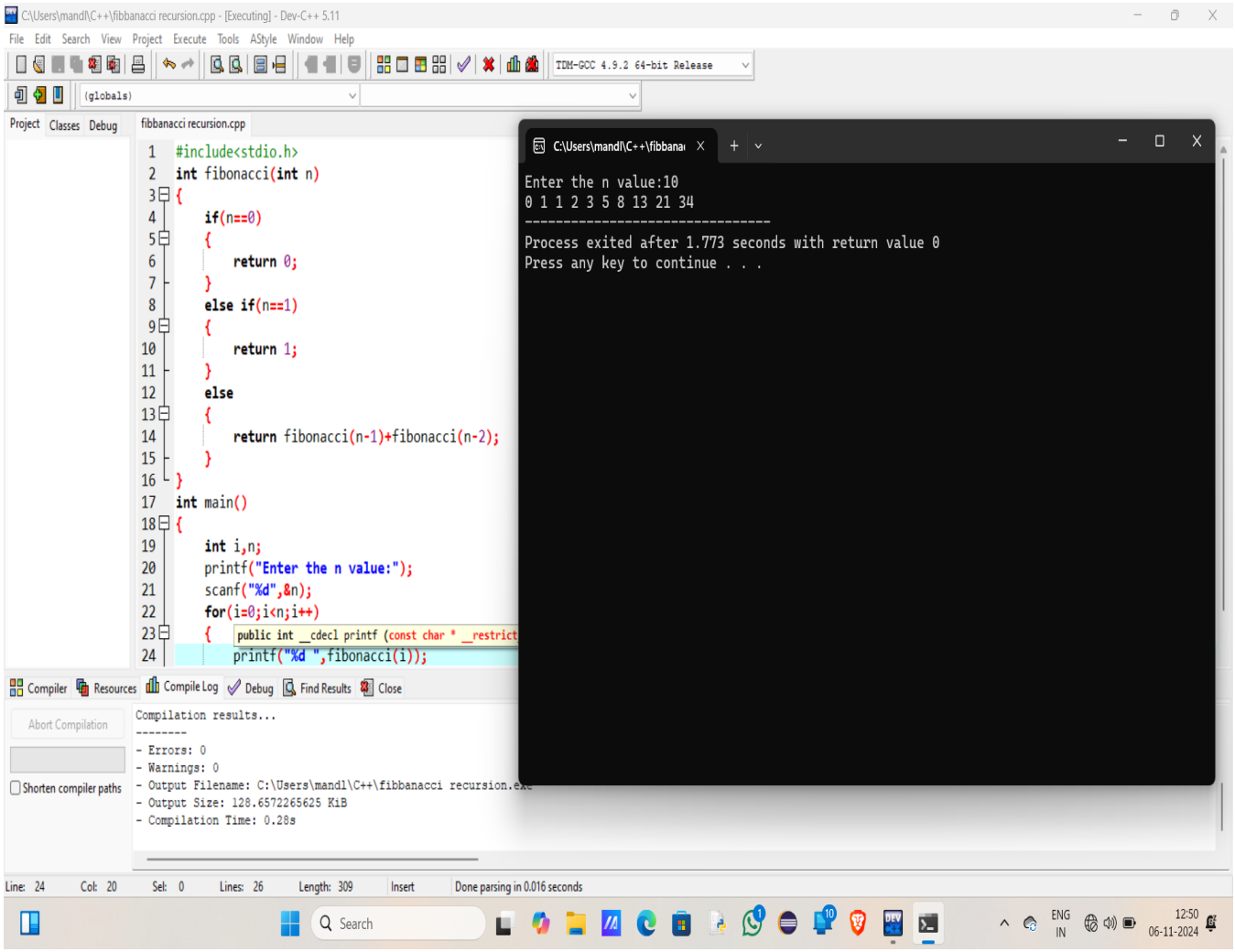
DAA PRACTICAL PROGRAMS

1. Fibonacci number using recursion

```
#include<stdio.h>
int fibonacci(int n)
{
	if(n==0)
	{
		return 0;
	}
	else if(n==1)
	{
		return 1;
	}
	else
	{
		return fibonacci(n-1)+fibonacci(n-2);
	}
}
int main()
{
	int i,n;
	printf("Enter the n value:");
	scanf("%d",&n);
	for(i=0;i<n;i++)
	{
		printf("%d ",fibonacci(i));
```

```
        }
}
```

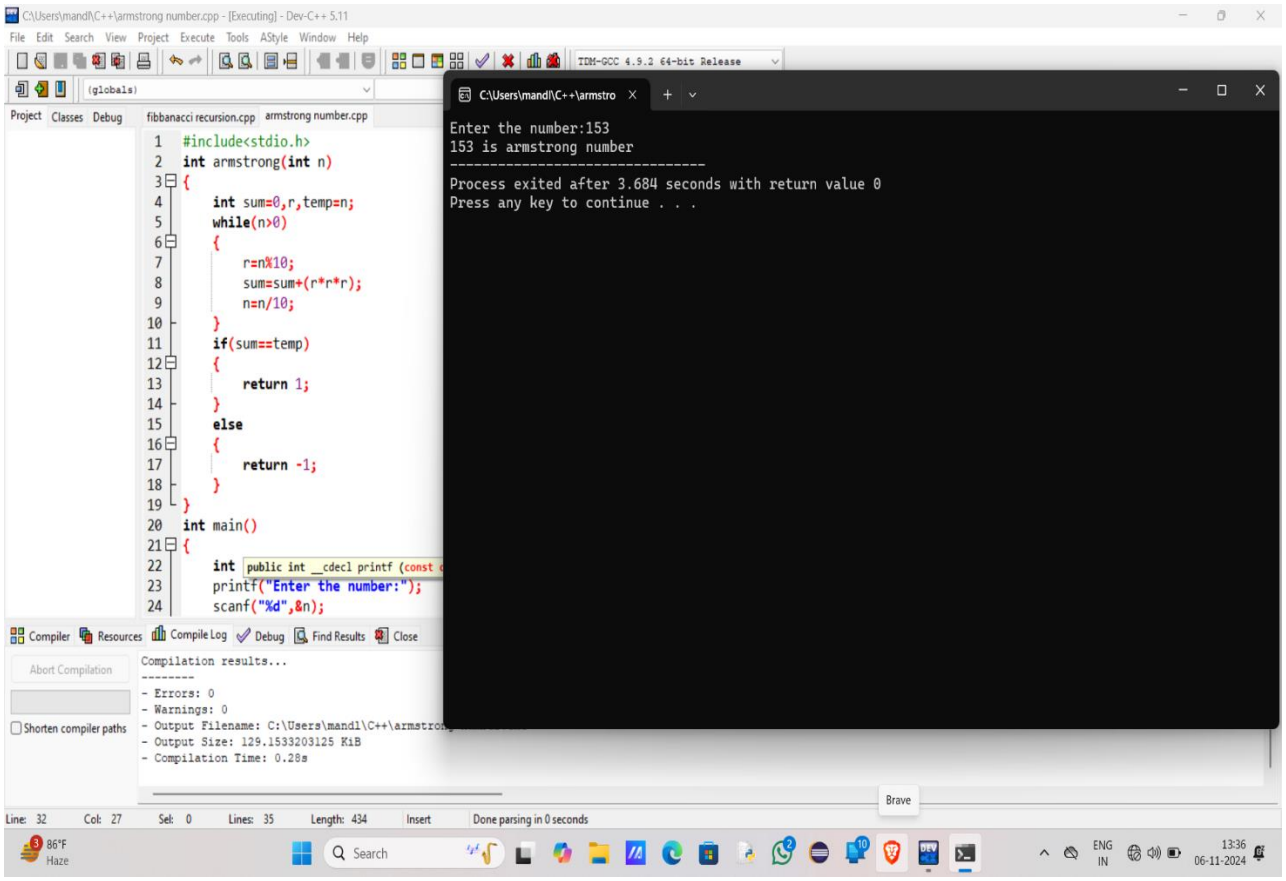


2. Armstrong number

```
#include<stdio.h>
int armstrong(int n)
{
        int sum=0,r,temp=n;
        while(n>0)
        {
                r=n%10;
                sum=sum+(r*r*r);
```

```
        n=n/10;
    }
    if(sum==temp)
    {
        return 1;
    }
    else
    {
        return -1;
    }
}
int main()
{
    int n;
    printf("Enter the number:");
    scanf("%d",&n);
    int result=armstrong(n);
    if(result==1)
    {
        printf("%d is armstrong number",n);
    }
    else
    {
        printf("%d is not armstrong number",n);
    }
    return 0;
```

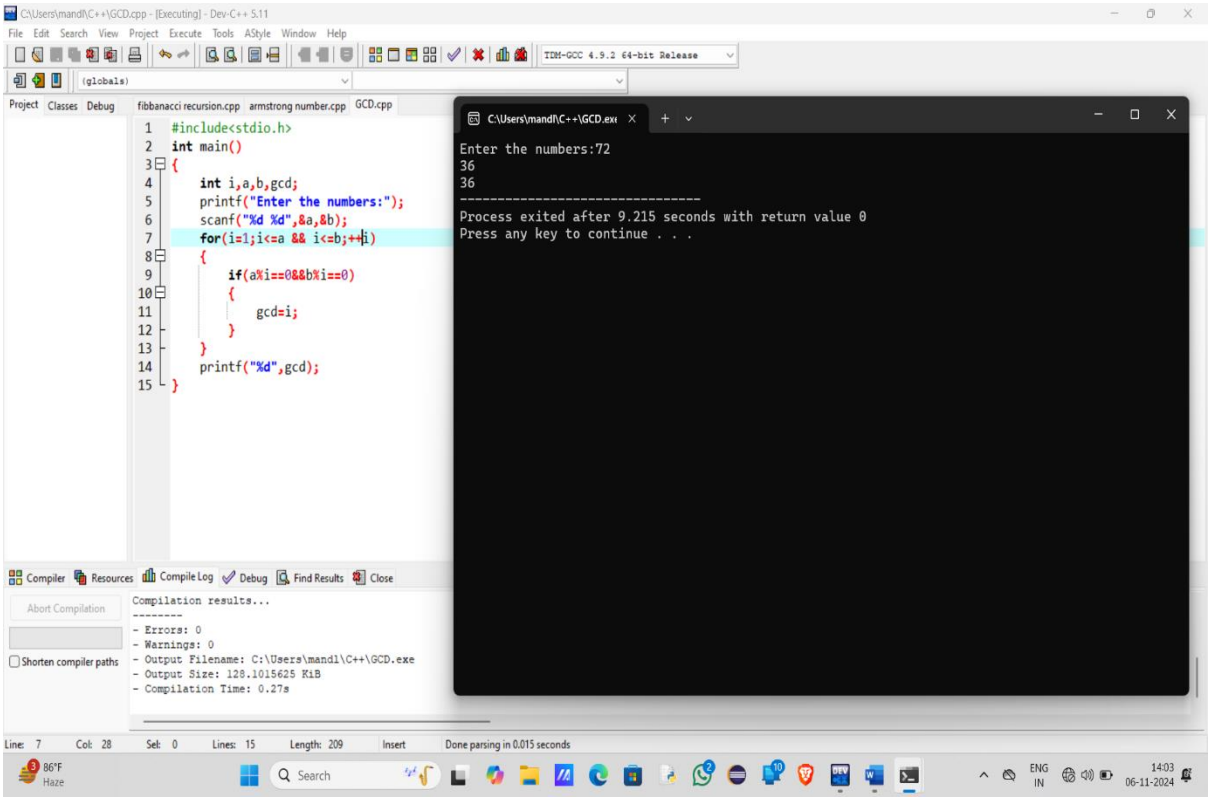
```
}
```



3. GCD

```
#include<stdio.h>
int main()
{
  int i,a,b,gcd;
```

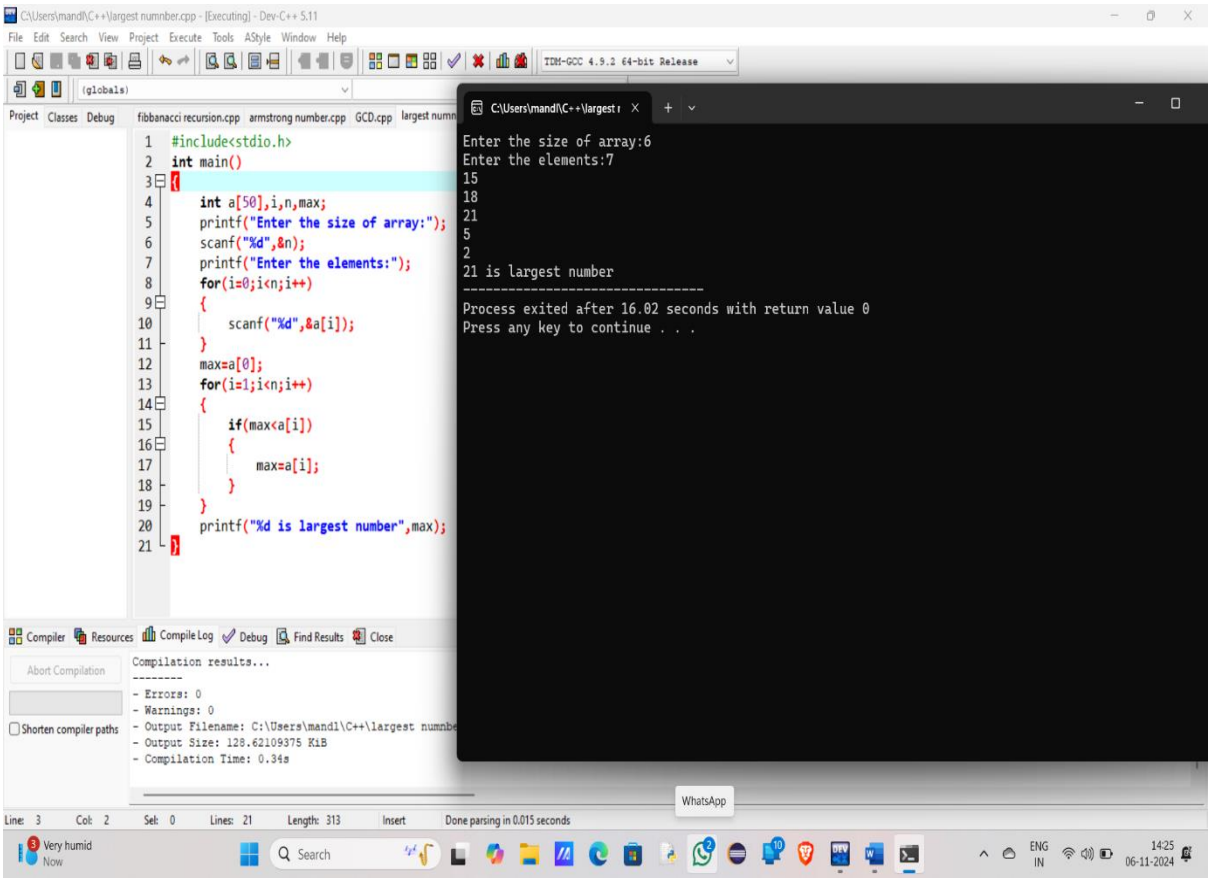
```
    printf("Enter the numbers:");
    scanf("%d %d",&a,&b);
    for(i=1;i<=a && i<=b;++i)
    {
        if(a%i==0&&b%i==0)
        {
            gcd=i;
        }
    }
    printf("%d",gcd);
}
```



4. Largest number in an array

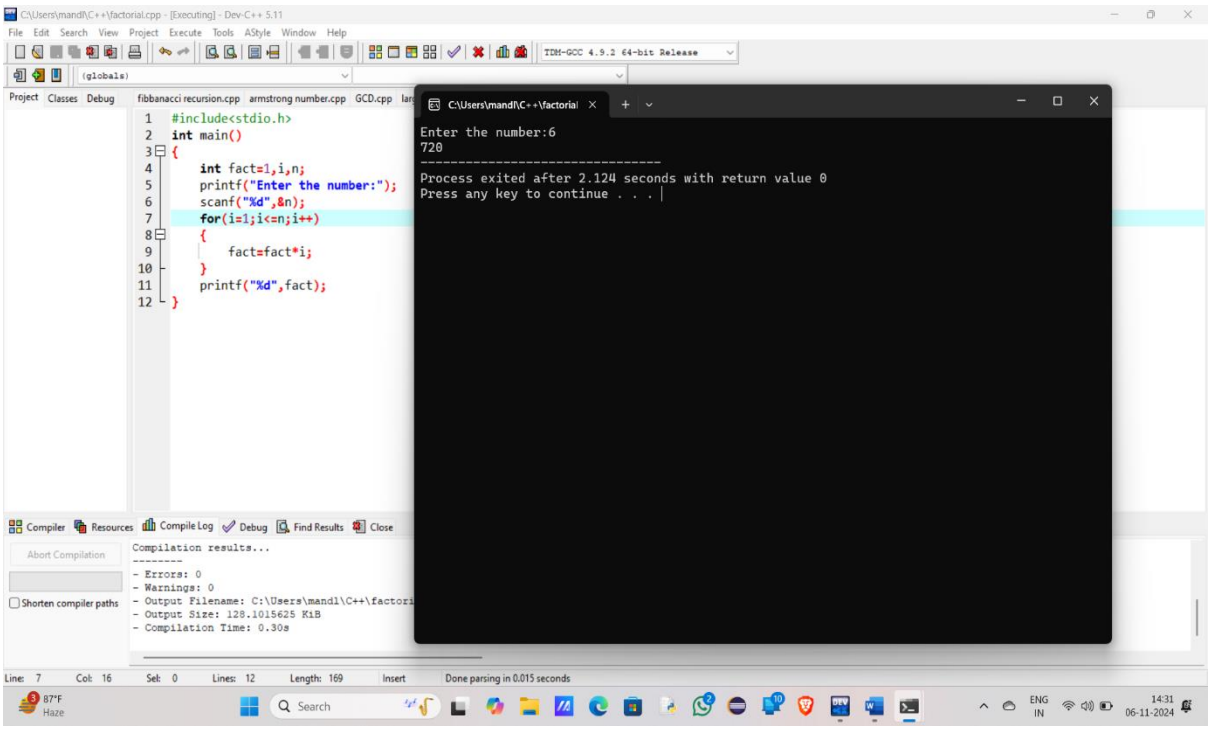
```
#include<stdio.h>
int main()
{
```

```
        int a[50],i,n,max;
        printf("Enter the size of array:");
        scanf("%d",&n);
        printf("Enter the elements:");
        for(i=0;i<n;i++)
        {
            scanf("%d",&a[i]);
        }
        max=a[0];
        for(i=1;i<n;i++)
        {
            if(max<a[i])
            {
                max=a[i];
            }
        }
        printf("%d is largest number",max);
}
```



