],c[50],d[50];

int main()
{
 int i;
 int deg1,deg2; //stores degrees of the polynomial
 int k=0,l=0,m=0;


 printf("Enter the highest degree of poly1:");
 scanf("%d",&deg1);

 //taking polynomial terms from the user
 for(i=0;i<=deg1;i++)
 {

    //entering values in coefficient of the polynomial
    terms printf("\nEnter the coeff of x^%d :",i);
    scanf("%f",&a[i].coeff);

          //entering values in exponent of the polynomial terms
          a[k++].exp = i;
 }
```

```
//taking second polynomial from the user
printf("\nEnter the highest degree of poly2:");
scanf("%d",&deg2);

for(i=0;i<=deg2;i++)
{
     printf("\nEnter the coeff of x^%d :",i);
     scanf("%f",&b[i].coeff);

          b[l++].exp = i;
}


//printing first polynomial
 printf("\nExpression 1 = %.1f",a[0].coeff);
 for(i=1;i<=deg1;i++)
 {
   printf("+ %.1fx^%d",a[i].coeff,a[i].exp);
 }


 //printing second polynomial
 printf("\nExpression 2 = %.1f",b[0].coeff);
 for(i=1;i<=deg2;i++)
   {
     printf("+ %.1fx^%d",b[i].coeff,b[i].exp);
   }


//Adding the polynomials
 if(deg1>deg2)
   {
                  for(i=0;i<=deg2;i++)
                   {
                          c[m].coeff = a[i].coeff + b[i].coeff;
                          c[m].exp = a[i].exp;
                          m++;
                   }

                   for(i=deg2+1;i<=deg1;i++)
                   {
                          c[m].coeff = a[i].coeff;
                          c[m].exp = a[i].exp;
                          m++;
                   }

   }
```

```
 else
  {
    for(i=0;i<=deg1;i++)
     {
       c[m].coeff = a[i].coeff + b[i].coeff;
       c[m].exp = a[i].exp;
       m++;
     }

           for(i=deg1+1;i<=deg2;i++)
    {
      c[m].coeff = b[i].coeff;
      c[m].exp = b[i].exp;
      m++;
    }
  }



  //printing the sum of the two polynomials
  printf("\nExpression after additon = %.1f",c[0].coeff);
  for(i=1;i<m;i++)
  {
     printf("+ %.1fx^%d",c[i].coeff,c[i].exp);
   }

  return 0;
}
```

OUTPUT:


C:\Users\dell\Desktop\Untitled23.exe
Enter the highest degree of poly1:3
Enter the coeff of x^0 :2
Enter the coeff of x^1 :3
Enter the coeff of x^2 :1
Enter the coeff of x^3 :3
Enter the highest degree of poly2:4
Enter the coeff of x^0 :3
Enter the coeff of x^1 :2
Enter the coeff of x^2 :3
Enter the coeff of x^3 :1
Enter the coeff of x^4 :2
Expression 1 = 2.0+ 3.0x^1+ 1.0x^2+ 3.0x^3
Expression 2 = 3.0+ 2.0x^1+ 3.0x^2+ 1.0x^3+ 2.0x^4
Expression after additon = 5.0+ 5.0x^1+ 4.0x^2+ 4.0x^3+ 2.0x^4
Process exited after 39.6 seconds with return value 0
Press any key to continue . . .

PROGRAM NO. 14.0

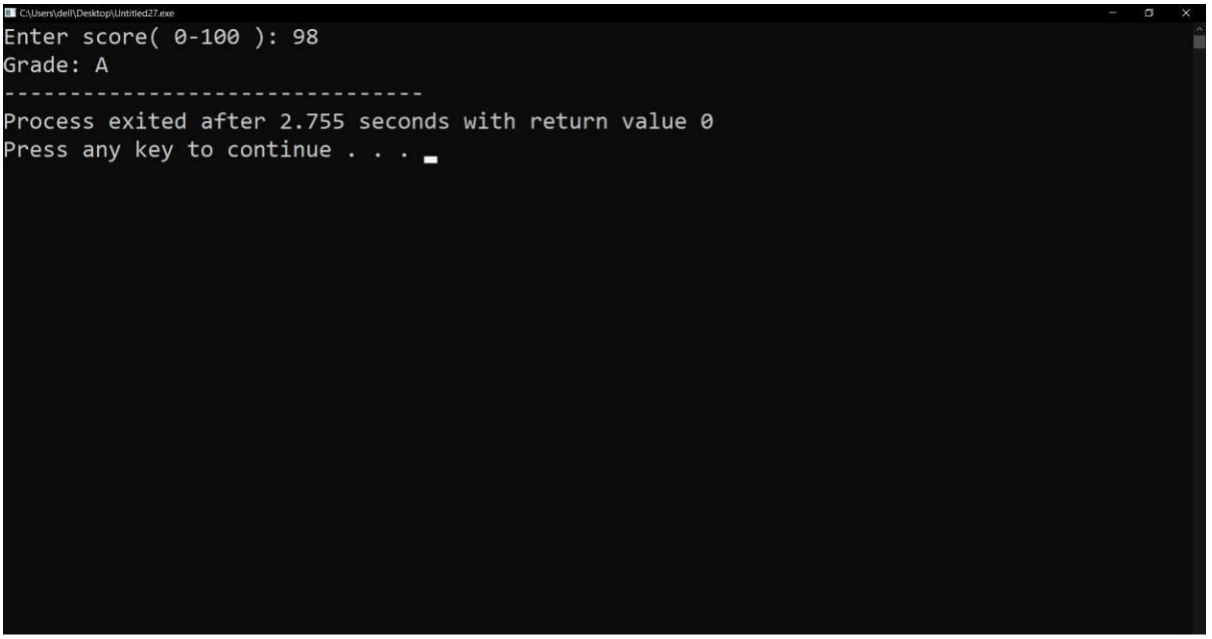
OBJECTIVE:

Write a program in C to implement to calculate grade of a student by given marks.

CODING:

```
#include<stdio.h>
int main()
{
  int score;

  printf("Enter score( 0-100 ): ");
  scanf("%d", &score);

  switch( score / 10 )
  {

  case 10:
  case 9:
    printf("Grade: A");
    break;

  case 8:
    printf("Grade: B");
    break;

  case 7:
    printf("Grade: C");
    break;

  case 6:
    printf("Grade: D");
    break;

  case 5:
    printf("Grade: E");
    break;

  default:
    printf("Grade: F");
    break;

  }

  return 0;
}
```

OUTPUT:


C:\Users\dell\Desktop\Untitled27.exe
Enter score(0-100): 98
Grade: A

Process exited after 2.755 seconds with return value 0
Press any key to continue . . .