5. Factorial of a number

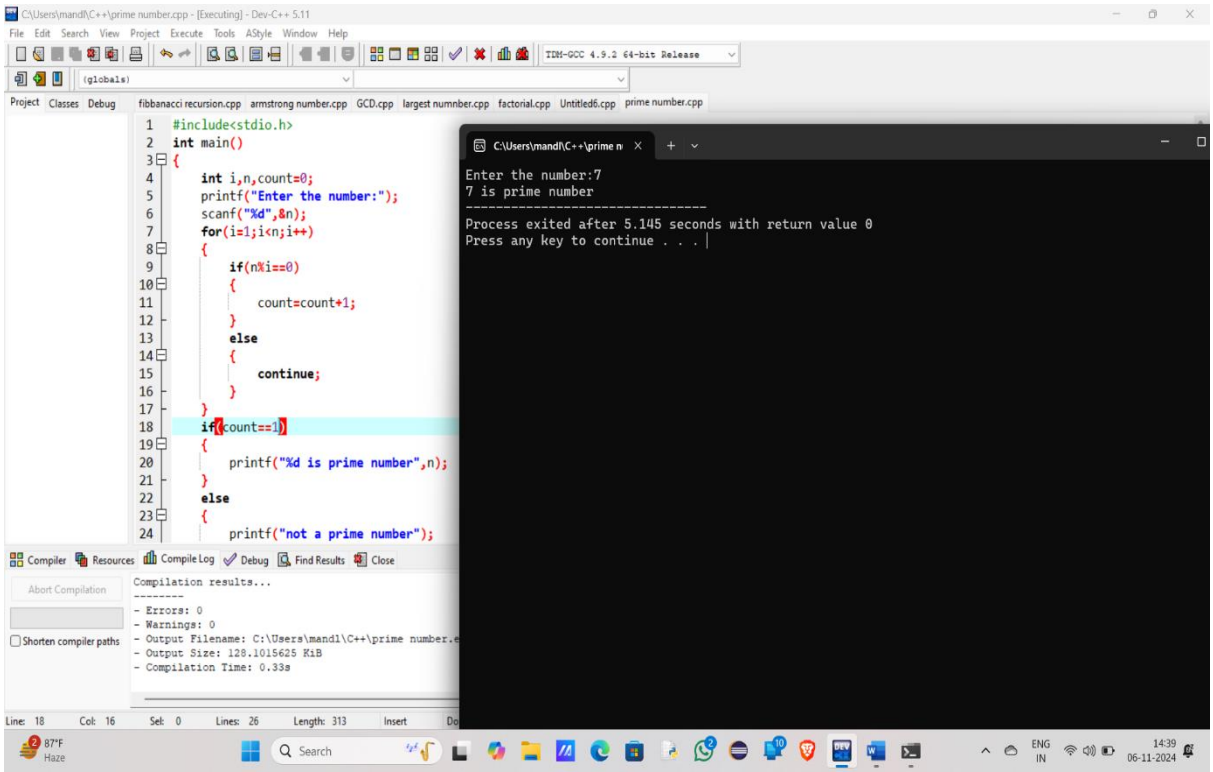
```
#include<stdio.h>
int main()
{
        int fact=1,i,n;
        printf("Enter the number:");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
          fact=fact*i;
        }
        printf("%d",fact);
}
```



6. Prime Number:

```
#include<stdio.h>
int main()
{
        int i,n,count=0;
        printf("Enter the number:");
        scanf("%d",&n);
        for(i=1;i<n;i++)
        {
```

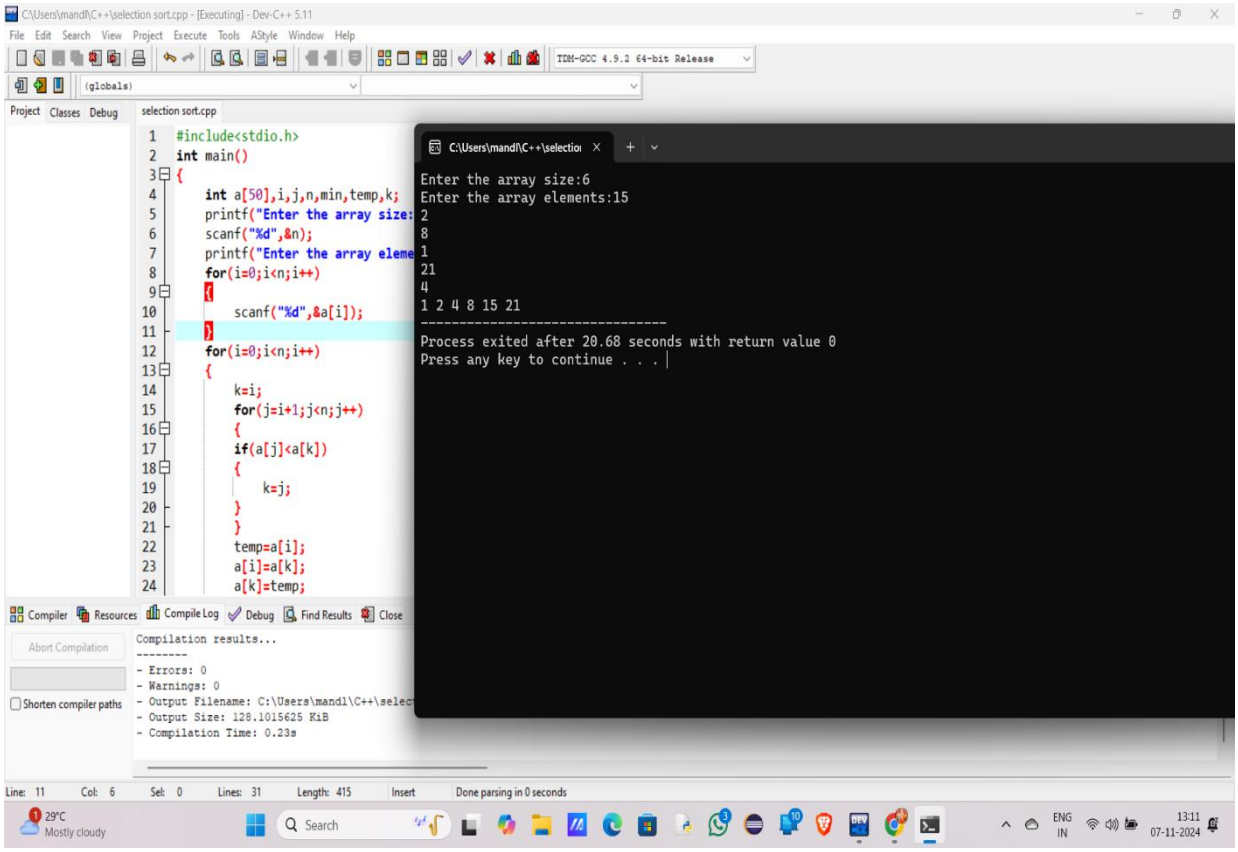
```
        if(n%i==0)
        {
                count=count+1;
        }
        else
        {
                continue;
        }
    }
    if(count==1)
    {
        printf("%d is prime number",n);
    }
    else
    {
        printf("not a prime number");
    }
}
```



7. To Perform Selection Section:

```
#include<stdio.h>
int main()
{
int a[50],i,j,n,min,temp,k;
printf("Enter the array size:");
scanf("%d",&n);
printf("Enter the array elements:");
for(i=0;i<n;i++)
{
        scanf("%d",&a[i]);
}
for(i=0;i<n;i++)
{
        k=i;
        for(j=i+1;j<n;j++)
        {
        if(a[j]<a[k])
        {
                k=j;
        }
        }
        temp=a[i];
        a[i]=a[k];
        a[k]=temp;
}
for(i=0;i<n;i++)
```

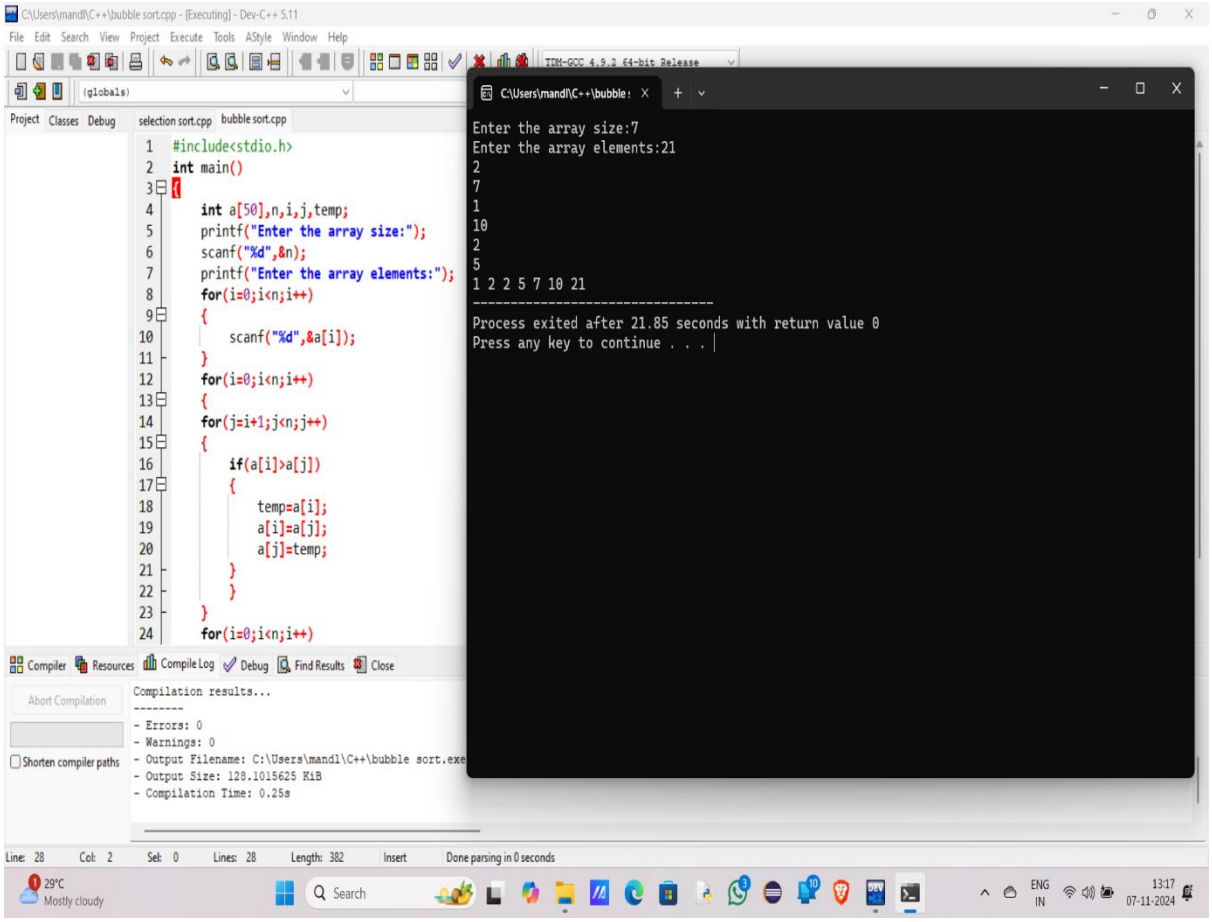
```
{
        printf("%d ",a[i]);
}
return 0;
}
```



8. Bubble sort:

```
#include<stdio.h>
int main()
{
  int a[50],n,i,j,temp;
  printf("Enter the array size:");
  scanf("%d",&n);
  printf("Enter the array elements:");
```

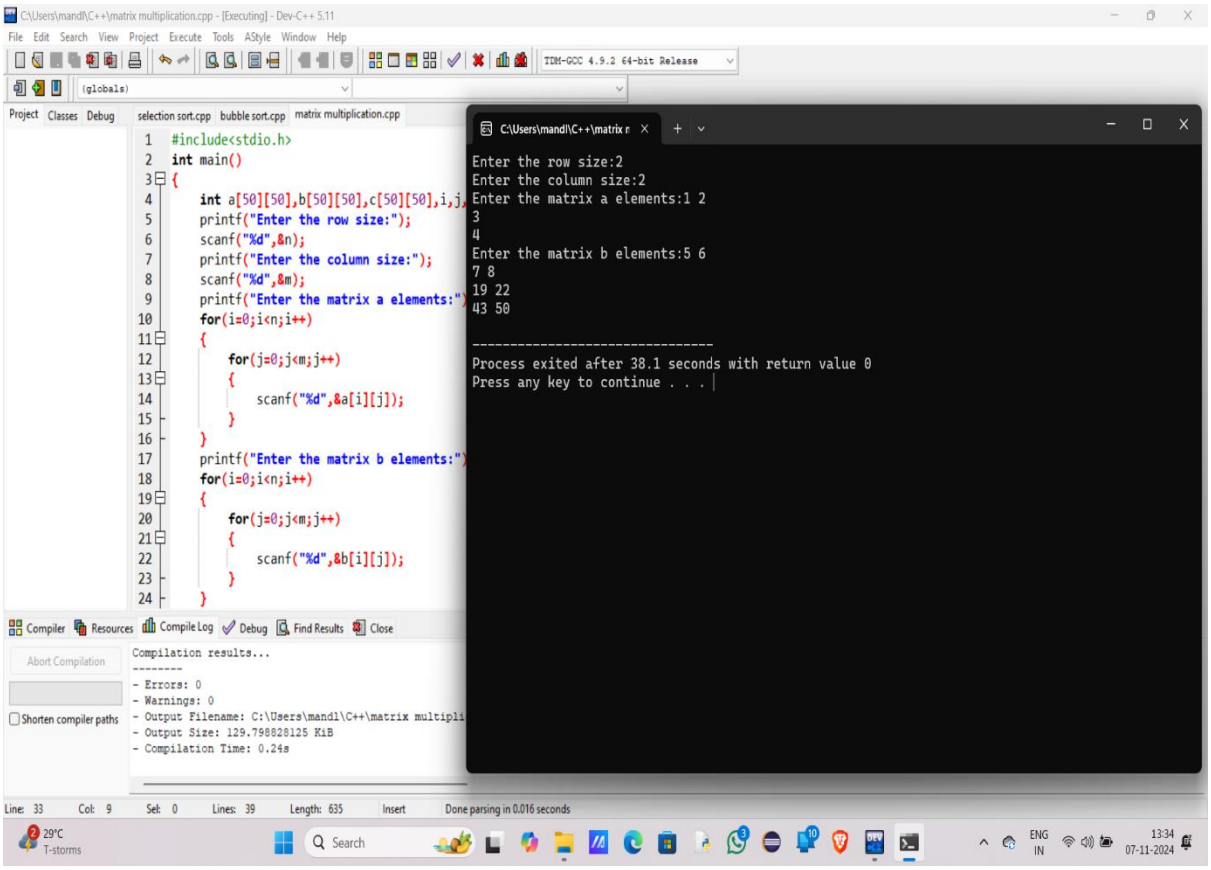
```
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    for(i=0;i<n;i++)
    {
    for(j=i+1;j<n;j++)
    {
        if(a[i]>a[j])
        {
            temp=a[i];
            a[i]=a[j];
            a[j]=temp;
        }
        }
    }
    for(i=0;i<n;i++)
    {
        printf("%d ",a[i]);
    }
}
```



9. Matrix Multiplication:

```
#include<stdio.h>
int main()
{
        int a[50][50],b[50][50],c[50][50],i,j,k,n,m;
        printf("Enter the row size:");
        scanf("%d",&n);
        printf("Enter the column size:");
        scanf("%d",&m);
        printf("Enter the matrix a elements:");
        for(i=0;i<n;i++)
        {
                for(j=0;j<m;j++)
                {
                        scanf("%d",&a[i][j]);
                }
        }
```

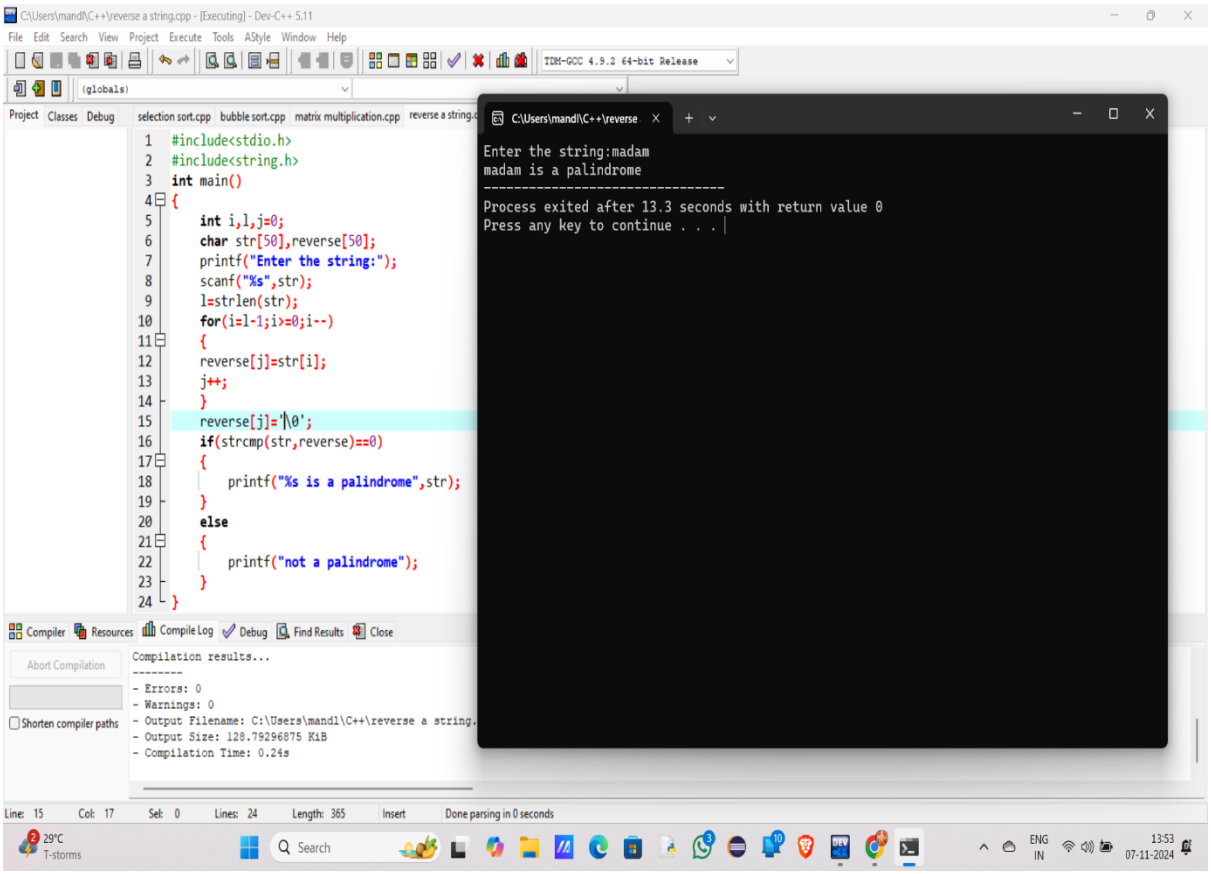
```
        printf("Enter the matrix b elements:");
        for(i=0;i<n;i++)
        {
                for(j=0;j<m;j++)
                {
                        scanf("%d",&b[i][j]);
                }
        }
        for(i=0;i<n;i++)
        {
                for(j=0;j<m;j++)
                {
                c[i][j]=0;
                for(k=0;k<n;k++)
                {
                c[i][j]=c[i][j]+a[i][k]*b[k][j];
           }
                printf("%d ",c[i][j]);
                }
                        printf("\n");
        }
        return 0;
}
```



10.String is a palindrome:

```
#include<stdio.h>
#include<string.h>
int main()
{
        int i,l,j=0;
        char str[50],reverse[50];
        printf("Enter the string:");
        scanf("%s",str);
        l=strlen(str);
        for(i=l-1;i>=0;i--)
        {
        reverse[j]=str[i];
        j++;
        }
        reverse[j]='\0';
        if(strcmp(str,reverse)==0)
        {
```

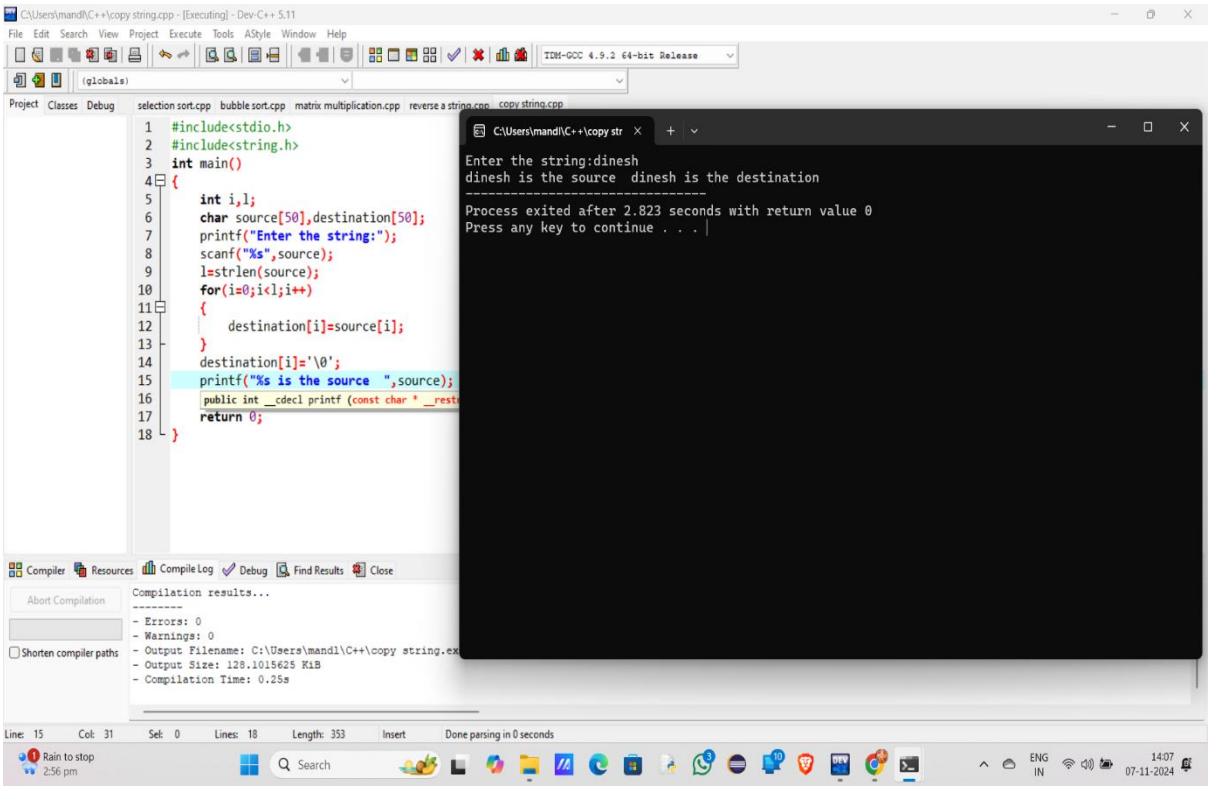
```
            printf("%s is a palindrome",str);
        }
        else
        {
            printf("not a palindrome");
        }
}
```



11.Copy String:

```
#include<stdio.h>
#include<string.h>
int main()
{
        int i,l;
        char source[50],destination[50];
        printf("Enter the string:");
        scanf("%s",source);
        l=strlen(source);
```

```
        for(i=0;i<l;i++)
        {
                destination[i]=source[i];
        }
        destination[i]='\0';
        printf("%s is the source  ",source);
        printf("%s is the destination",destination);
        return 0;
}
```

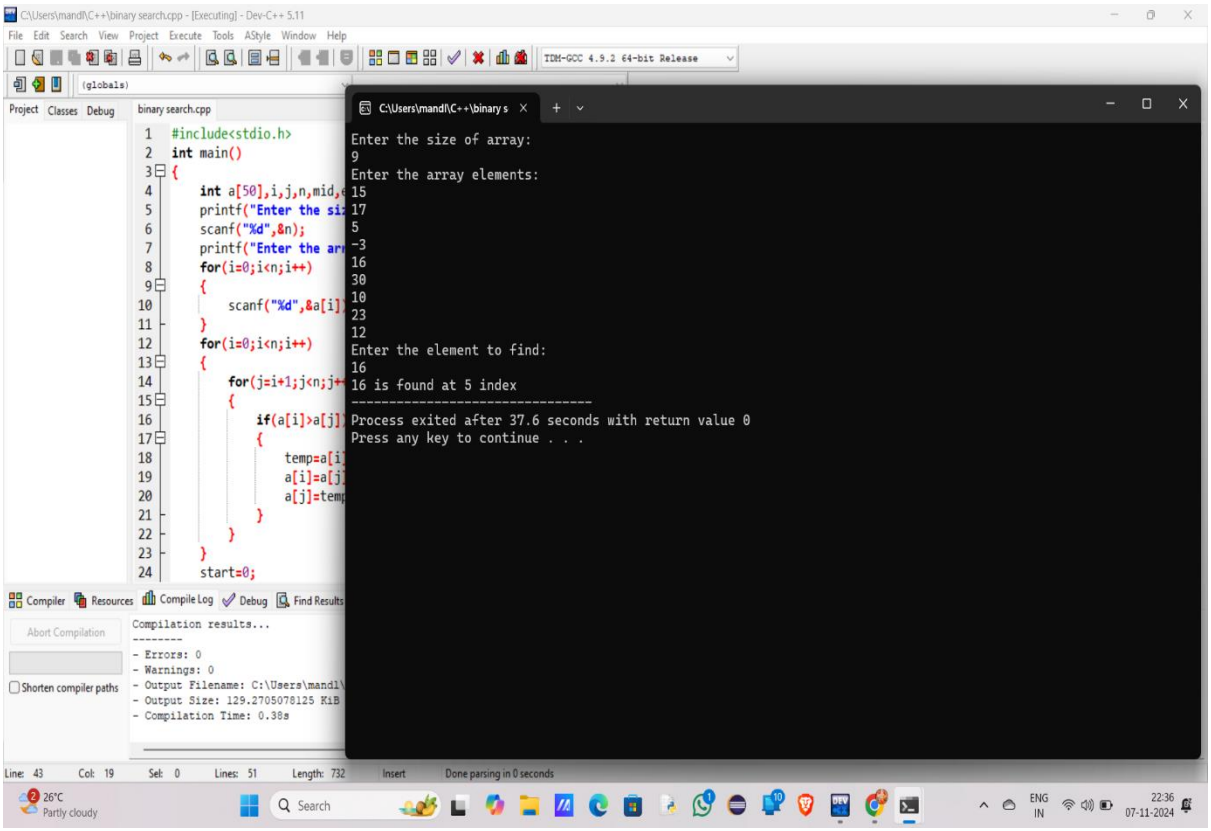


12.Binary Search:

```
#include<stdio.h>
int main()
{
        int a[50],i,j,n,mid,end,start,temp,t,c=0;
        printf("Enter the size of array:\n");
        scanf("%d",&n);
        printf("Enter the array elements:\n");
        for(i=0;i<n;i++)
        {
```

```
        scanf("%d",&a[i]);
}
for(i=0;i<n;i++)
{
        for(j=i+1;j<n;j++)
        {
                if(a[i]>a[j])
                {
                        temp=a[i];
                        a[i]=a[j];
                        a[j]=temp;
                }
        }
}
start=0;
end=n-1;
printf("Enter the element to find:\n");
scanf("%d",&t);
while(start<=end)
{
        mid=start+(end-start)/2;
        if(a[mid]==t)
        {
                c=1;
                printf("%d is found at %d index",t,mid);
                break;
        }
        else if(a[mid]<t)
        {
                start=mid+1;
        }
        else
        {
        end=mid-1;
        }
}
if(c==0)
```

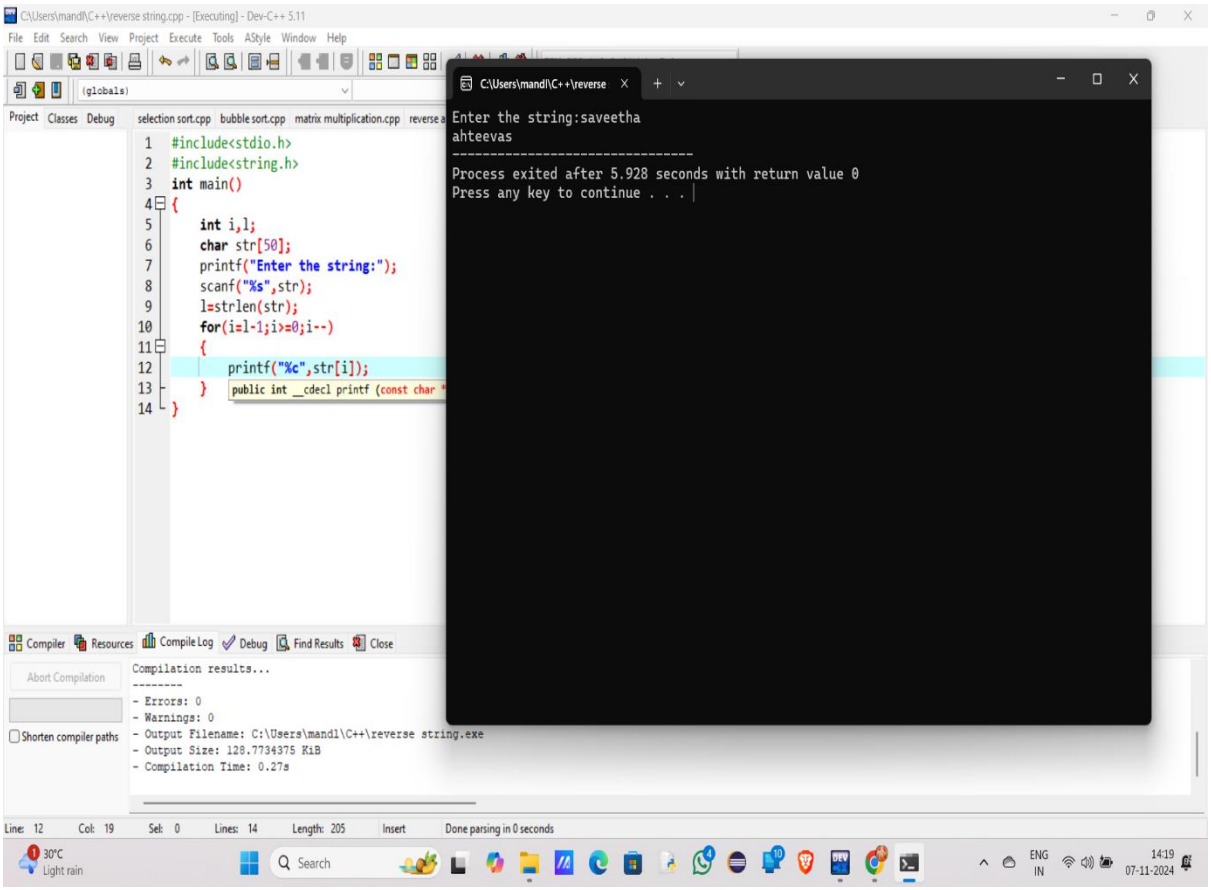
```
        {
                printf("not found");
        }
        return 0;
}
```



13.Reverse a String:

```
#include<string.h>
int main()
{
        int i,l,j=0;
        char str[50],reverse[50];
        printf("Enter the string:");
        scanf("%s",str);
        l=strlen(str);
        for(i=l-1;i>=0;i--)
        {
```

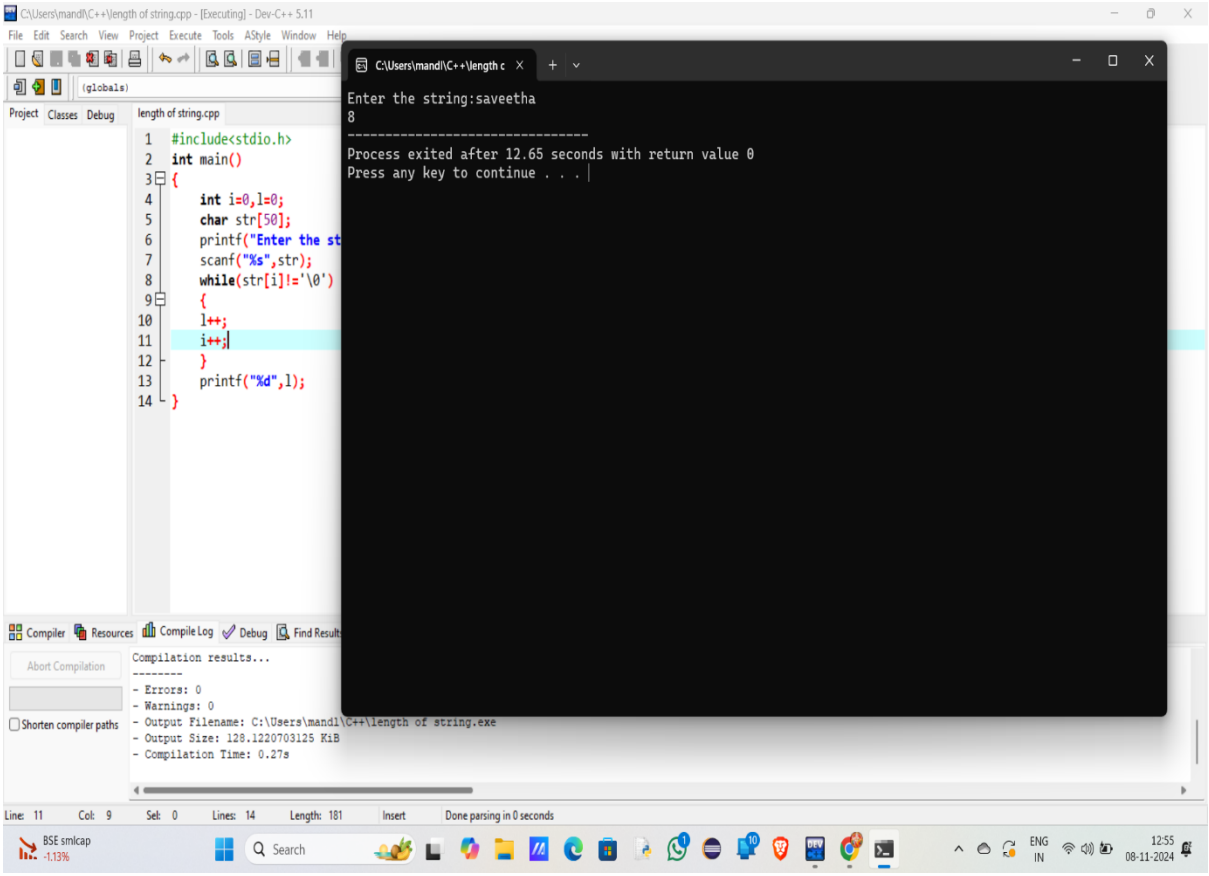
```
        reverse[j]=str[i];
        j++;
        }
        reverse[j]='\0';
        printf("%s\n",reverse);
}
```



14.Length of String:

```
#include<stdio.h>
int main()
{
        int i=0,l=0;
        char str[50];
        printf("Enter the string:");
        scanf("%s",str);
        while(str[i]!='\0')
        {
```

```
        l++;
        i++;
        }
        printf("%d",l);
}
```

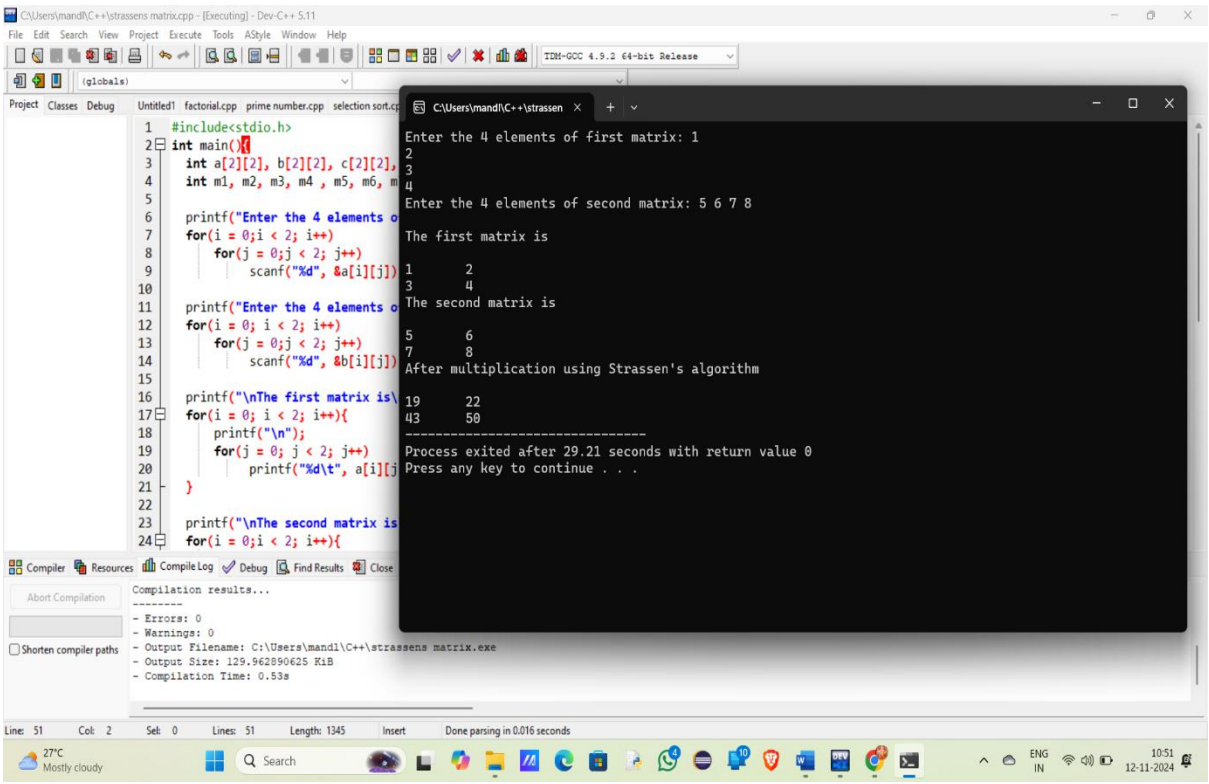


15.Strassen's Matrix:

```
#include<stdio.h>
int main(){
 int a[2][2], b[2][2], c[2][2], i, j;
 int m1, m2, m3, m4 , m5, m6, m7;

 printf("Enter the 4 elements of first matrix: ");
 for(i = 0;i < 2; i++)
    for(j = 0;j < 2; j++)
        scanf("%d", &a[i][j]);
```

```
printf("Enter the 4 elements of second matrix: ");
for(i = 0; i < 2; i++)
   for(j = 0;j < 2; j++)
       scanf("%d", &b[i][j]);

printf("\nThe first matrix is\n");
for(i = 0; i < 2; i++){
   printf("\n");
   for(j = 0; j < 2; j++)
       printf("%d\t", a[i][j]);
}

printf("\nThe second matrix is\n");
for(i = 0;i < 2; i++){
   printf("\n");
   for(j = 0;j < 2; j++)
       printf("%d\t", b[i][j]);
}

m1= (a[0][0] + a[1][1]) * (b[0][0] + b[1][1]);
m2= (a[1][0] + a[1][1]) * b[0][0];
m3= a[0][0] * (b[0][1] - b[1][1]);
m4= a[1][1] * (b[1][0] - b[0][0]);
m5= (a[0][0] + a[0][1]) * b[1][1];
m6= (a[1][0] - a[0][0]) * (b[0][0]+b[0][1]);
m7= (a[0][1] - a[1][1]) * (b[1][0]+b[1][1]);

c[0][0] = m1 + m4- m5 + m7;
c[0][1] = m3 + m5;
c[1][0] = m2 + m4;
c[1][1] = m1 - m2 + m3 + m6;

 printf("\nAfter multiplication using Strassen's algorithm \n");
 for(i = 0; i < 2 ; i++){
   printf("\n");
   for(j = 0;j < 2; j++)
       printf("%d\t", c[i][j]);
```

```
    }

    return 0;
}
```

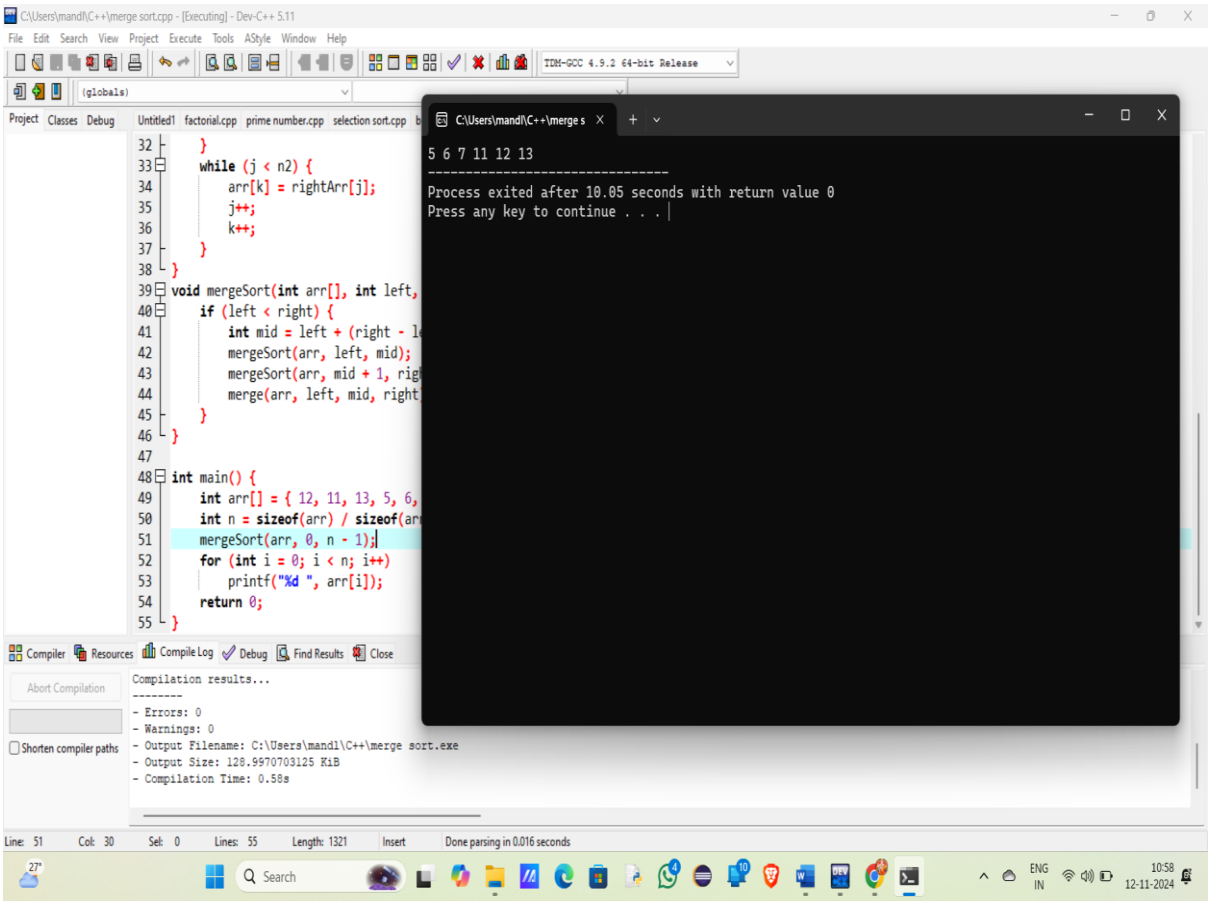


16.Merge Sort:

```
#include <stdio.h>
#include <stdlib.h>
void merge(int arr[], int left, int mid, int right) {
    int i, j, k;
    int n1 = mid - left + 1;
    int n2 = right - mid;
    int leftArr[n1], rightArr[n2];
    for (i = 0; i < n1; i++)
        leftArr[i] = arr[left + i];
    for (j = 0; j < n2; j++)
        rightArr[j] = arr[mid + 1 + j];
    i = 0;
    j = 0;
```

```
    k = left;
    while (i < n1 && j < n2) {
        if (leftArr[i] <= rightArr[j]) {
            arr[k] = leftArr[i];
            i++;
        }
        else {
            arr[k] = rightArr[j];
            j++;
        }
        k++;
    }

    // Copy the remaining elements of leftArr[], if any
    while (i < n1) {
        arr[k] = leftArr[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = rightArr[j];
        j++;
        k++;
    }
}
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

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

```
    mergeSort(arr, 0, n - 1);
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

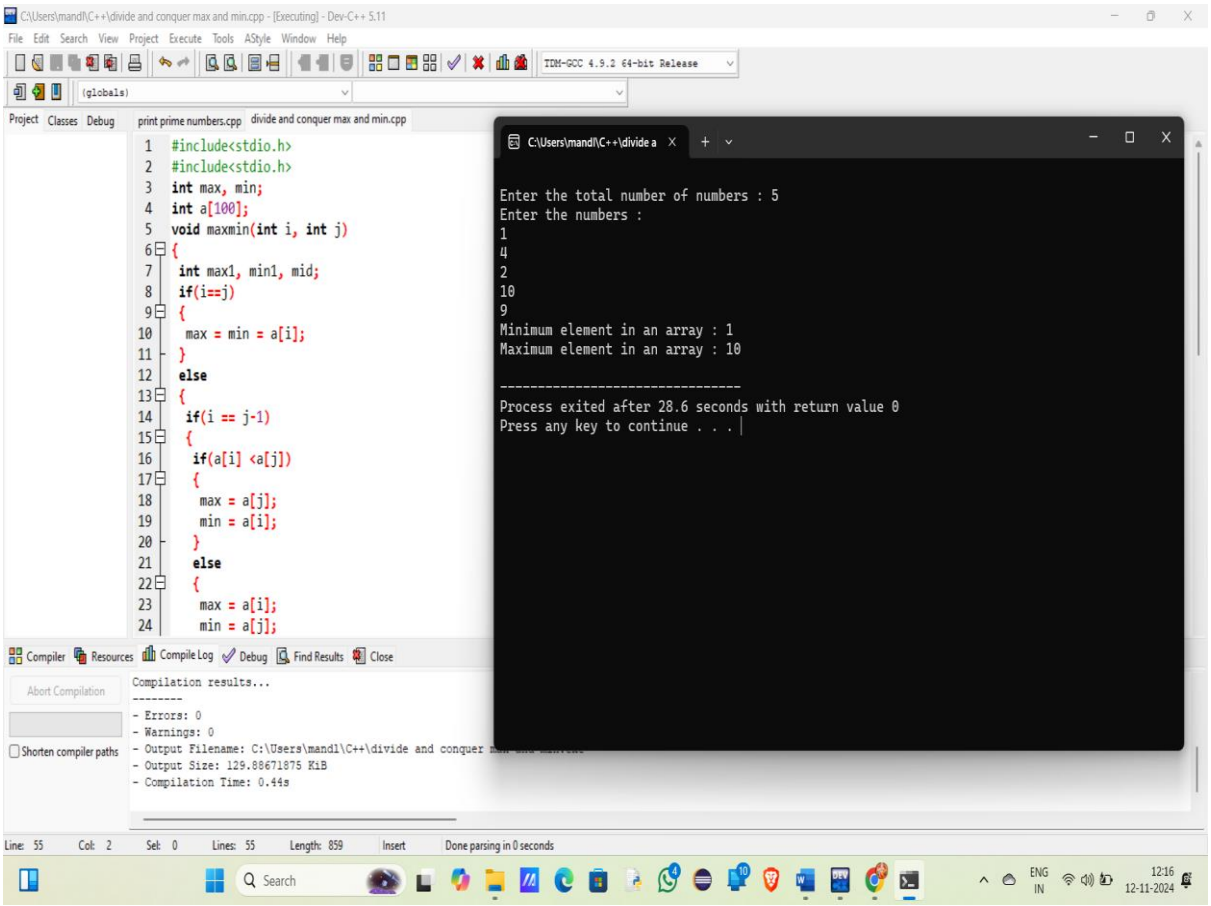


17.Divide and Conquer Strategy for Finding MAX and MIN Values:

```
#include<stdio.h>
#include<stdio.h>
int max, min;
int a[100];
void maxmin(int i, int j)
{
 int max1, min1, mid;
 if(i==j)
```

```
{
 max = min = a[i];
}
else
{
 if(i == j-1)
 {
  if(a[i] <a[j])
  {
   max = a[j];
   min = a[i];
  }
  else
  {
   max = a[i];
   min = a[j];
  }
 }
 else
 {
  mid = (i+j)/2;
  maxmin(i, mid);
  max1 = max; min1 = min;
  maxmin(mid+1, j);
  if(max <max1)
   max = max1;
  if(min > min1)
   min = min1;
 }
}
}
int main ()
{
 int i, num;
 printf ("\nEnter the total number of numbers : ");
 scanf ("%d",&num);
 printf ("Enter the numbers : \n");
```

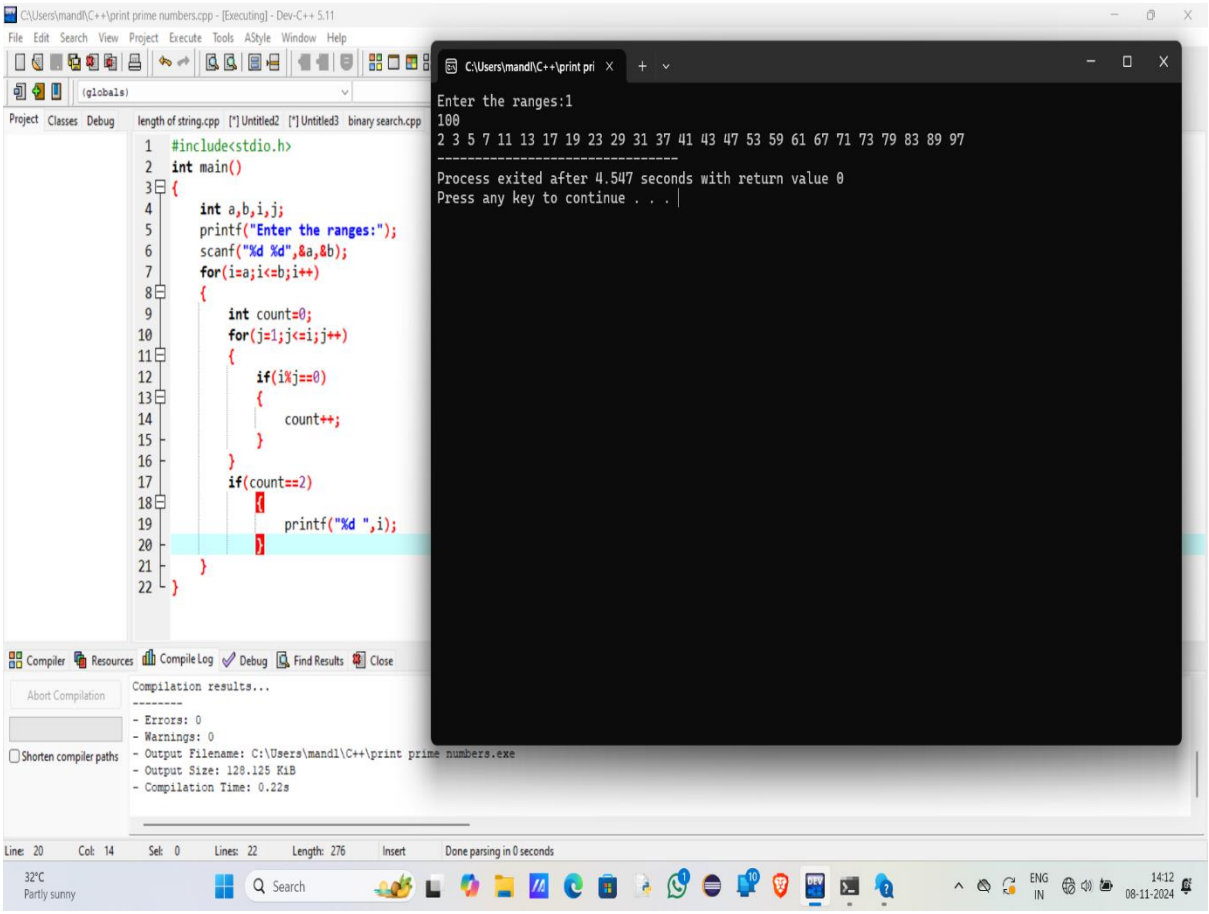
```
for (i=1;i<=num;i++)
 scanf ("%d",&a[i]);

max = a[0];
min = a[0];
maxmin(1, num);
printf ("Minimum element in an array : %d\n", min);
printf ("Maximum element in an array : %d\n", max);
return 0;
}
```



18.Prime numbers between Range:

```
#include<stdio.h>
int main()
{
        int a,b,i,j;
```

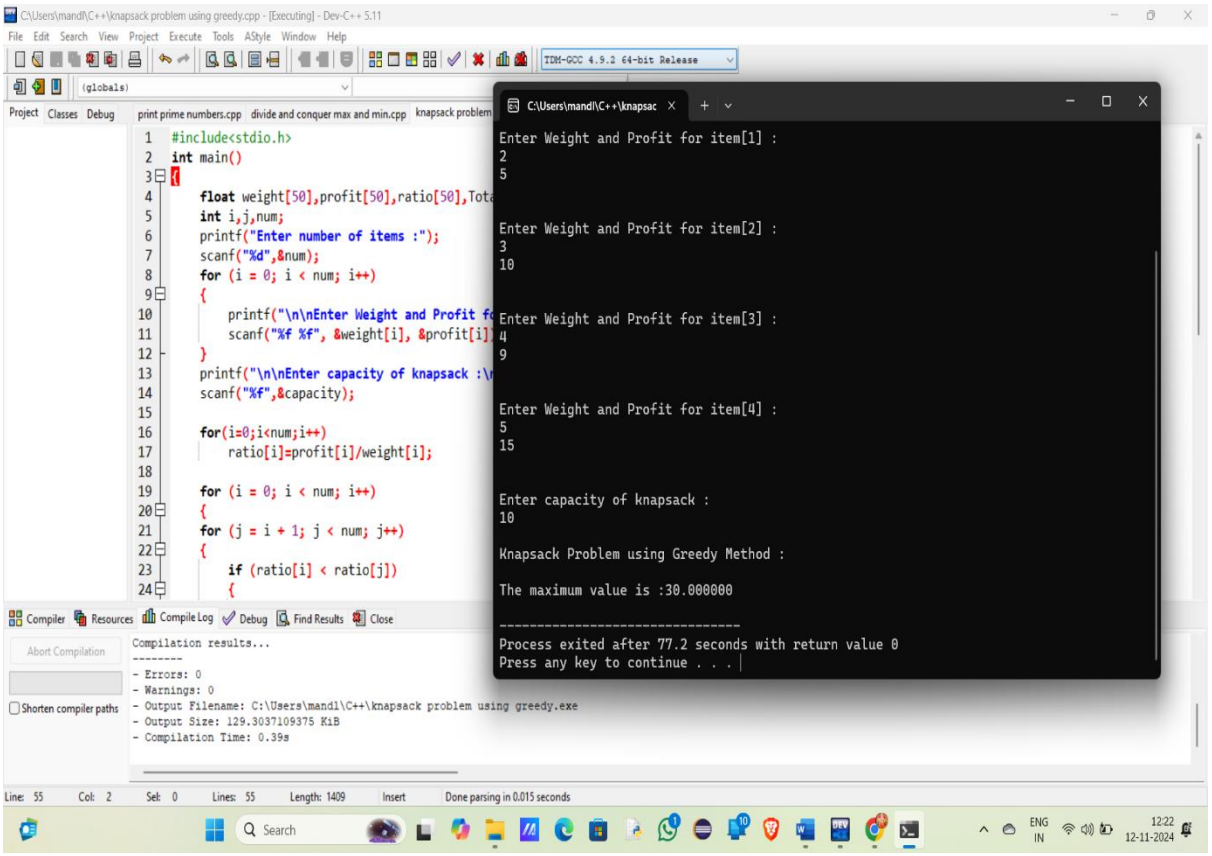
```
        printf("Enter the ranges:");
        scanf("%d %d",&a,&b);
        for(i=a;i<=b;i++)
        {
                int count=0;
                for(j=1;j<=i;j++)
                {
                        if(i%j==0)
                        {
                                count++;
                   }
                }
                if(count==2)
                        {
                                printf("%d ",i);
                        }
        }
}
```



19.Knapsack Problem using Greedy Technique:

```
#include<stdio.h>
int main()
{
   float weight[50],profit[50],ratio[50],Totalvalue,temp,capacity,amount;
   int i,j,num;
   printf("Enter number of items :");
   scanf("%d",&num);
   for (i = 0; i < num; i++)
   {
      printf("\n\nEnter Weight and Profit for item[%d] :\n",i);
      scanf("%f %f", &weight[i], &profit[i]);
   }
   printf("\n\nEnter capacity of knapsack :\n");
   scanf("%f",&capacity);

   for(i=0;i<num;i++)
      ratio[i]=profit[i]/weight[i];

   for (i = 0; i < num; i++)
   {
        for (j = i + 1; j < num; j++)
   {
                if (ratio[i] < ratio[j])
      {
         temp = ratio[j];
         ratio[j] = ratio[i];
         ratio[i] = temp;

         temp = weight[j];
         weight[j] = weight[i];
         weight[i] = temp;

         temp = profit[j];
```

```
            profit[j] = profit[i];
            profit[i] = temp;
          }
            }
            }

    printf("\nKnapsack Problem using Greedy Method :\n");
    for (i = 0; i < num; i++)
    {
     if (weight[i] > capacity)
        break;
      else
     {
        Totalvalue = Totalvalue + profit[i];
                capacity = capacity - weight[i];
      }
    }
      if (i < num)
      Totalvalue = Totalvalue + (ratio[i]*capacity);
    printf("\nThe maximum value is :%f\n",Totalvalue);
    return 0;
}
```



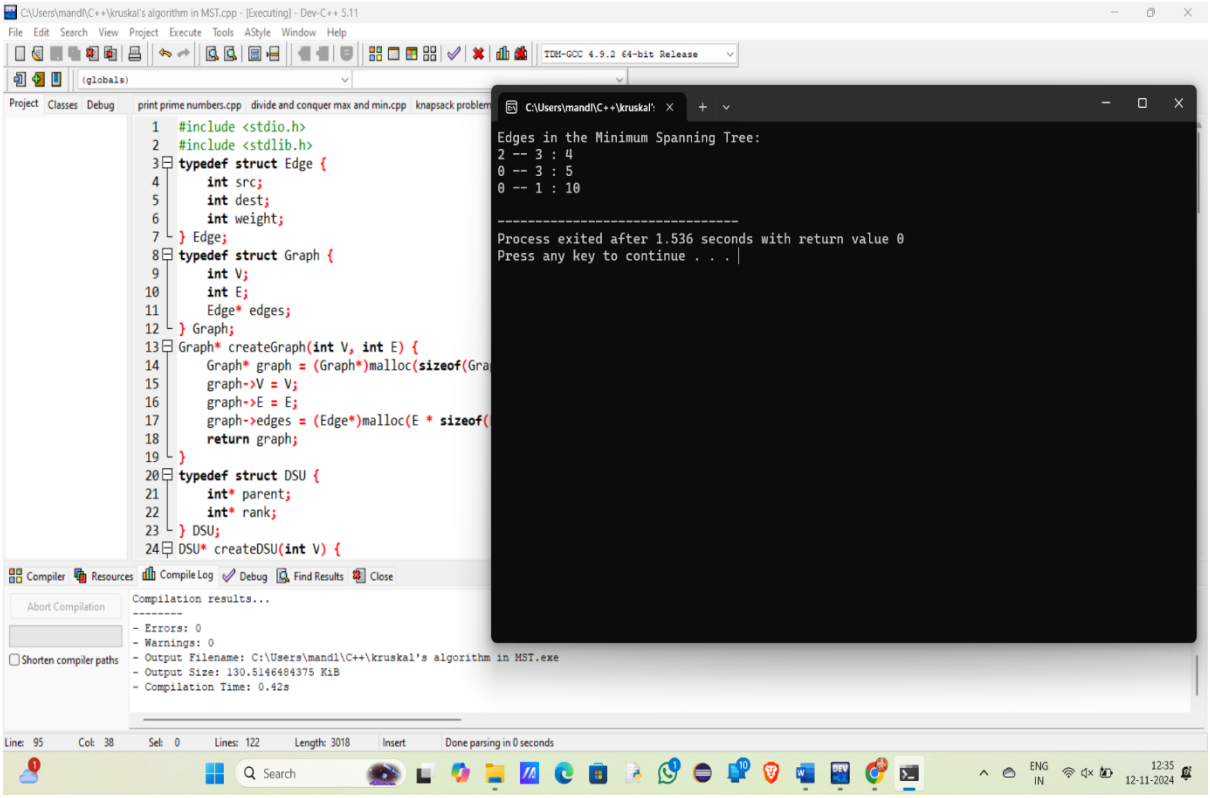
20.MST (minimum spanning tree) using Kruskal's Algorithm:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Edge {
    int src;
    int dest;
    int weight;
} Edge;
typedef struct Graph {
    int V;
    int E;
    Edge* edges;
} Graph;
Graph* createGraph(int V, int E) {
    Graph* graph = (Graph*)malloc(sizeof(Graph));
    graph->V = V;
    graph->E = E;
```

```
    graph->edges = (Edge*)malloc(E * sizeof(Edge));
    return graph;
}
typedef struct DSU {
    int* parent;
    int* rank;
} DSU;
DSU* createDSU(int V) {
    DSU* dsu = (DSU*)malloc(sizeof(DSU));
    dsu->parent = (int*)malloc(V * sizeof(int));
    dsu->rank = (int*)malloc(V * sizeof(int));
    for (int i = 0; i < V; i++) {
        dsu->parent[i] = i;
        dsu->rank[i] = 0;
    }
    return dsu;
}
int find(DSU* dsu, int x) {
    if (dsu->parent[x] != x) {
        dsu->parent[x] = find(dsu, dsu->parent[x]);
    }
    return dsu->parent[x];
}
void unionSets(DSU* dsu, int x, int y) {
    int rootX = find(dsu, x);
    int rootY = find(dsu, y);

    if (rootX != rootY) {
        if (dsu->rank[rootX] < dsu->rank[rootY]) {
            dsu->parent[rootX] = rootY;
        } else if (dsu->rank[rootX] > dsu->rank[rootY]) {
            dsu->parent[rootY] = rootX;
        } else {
            dsu->parent[rootY] = rootX;
            dsu->rank[rootX]++;
        }
    }
```

```
}
int compareEdges(const void* a, const void* b) {
    Edge* edgeA = (Edge*)a;
    Edge* edgeB = (Edge*)b;
    return edgeA->weight - edgeB->weight;
}
void kruskalMST(Graph* graph) {
    qsort(graph->edges, graph->E, sizeof(Edge), compareEdges);

    DSU* dsu = createDSU(graph->V);

    Edge* result = (Edge*)malloc((graph->V - 1) * sizeof(Edge));
    int edgeCount = 0;
    int i = 0;

    while (edgeCount < graph->V - 1 && i < graph->E) {
        Edge nextEdge = graph->edges[i++];
        int srcRoot = find(dsu, nextEdge.src);
        int destRoot = find(dsu, nextEdge.dest);

        if (srcRoot != destRoot) {
            result[edgeCount++] = nextEdge;
            unionSets(dsu, srcRoot, destRoot);
        }
    }

    printf("Edges in the Minimum Spanning Tree:\n");
    for (int j = 0; j < edgeCount; j++) {
        printf("%d -- %d : %d\n", result[j].src, result[j].dest, result[j].weight);
    }

    free(result);
    free(dsu->parent);
    free(dsu->rank);
    free(dsu);
}
```

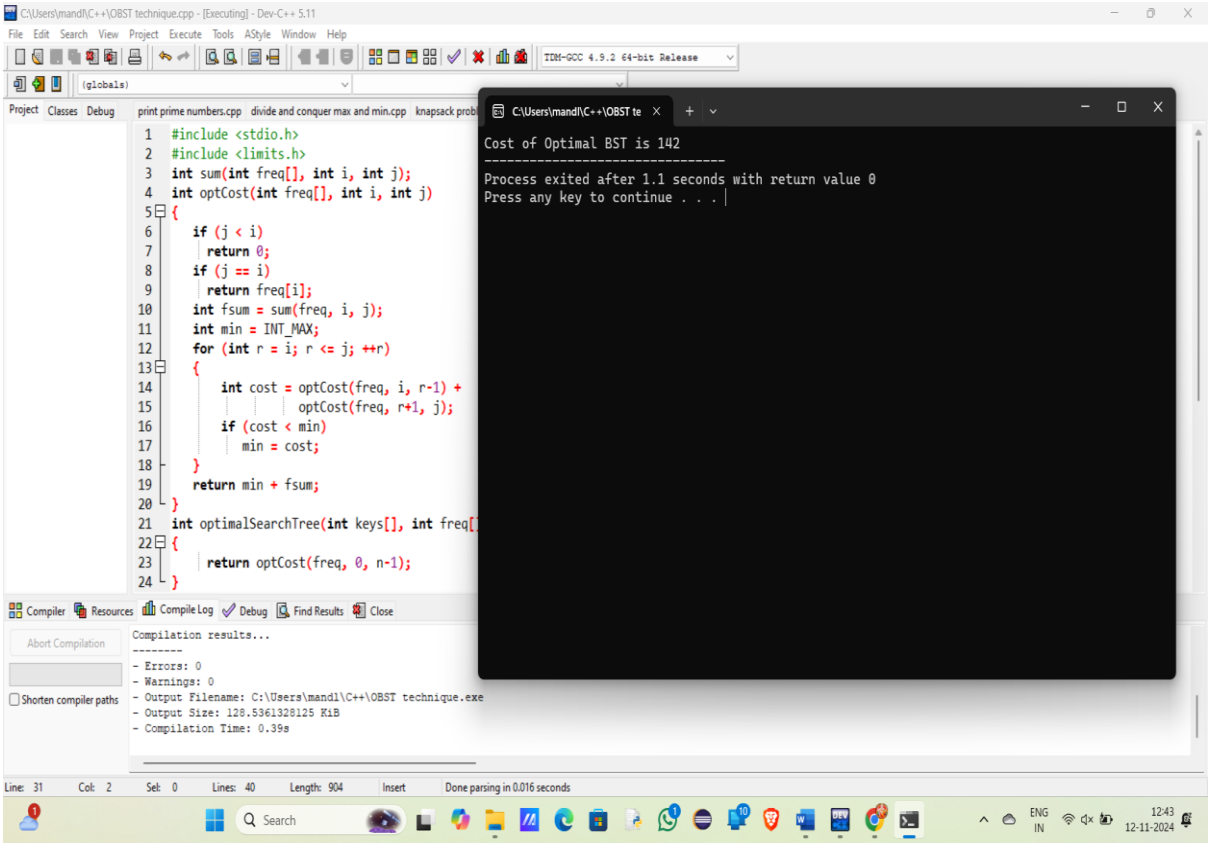
```
int main() {
    int V = 4;
    int E = 5;

    Graph* graph = createGraph(V, E);
    graph->edges[0].src = 0;
    graph->edges[0].dest = 1;
    graph->edges[0].weight = 10;

    graph->edges[1].src = 0;
    graph->edges[1].dest = 2;
    graph->edges[1].weight = 6;

    graph->edges[2].src = 0;
    graph->edges[2].dest = 3;
    graph->edges[2].weight = 5;

    graph->edges[3].src = 1;
    graph->edges[3].dest = 3;
    graph->edges[3].weight = 15;

    graph->edges[4].src = 2;
    graph->edges[4].dest = 3;
    graph->edges[4].weight = 4;

    kruskalMST(graph);

    free(graph->edges);
    free(graph);

    return 0;
}
```



21.OBST (Optimal Binary Search Tree) Technique:

```
#include <stdio.h>
#include <limits.h>
int sum(int freq[], int i, int j);
int optCost(int freq[], int i, int j)
{
  if (j < i)
    return 0;
  if (j == i)
    return freq[i];
  int fsum = sum(freq, i, j);
  int min = INT_MAX;
  for (int r = i; r <= j; ++r)
  {
     int cost = optCost(freq, i, r-1) +
              optCost(freq, r+1, j);
     if (cost < min)
       min = cost;
```

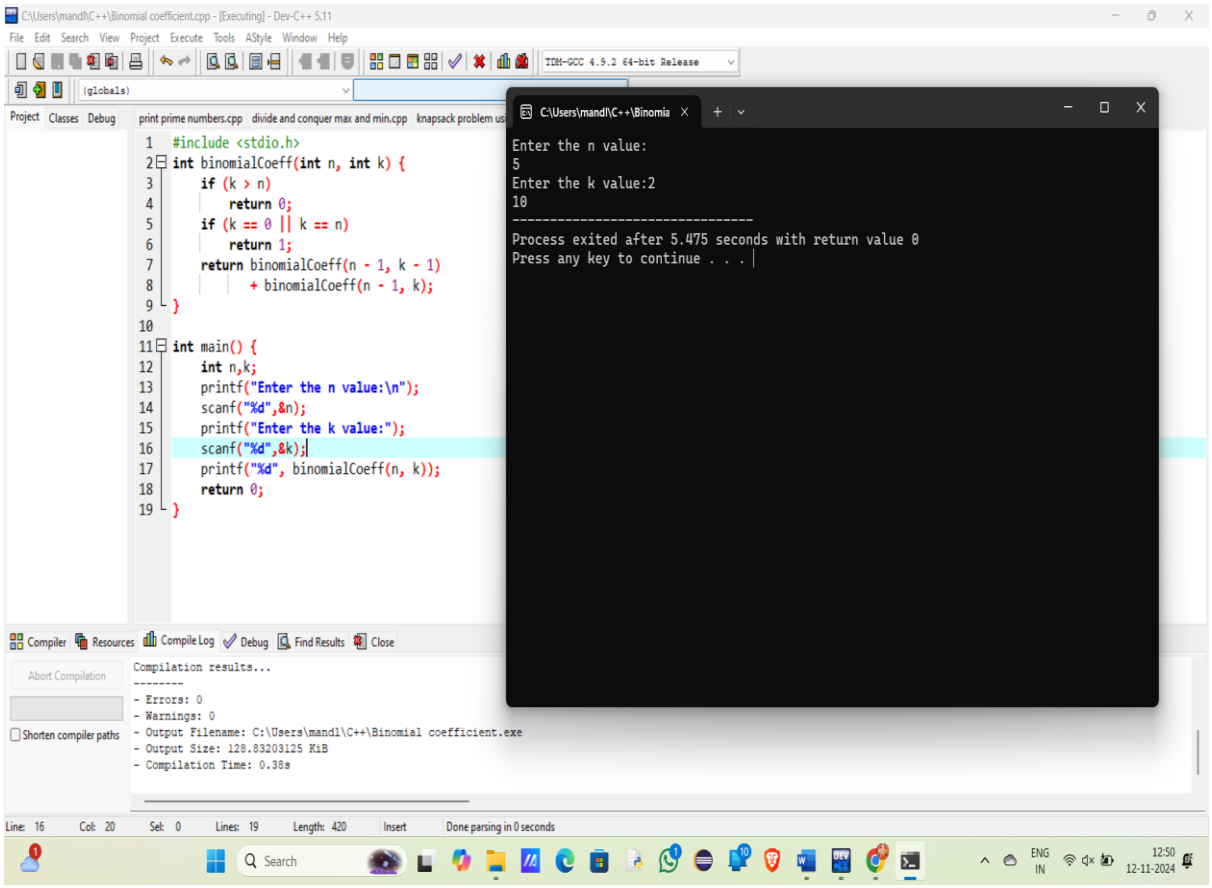
```
    }
    return min + fsum;
}
int optimalSearchTree(int keys[], int freq[], int n)
{
     return optCost(freq, 0, n-1);
}
int sum(int freq[], int i, int j)
{
    int s = 0;
    for (int k = i; k <=j; k++)
       s += freq[k];
    return s;
}
int main()
{
    int keys[] = {10, 12, 20};
    int freq[] = {34, 8, 50};
    int n = sizeof(keys)/sizeof(keys[0]);
    printf("Cost of Optimal BST is %d ",
              optimalSearchTree(keys, freq, n));
    return 0;
}
```



22.Binary Coefficient:

```
#include <stdio.h>
int binomialCoeff(int n, int k) {
   if (k > n)
      return 0;
   if (k == 0 || k == n)
      return 1;
   return binomialCoeff(n - 1, k - 1)
         + binomialCoeff(n - 1, k);
}

int main() {
   int n,k;
   printf("Enter the n value:\n");
   scanf("%d",&n);
   printf("Enter the k value:");
```

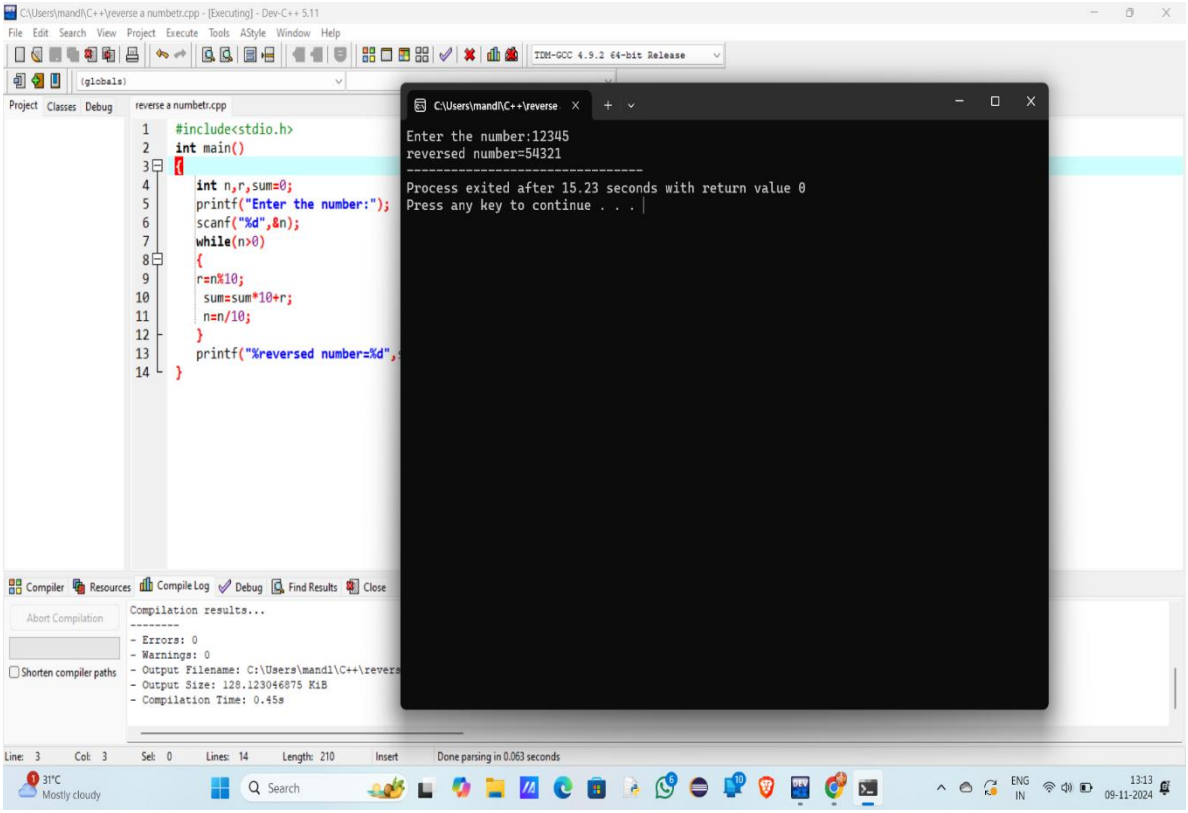
```
    scanf("%d",&k);
    printf("%d", binomialCoeff(n, k));
    return 0;
}
```



23.Reverse of a Number:

```
#include<stdio.h>
 int main()
 {
 int n,r,sum=0;
 printf("Enter the number:");
 scanf("%d",&n);
 while(n>0)
 {
 r=n%10;
  sum=sum*10+r;
```

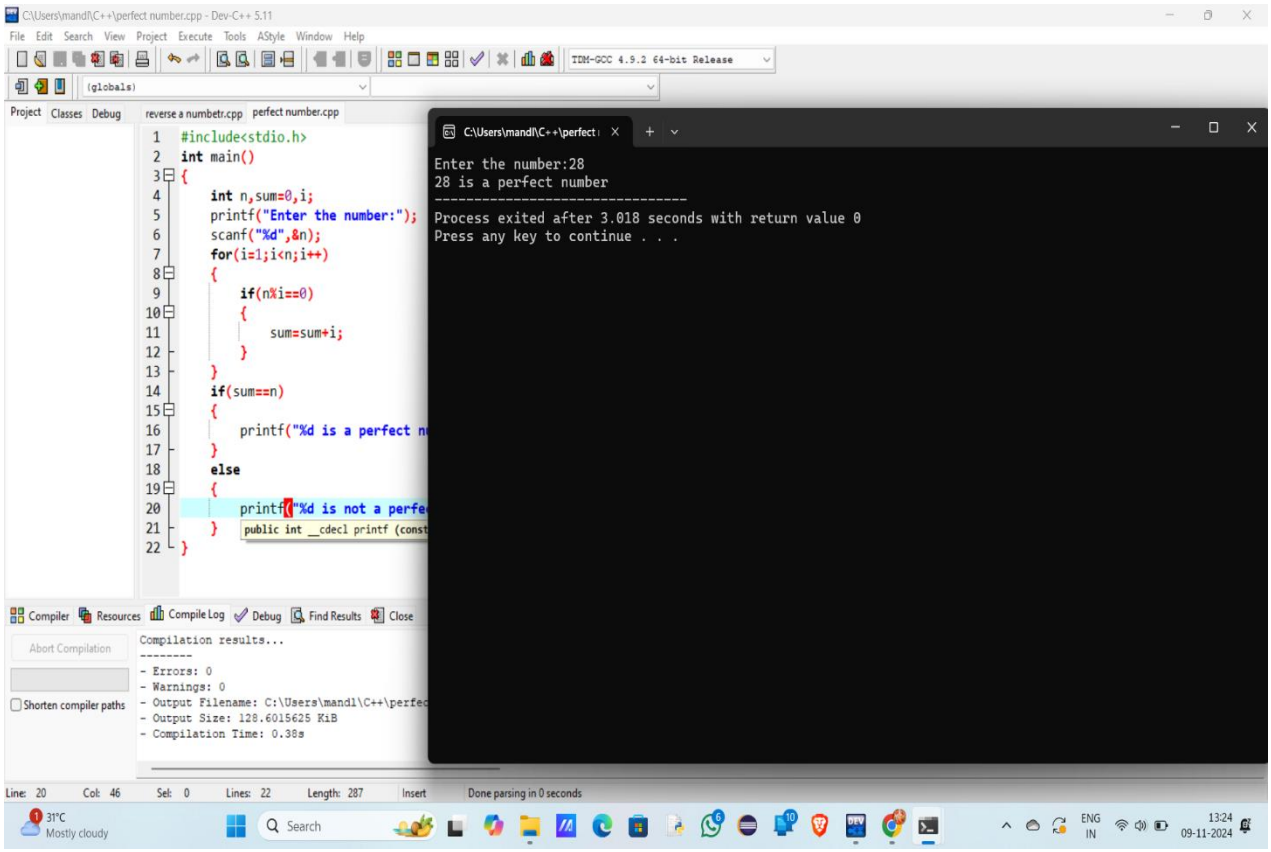
```
 n=n/10;
 }
 printf("%reversed number=%d",sum);
 }
```



24.Perfect Number:

```
#include<stdio.h>
int main()
{
 int n,sum=0,i;
 printf("Enter the number:");
 scanf("%d",&n);
 for(i=1;i<n;i++)
 {
        if(n%i==0)
        {
                sum=sum+i;
        }
 }
```

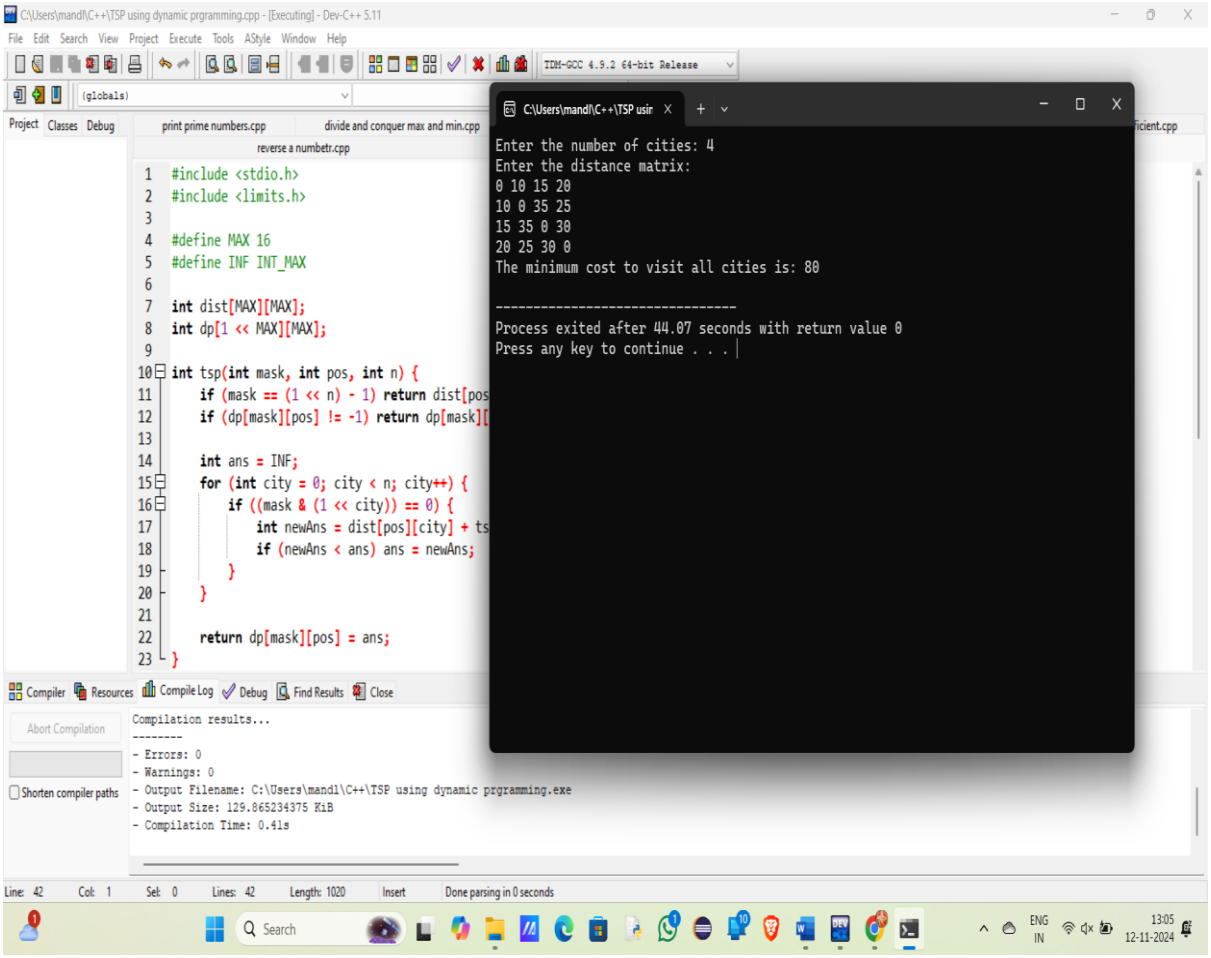
```
if(sum==n)
{
        printf("%d is a perfect number",n);
}
else
{
        printf("%d is not a perfect number",n);
}
}
```



25.TSP (Travelling Sales Man Problem) using Dynamic programming:

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

#define MAX 16
#define INF INT_MAX
```

```
int dist[MAX][MAX];
int dp[1 << MAX][MAX];

int tsp(int mask, int pos, int n) {
   if (mask == (1 << n) - 1) return dist[pos][0];
   if (dp[mask][pos] != -1) return dp[mask][pos];

   int ans = INF;
   for (int city = 0; city < n; city++) {
      if ((mask & (1 << city)) == 0) {
         int newAns = dist[pos][city] + tsp(mask | (1 << city), city, n);
         if (newAns < ans) ans = newAns;
      }
   }

   return dp[mask][pos] = ans;
}

int main() {
   int n;
   printf("Enter the number of cities: ");
   scanf("%d", &n);

   printf("Enter the distance matrix:\n");
   for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
         scanf("%d", &dist[i][j]);

   for (int i = 0; i < (1 << n); i++)
      for (int j = 0; j < n; j++)
         dp[i][j] = -1;

   printf("The minimum cost to visit all cities is: %d\n", tsp(1, 0, n));
   return 0;
```

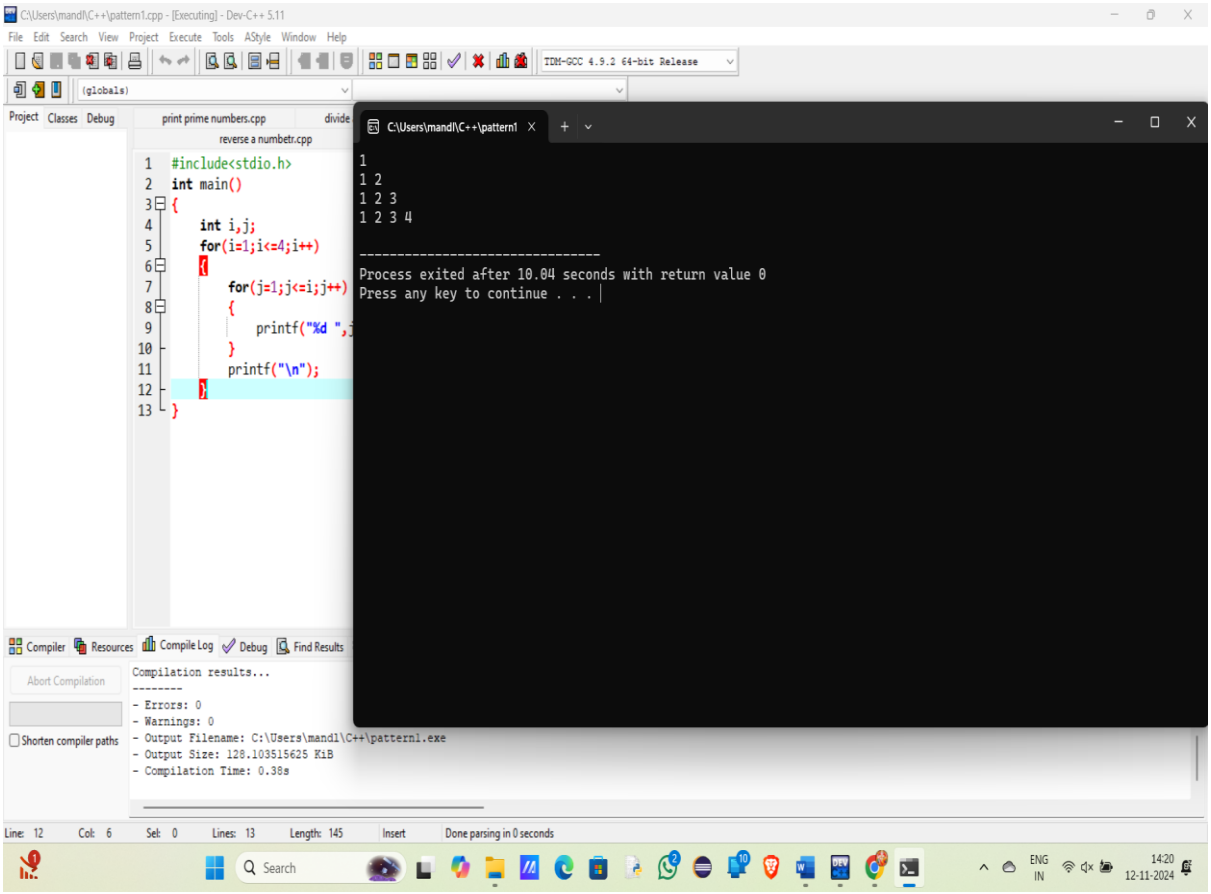


26. Print the pattern 1

```
1 2
1 2 3
1 2 3 4
```

```
#include<stdio.h>
int main()
{
 int i,j;
 for(i=1;i<=4;i++)
 {
        for(j=1;j<=i;j++)
        {
                printf("%d ",j);
        }
        printf("\n");
 }
```

```
}
```

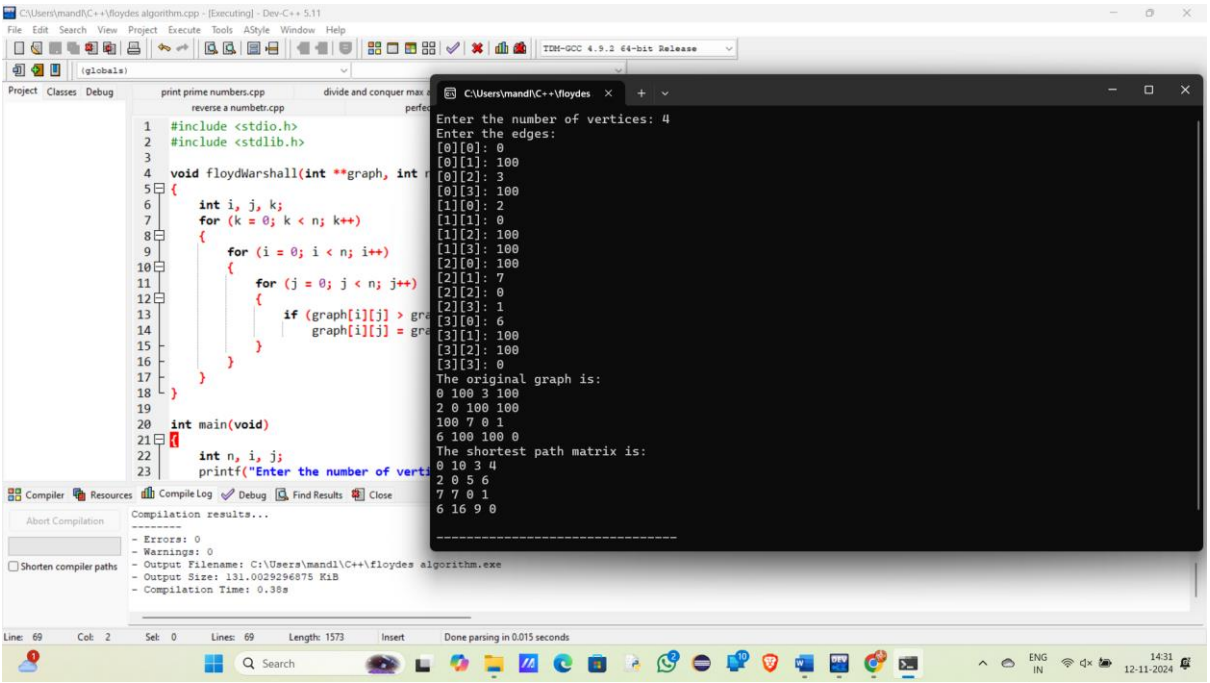


27.Floyde's Warshell Algorithm:

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

void floydWarshall(int **graph, int n)
{
   int i, j, k;
   for (k = 0; k < n; k++)
   {
      for (i = 0; i < n; i++)
      {
         for (j = 0; j < n; j++)
         {
            if (graph[i][j] > graph[i][k] + graph[k][j])
               graph[i][j] = graph[i][k] + graph[k][j];
```

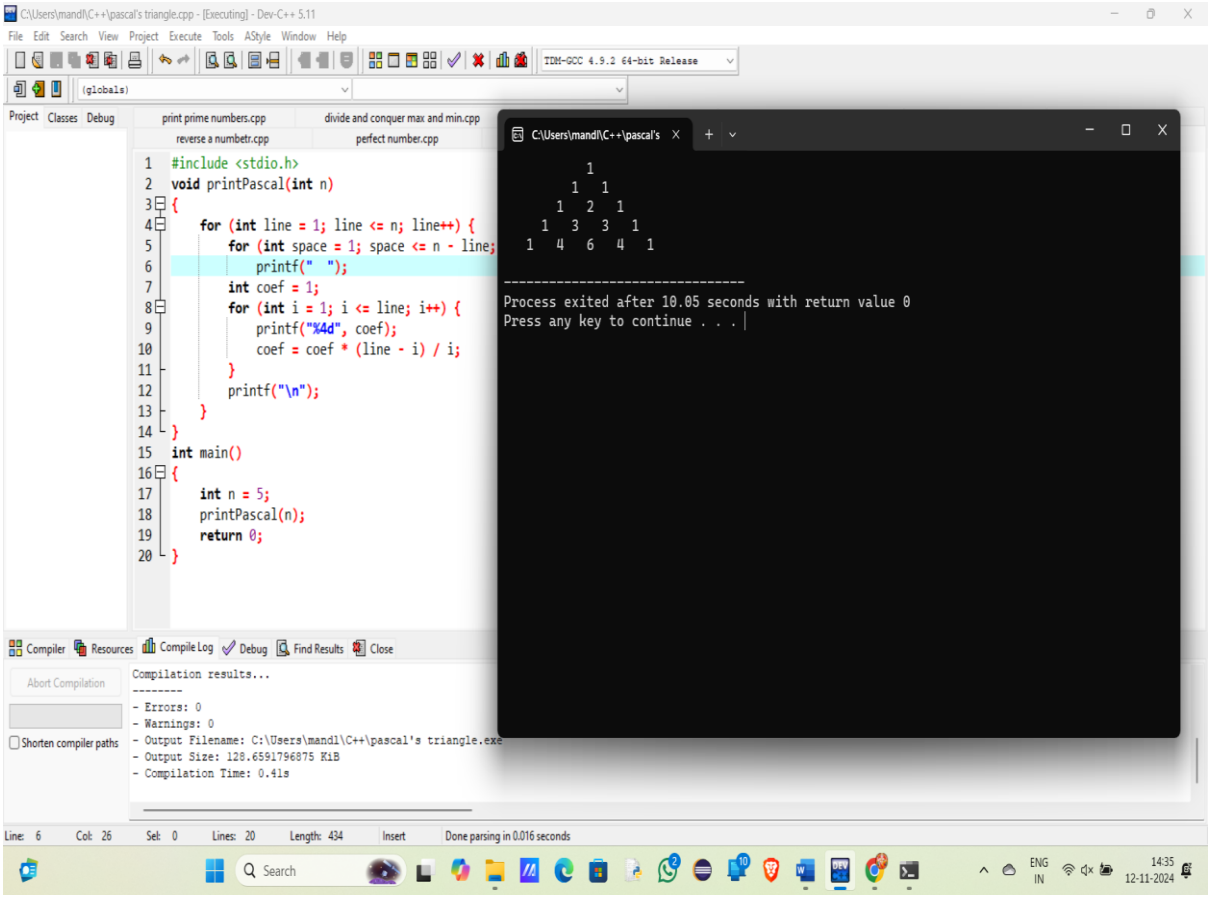
```
            }
        }
    }
}

int main(void)
{
    int n, i, j;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    int **graph = (int **)malloc((long unsigned) n * sizeof(int *));
    for (i = 0; i < n; i++)
    {
        graph[i] = (int *)malloc((long unsigned) n * sizeof(int));
    }
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            if (i == j)
                graph[i][j] = 0;
            else
                graph[i][j] = 100;
        }
    }
    printf("Enter the edges: \n");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            printf("[%d][%d]: ", i, j);
            scanf("%d", &graph[i][j]);
        }
    }
    printf("The original graph is:\n");
    for (i = 0; i < n; i++)
    {
```

```
        for (j = 0; j < n; j++)
        {
            printf("%d ", graph[i][j]);
        }
        printf("\n");
    }
    floydWarshall(graph, n);
    printf("The shortest path matrix is:\n");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            printf("%d ", graph[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```



28.Pascal’s Triangle:

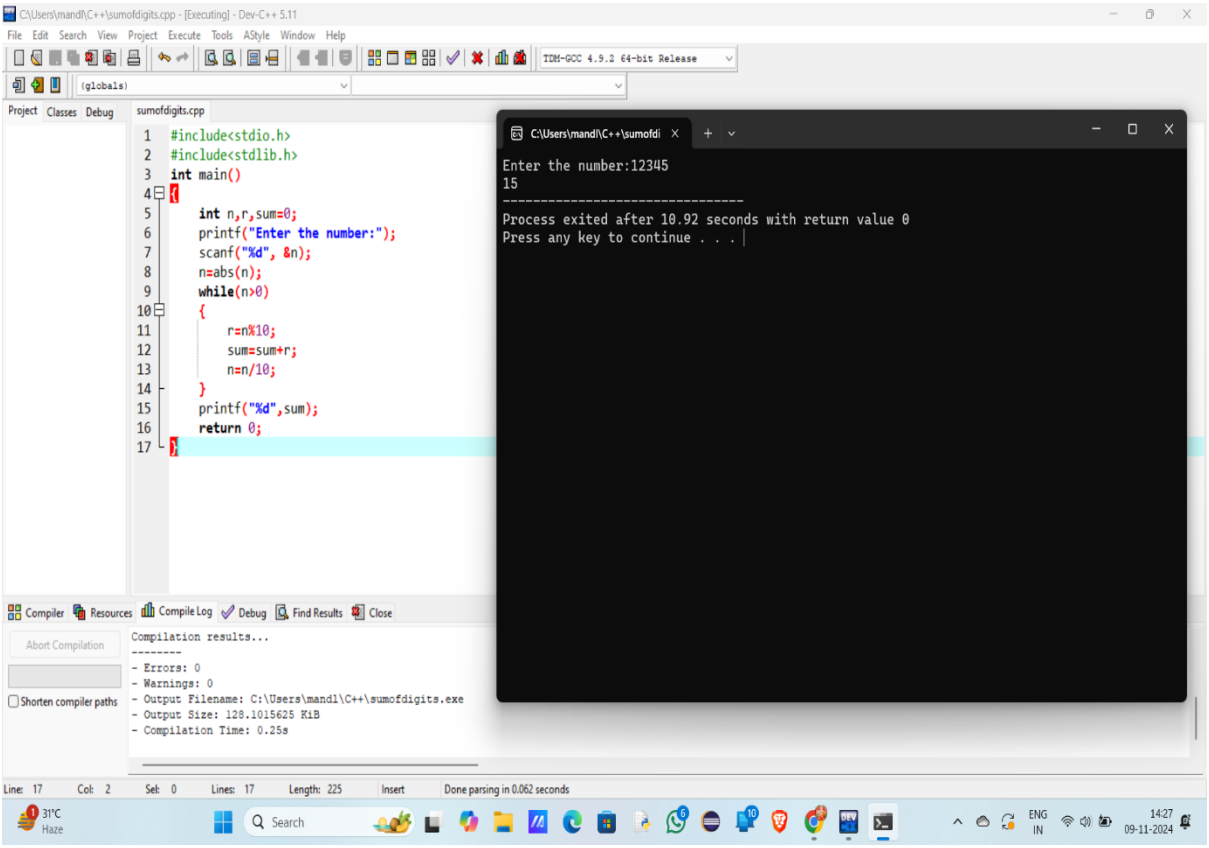
```
#include <stdio.h>
void printPascal(int n)
{
   for (int line = 1; line <= n; line++) {
      for (int space = 1; space <= n - line; space++)
         printf("  ");
      int coef = 1;
      for (int i = 1; i <= line; i++) {
         printf("%4d", coef);
         coef = coef * (line - i) / i;
      }
      printf("\n");
   }
}
int main()
{
   int n = 5;
   printPascal(n);
   return 0;
}
```



29.Sum of Digits of a Number:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
 int n,r,sum=0;
 printf("Enter the number:");
 scanf("%d", &n);
 n=abs(n);
 while(n>0)
 {
        r=n%10;
        sum=sum+r;
        n=n/10;
 }
 printf("%d",sum);
```

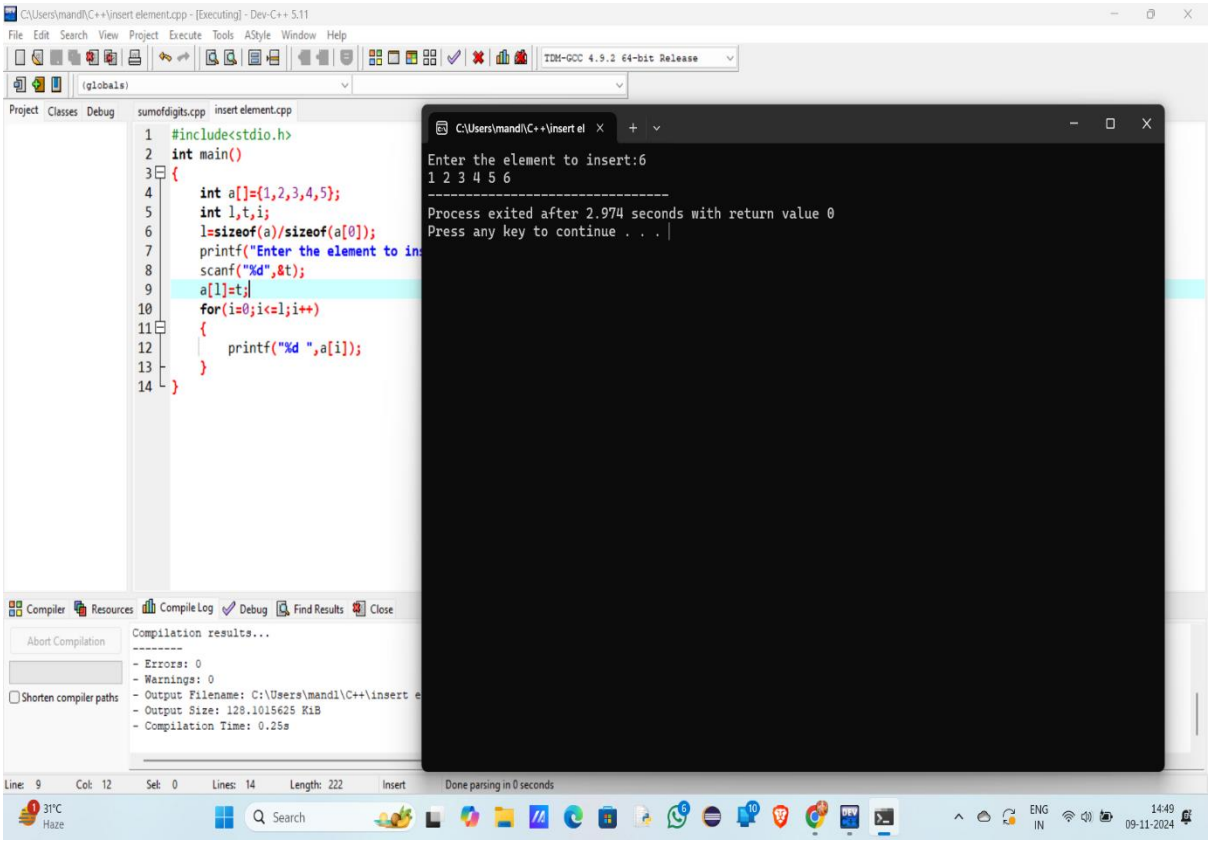
```
 return 0;
}
```



30.Insert an Element in an Array:

```
#include<stdio.h>
int main()
{
 int a[]={1,2,3,4,5};
 int l,t,i;
 l=sizeof(a)/sizeof(a[0]);
 printf("Enter the element to insert:");
 scanf("%d",&t);
 a[l]=t;
 for(i=0;i<=l;i++)
 {
        printf("%d ",a[i]);
 }
```

}

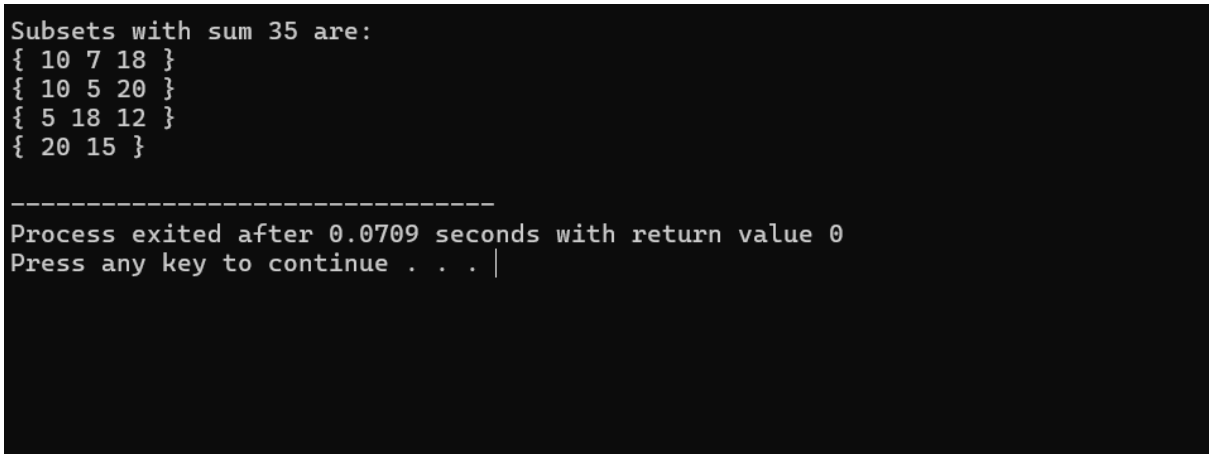


31. **Sum of Subsets Using Backtracking**:

```
#include <stdio.h>
void subsetSum(int arr[], int n, int target_sum, int index, int current_sum,
int current_subset[], int subset_size) {
   if (current_sum == target_sum) {
printf("{ ");
      for (int i = 0; i<subset_size; i++) {
printf("%d ", current_subset[i]);
      }
printf("}\n");
      return;
   }
   if (current_sum>target_sum || index == n) {
      return;
   }
current_subset[subset_size] = arr[index];
```

```
subsetSum(arr, n, target_sum, index + 1, current_sum + arr[index],
current_subset, subset_size + 1);
subsetSum(arr, n, target_sum, index + 1, current_sum, current_subset,
subset_size);
}
void findAllSubsets(int arr[], int n, int target_sum) {
   int current_subset[n];
subsetSum(arr, n, target_sum, 0, 0, current_subset, 0);
}
int main() {
   int arr[] = {10, 7, 5, 18, 12, 20, 15};
   int target_sum = 35;
   int n = sizeof(arr) / sizeof(arr[0]);
printf("Subsets with sum %d are:\n", target_sum);
findAllSubsets(arr, n, target_sum);
   return 0;
}
```

OUTPUT:



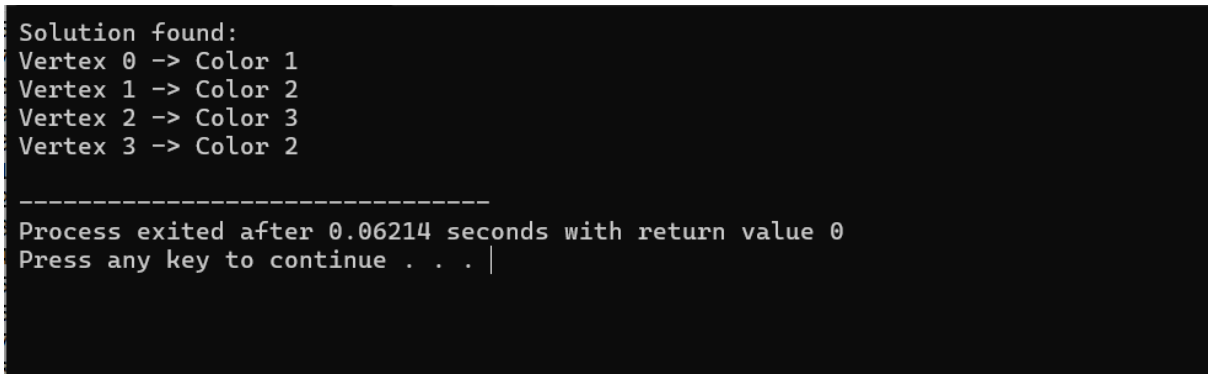
32.GRAPH COLOURING USING BACKTRACKING:

```
#include <stdio.h>
#include <stdbool.h>
#define N 4
bool isSafe(int vertex, int graph[N][N], int colors[], int color) {
   for (int i = 0; i< N; i++) {
      if (graph[vertex][i] &&colors[i] == color) {
         return false;
```

```
        }
    }
    return true;
}
bool graphColoring(int graph[N][N], int m, int colors[], int vertex) {
    if (vertex == N) {
        return true;
    }
    for (int color = 1; color<= m; color++) {
        if (isSafe(vertex, graph, colors, color)) {
colors[vertex] = color;
            if (graphColoring(graph, m, colors, vertex + 1)) {
                return true;
            }
colors[vertex] = 0;
        }
    }
    return false;
}
void solveGraphColoring(int graph[N][N], int m) {
    int colors[N] = {0};
    if (graphColoring(graph, m, colors, 0)) {
printf("Solution found:\n");
        for (int i = 0; i< N; i++) {
printf("Vertex %d ->Color %d\n", i, colors[i]);
        }
    } else {
printf("No solution exists\n");
    }
}
int main() {
    int graph[N][N] = {
        {0, 1, 1, 1},
        {1, 0, 1, 0},
        {1, 1, 0, 1},
        {1, 0, 1, 0}
    };
```

```
   int m = 3;
solveGraphColoring(graph, m);
   return 0;
}
```

OUTPUT:



33.CONTAINER LOADING PROBLEM:

```
#include <stdio.h>
int maxLoad = 0;
void backtrack(int weights[], int n, int capacity, int index, int currentLoad)
{
   if (currentLoad> capacity) {
      return;
   }
   if (currentLoad>maxLoad) {
maxLoad = currentLoad;
   }
   if (index == n) {
      return;
   }
backtrack(weights, n, capacity, index + 1, currentLoad + weights[index]);
backtrack(weights, n, capacity, index + 1, currentLoad);
}
int maxContainerLoad(int weights[], int n, int capacity) {
maxLoad = 0;
backtrack(weights, n, capacity, 0, 0);
   return maxLoad;
}
int main() {
```

```
   int weights[] = {10, 20, 30, 40};
   int n = sizeof(weights) / sizeof(weights[0]);
   int capacity = 50;
   int maxLoadPossible = maxContainerLoad(weights, n, capacity);
printf("Maximum load that can be loaded: %d\n", maxLoadPossible);
   return 0;
}
```

OUTPUT:

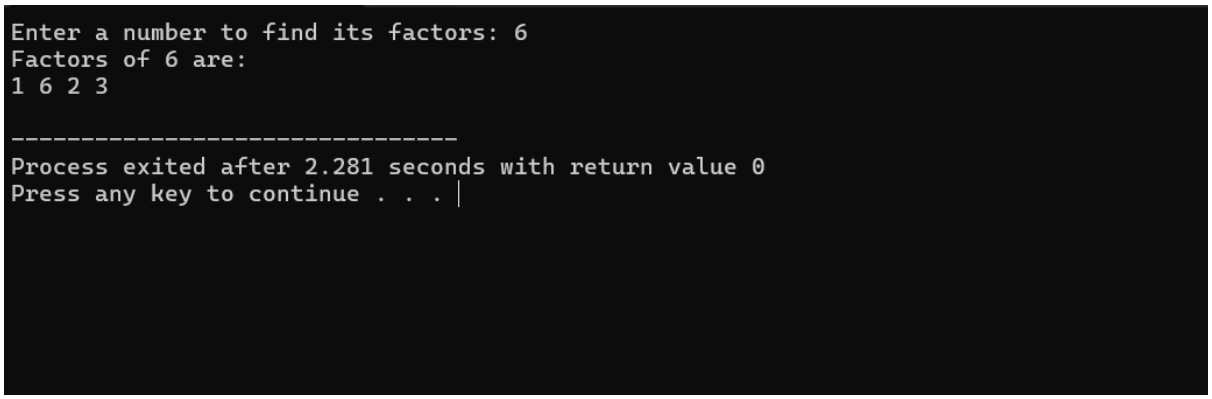
```
Maximum load that can be loaded: 50

--------------------------------
Process exited after 0.06523 seconds with return value 0
Press any key to continue . . .
```

34.LIST OF ALL FACTORS FOR N VALUE:

```
#include <stdio.h>
#include <math.h>
void findFactors(int n) {
printf("Factors of %d are:\n", n);
   for (int i = 1; i<= sqrt(n); i++) {
      if (n % i == 0) {
printf("%d ", i);
         if (i != n / i) {
printf("%d ", n / i);
         }
      }
   }
printf("\n");
}
int main() {
   int n;
printf("Enter a number to find its factors: ");
scanf("%d", &n);
findFactors(n);
   return 0;
```

```
}
```

OUTPUT:



35.JOB ASSIGNMENT PROBLEM USING BRANCH AND BOUND:

```
#include <stdio.h>
#include <limits.h>
#include <stdbool.h>
#define N 4
typedef struct Node {
   int cost;
   int lowerBound;
   int jobAssignment[N];
   bool assigned[N];
   int level;
} Node;
int calculateLowerBound(int costMatrix[N][N], bool assigned[N], int level)
{
   int lowerBound = 0;

   for (int i = level; i< N; i++) {
      int minCost = INT_MAX;
      for (int j = 0; j < N; j++) {
         if (!assigned[j] &&costMatrix[i][j] <minCost) {
minCost = costMatrix[i][j];
         }
      }
lowerBound += minCost;
```

```
    }
    return lowerBound;
}
void branchAndBound(int costMatrix[N][N]) {
    int minCost = INT_MAX;
    Node bestNode;
    Node root;
root.cost = 0;
root.level = 0;
    for (int i = 0; i< N; i++) {
root.assigned[i] = false;
root.jobAssignment[i] = -1;
    }
root.lowerBound = calculateLowerBound(costMatrix, root.assigned,
root.level);
    Node queue[N * N];
    int queueSize = 0;
    queue[queueSize++] = root;
    while (queueSize> 0) {
        Node currentNode = queue[--queueSize];
        if (currentNode.lowerBound>= minCost) continue;
        if (currentNode.level == N) {
            if (currentNode.cost<minCost) {
minCost = currentNode.cost;
bestNode = currentNode;
            }
            continue;
        }
        for (int job = 0; job < N; job++) {
            if (!currentNode.assigned[job]) {
                Node newNode = currentNode;
newNode.level++;
newNode.jobAssignment[currentNode.level - 1] = job;
newNode.cost += costMatrix[currentNode.level - 1][job];
newNode.assigned[job] = true;
newNode.lowerBound = newNode.cost +
calculateLowerBound(costMatrix, newNode.assigned, newNode.level);
```

```
            if (newNode.lowerBound<minCost) {
                queue[queueSize++] = newNode;
            }
        }
    }
  }
printf("Minimum cost: %d\n", minCost);
printf("Job assignments:\n");
  for (int i = 0; i< N; i++) {
printf("Person %d -> Job %d\n", i, bestNode.jobAssignment[i]);
  }
}
int main() {
  int costMatrix[N][N] = {
    {9, 2, 7, 8},
    {6, 4, 3, 7},
    {5, 8, 1, 8},
    {7, 6, 9, 4}
  };
branchAndBound(costMatrix);
  return 0;
}
```

OUTPUT:

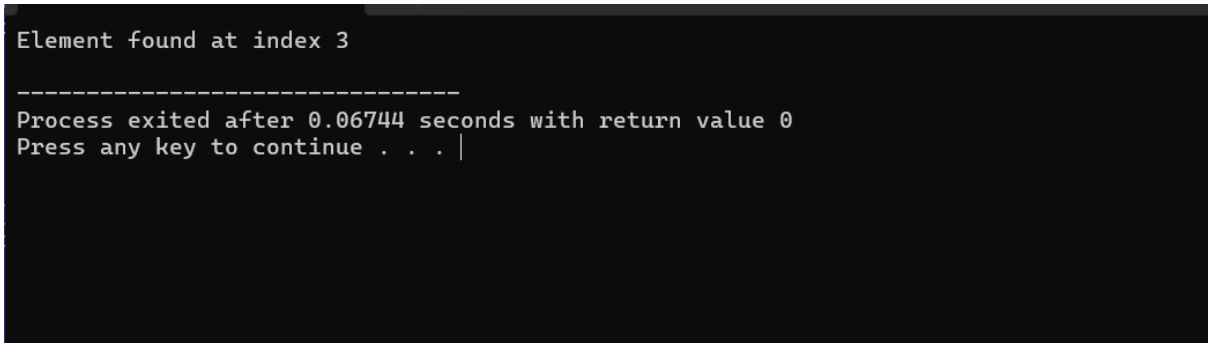
```
Minimum cost: 10
Job assignments:
Person 0 -> Job 1
Person 1 -> Job 2
Person 2 -> Job 0
Person 3 -> Job -1

--------------------------------
Process exited after 0.04755 seconds with return value 0
Press any key to continue . . .
```

36.LINEAR SEARCH:

```
#include <stdio.h>
int linearSearch(int arr[], int n, int target) {
  for (int i = 0; i< n; i++) {
```

```
        if (arr[i] == target) {
            return i;
        }
    }
    return -1;
}
int main() {
    int arr[] = {34, 21, 56, 78, 90, 23, 12};
    int n = sizeof(arr) / sizeof(arr[0]);
    int target = 78;
    int result = linearSearch(arr, n, target);
    if (result != -1) {
printf("Element found at index %d\n", result);
    } else {
printf("Element not found in the array\n");
    }
    return 0;
}
```

OUTPUT:



37. HAMILTONIAN CIRCUIT USING BACKTRACKING:

```
#include <stdio.h>
#include <stdbool.h>
#define V 5
bool canAddToPath(int v, int graph[V][V], int path[], int position) {
    if (graph[path[position - 1]][v] == 0)
        return false;
    for (int i = 0; i< position; i++) {
        if (path[i] == v)
```

```
            return false;
        }
        return true;
    }
    bool hamiltonianCycle(int graph[V][V], int path[], int position) {
        if (position == V) {
            if (graph[path[position - 1]][path[0]] == 1)
                return true;
            else
                return false;
        }
        for (int v = 1; v < V; v++) {
            if (canAddToPath(v, graph, path, position)) {
                path[position] = v;
                if (hamiltonianCycle(graph, path, position + 1))
                    return true;
                path[position] = -1;
            }
        }
        return false;
    }
    int main() {
        int graph[V][V] = {
            {0, 1, 0, 1, 0},
            {1, 0, 1, 1, 0},
            {0, 1, 0, 1, 1},
            {1, 1, 1, 0, 1},
            {0, 0, 1, 1, 0}
        };
        int path[V];
        for (int i = 0; i< V; i++) {
            path[i] = -1;
        }
    path[0] = 0;
        if (hamiltonianCycle(graph, path, 1)) {
    printf("Hamiltonian Cycle found: \n");
            for (int i = 0; i< V; i++) {
```

```
printf("%d ", path[i]);
      }
printf("%d\n", path[0]);
   } else {
printf("No Hamiltonian Cycle found\n");
   }
   return 0;
}
```

OUTPUT:

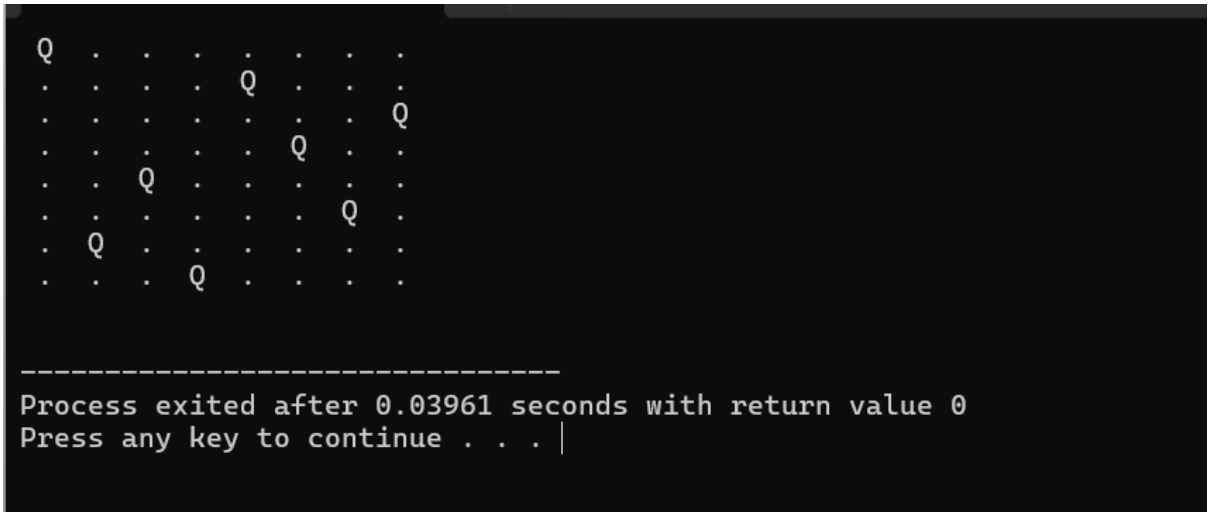
```
Hamiltonian Cycle found:
0 1 2 4 3 0

--------------------------------
Process exited after 0.05161 seconds with return value 0
Press any key to continue . . .
```

38.N QUEENS PROBLEM:

```
#include <stdio.h>
#include <stdbool.h>
#define N 8
int board[N][N];
void printSolution() {
   for (int i = 0; i< N; i++) {
      for (int j = 0; j < N; j++) {
         if (board[i][j] == 1)
printf(" Q ");
         else
printf(" . ");
      }
printf("\n");
   }
printf("\n");
}
bool isSafe(int row, int col) {
   for (int i = 0; i< row; i++) {
```

```
        if (board[i][col] == 1)
            return false;
    }
    for (int i = row, j = col; i>= 0 && j >= 0; i--, j--) {
        if (board[i][j] == 1)
            return false;
    }
    for (int i = row, j = col; i>= 0 && j < N; i--, j++) {
        if (board[i][j] == 1)
            return false;
    }
    return true;
}
bool solveNQueens(int row) {
    if (row == N)
        return true;
    for (int col = 0; col < N; col++) {
        if (isSafe(row, col)) {
            board[row][col] = 1;
            if (solveNQueens(row + 1))
                return true;
            board[row][col] = 0;
        }
    }
    return false;
}
int main() {
    for (int i = 0; i< N; i++)
        for (int j = 0; j < N; j++)
            board[i][j] = 0;
    if (solveNQueens(0)) {
printSolution();
    } else {
printf("No solution exists\n");
    }
    return 0;
}
```

OUTPUT:

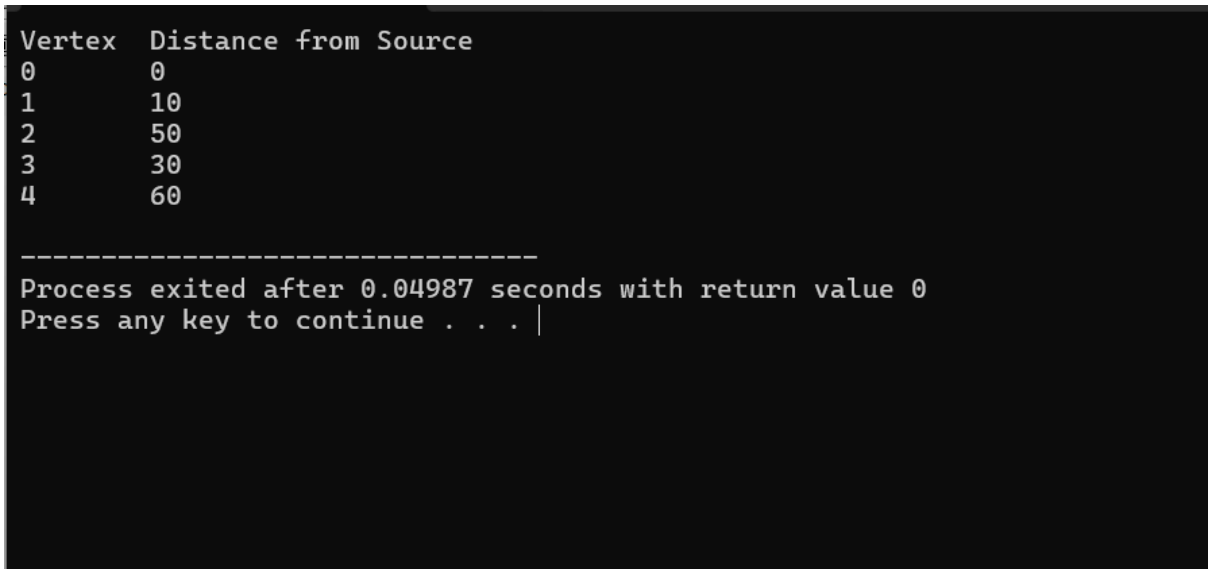


39.OPTIMAL COST BY USING APPROPRIATE ALGORITHM:

```
#include <stdio.h>
#include <limits.h>
#include <stdbool.h>
#define V 5
#define INF INT_MAX
void dijkstra(int graph[V][V], int src) {
   int dist[V];
   bool sptSet[V];
   for (int i = 0; i< V; i++) {
dist[i] = INF;
sptSet[i] = false;
   }
dist[src] = 0;
   for (int count = 0; count < V - 1; count++) {
      int u = -1;
      for (int v = 0; v < V; v++) {
         if (!sptSet[v] && (u == -1 || dist[v] <dist[u])) {
            u = v;
         }
      }
sptSet[u] = true;
      for (int v = 0; v < V; v++) {
```

```
        if (graph[u][v] && !sptSet[v] &&dist[u] != INF &&dist[u] +
graph[u][v] <dist[v]) {
dist[v] = dist[u] + graph[u][v];
        }
      }
   }
printf("Vertex\tDistance from Source\n");
   for (int i = 0; i< V; i++) {
printf("%d\t%d\n", i, dist[i]);
   }
}
int main() {
   int graph[V][V] = {
      {0, 10, 0, 30, 0},
      {10, 0, 50, 0, 0},
      {0, 50, 0, 20, 10},
      {30, 0, 20, 0, 60},
      {0, 0, 10, 60, 0}
   };

dijkstra(graph, 0);

   return 0;
}
```

OUTPUT:



40.MIN MAX VALUE SEPERATELY FOR ALL NUMBERS IN THE LIST:

```
#include <stdio.h>
void findMinMax(int numbers[], int size, int* min, int* max) {
   *min = numbers[0];
   *max = numbers[0];
   for (int i = 1; i< size; i++) {
      if (numbers[i] < *min) {
         *min = numbers[i];
      }
      if (numbers[i] > *max) {
         *max = numbers[i];
      }
   }
}
int main() {
   int numbers[] = {34, 21, 56, 78, 90, 23, 12};
   int size = sizeof(numbers) / sizeof(numbers[0]);
   int min, max;
findMinMax(numbers, size, &min, &max);
printf("Minimum value: %d\n", min);
printf("Maximum value: %d\n", max);
   return 0;
}
```

OUTPUT: