CSA0683-DESIGN AND ANALYSIS OF ALGORITHMS FOR SEARCHING ALGORITHMS

B.Nikhil(192111054)

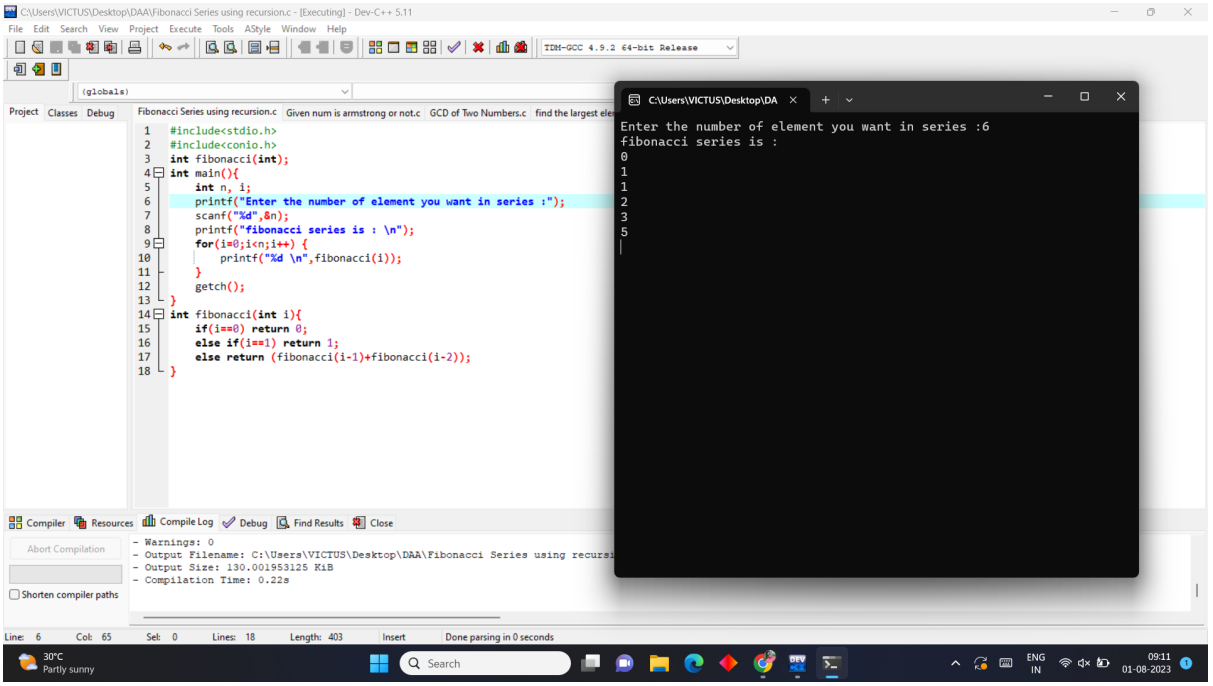
DAY-1

1. Write a C program to Print Fibonacci Series using recursion

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int fibonacci(int);
int main(){
	int n, i;
	printf("Enter the number of element you want in series :");
	scanf("%d",&n);
	printf("fibonacci series is : \n");
	for(i=0;i<n;i++) {
		printf("%d \n",fibonacci(i));
	}
	getch();
}
int fibonacci(int i){
	if(i==0) return 0;
	else if(i==1) return 1;
	else return (fibonacci(i-1)+fibonacci(i-2));
}
```

OUTPUT :

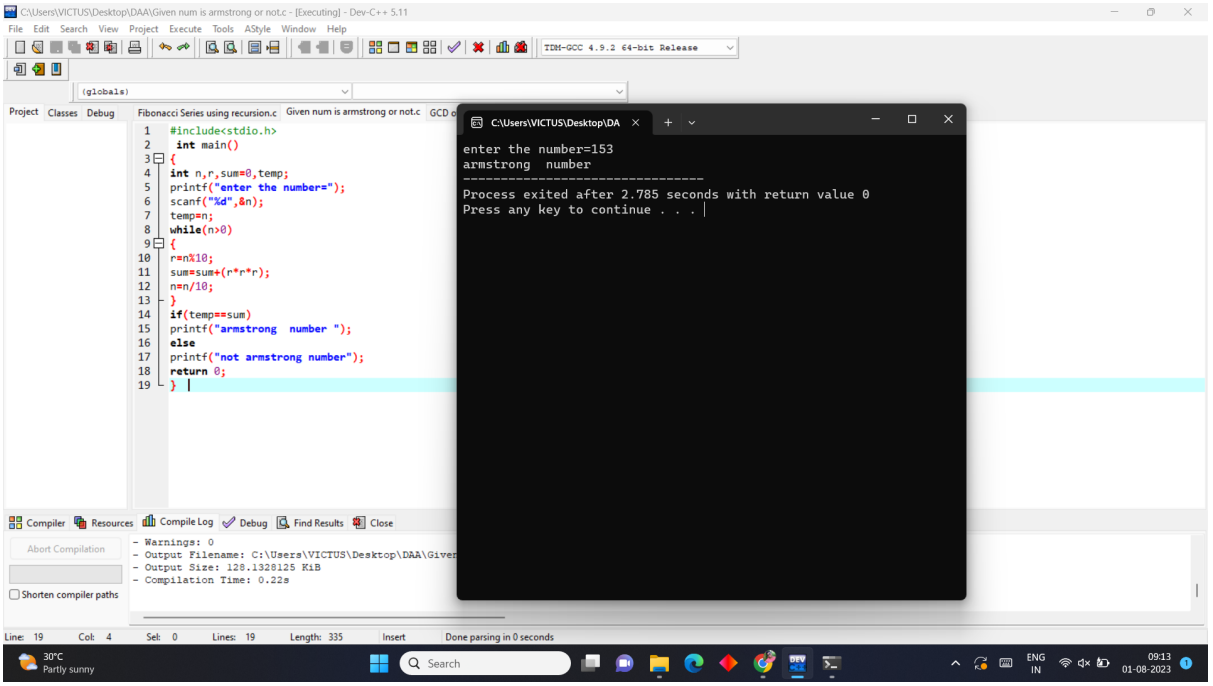


2. Write a C program to check if the given no is Armstrong or not.

PROGRAM:

```
#include<stdio.h>
 int main()
{
int n,r,sum=0,temp;
printf("enter the number=");
scanf("%d",&n);
temp=n;
while(n>0)
{
r=n%10;
sum=sum+(r*r*r);
n=n/10;
}
if(temp==sum)
printf("armstrong  number ");
else
printf("not armstrong number");
return 0;
}
```

OUTPUT:

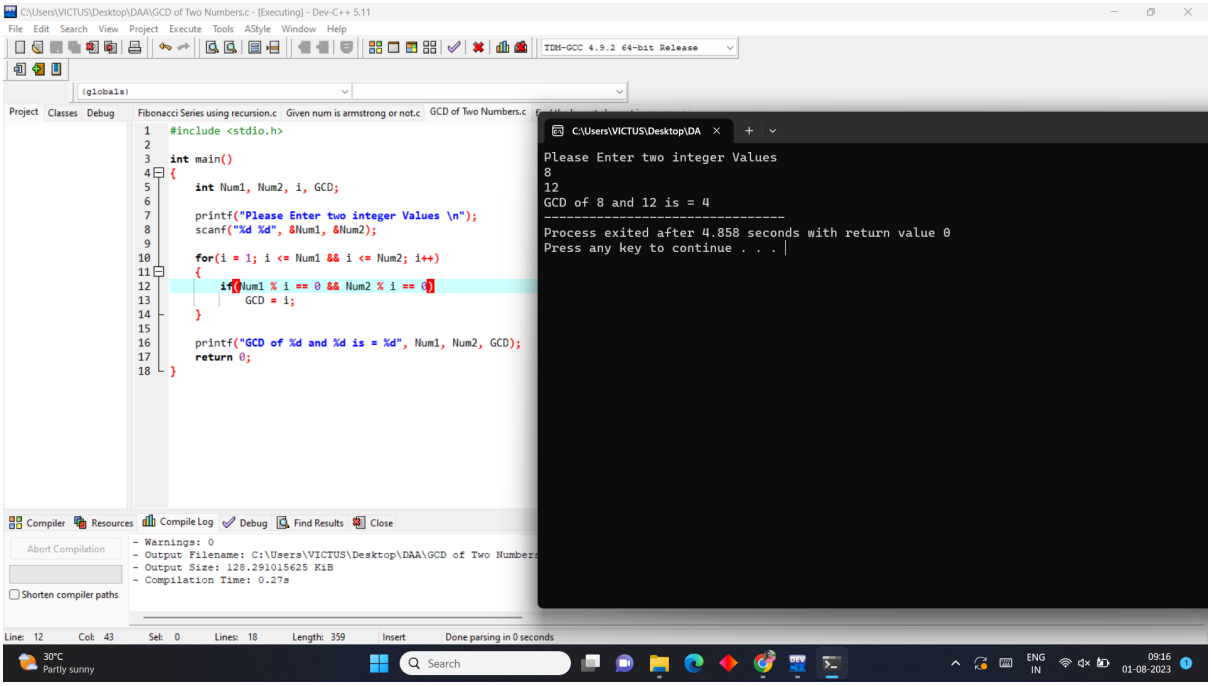


3. Write a C program to find the GCD of two numbers .

PROGRAM:

```
#include <stdio.h>
int main()
{
   int Num1, Num2, i, GCD;
   printf("Please Enter two integer Values \n");
   scanf("%d %d", &Num1, &Num2);
   for(i = 1; i <= Num1 && i <= Num2; i++)
   {
      if(Num1 % i == 0 && Num2 % i == 0)
         GCD = i;
   }
   printf("GCD of %d and %d is = %d", Num1, Num2, GCD);
   return 0;
}
```

OUTPUT :

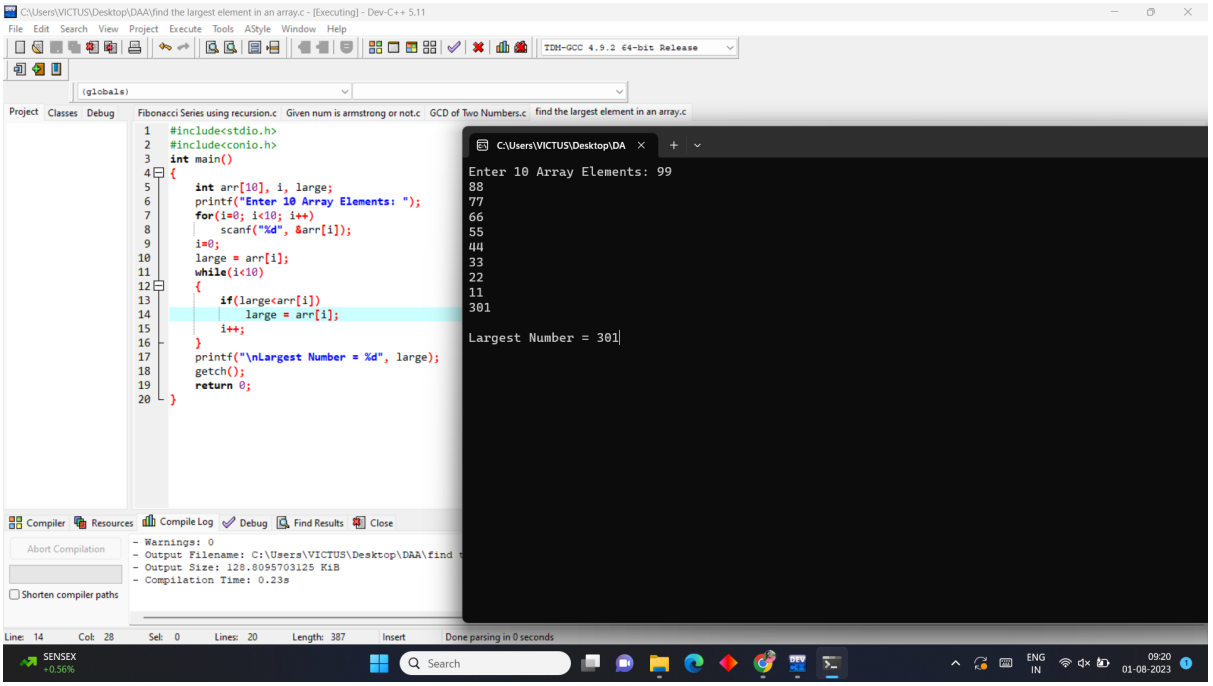


4. Write a C program to get the largest element of an array.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int main()
{
   int arr[10], i, large;
   printf("Enter 10 Array Elements: ");
   for(i=0; i<10; i++)
      scanf("%d", &arr[i]);
   i=0;
   large = arr[i];
   while(i<10)
   {
      if(large<arr[i])
         large = arr[i];
      i++;
   }
   printf("\nLargest Number = %d", large);
   getch();
   return 0;
}
```

OUTPUT

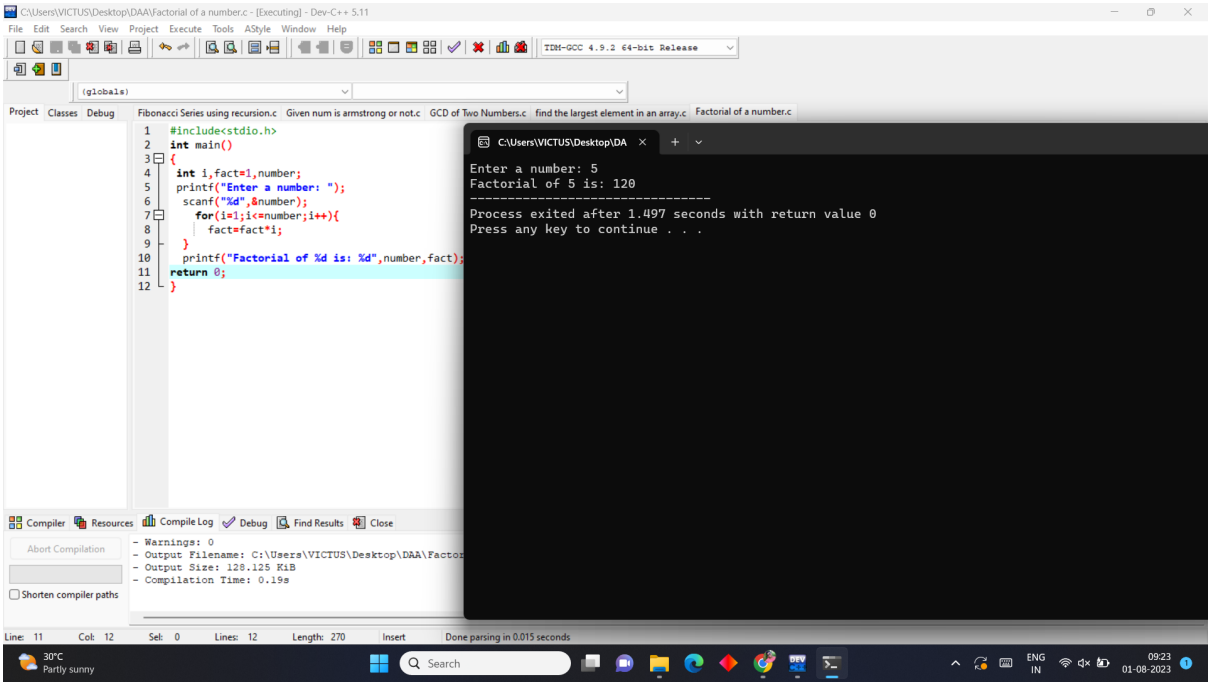


5. Write a C program to find the Factorial of a number .

PROGRAM

```
#include<stdio.h>
int main()
{
 int i,fact=1,number;
 printf("Enter a number: ");
  scanf("%d",&number);
    for(i=1;i<=number;i++){
      fact=fact*i;
  }
  printf("Factorial of %d is: %d",number,fact);
return 0;
}
```

OUTPUT

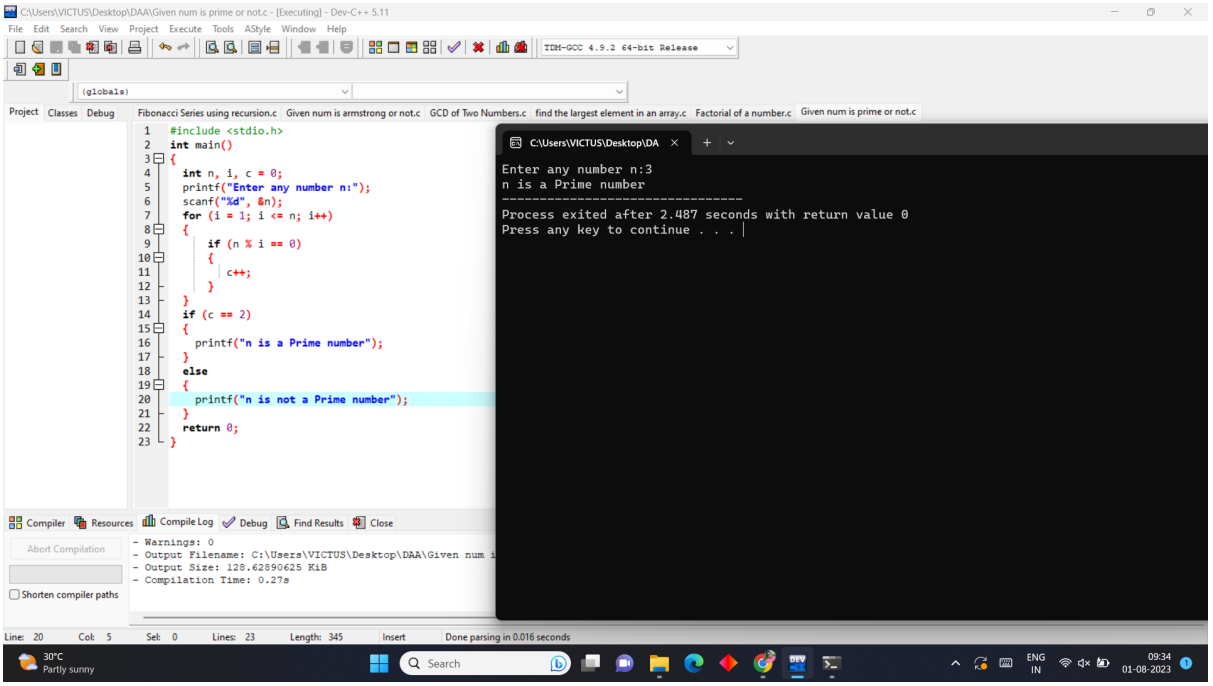


6. Write a C program to check if a number is a prime number or not .

PROGRAM

```
#include <stdio.h>
int main()
{
  int n, i, c = 0;
  printf("Enter any number n:");
  scanf("%d", &n);
  for (i = 1; i <= n; i++)
  {
     if (n % i == 0)
           {
        c++;
     }
  }
  if (c == 2)
  {
    printf("n is a Prime number");
  }
  else
  {
    printf("n is not a Prime number");
  }
  return 0;
}
```

OUTPUT

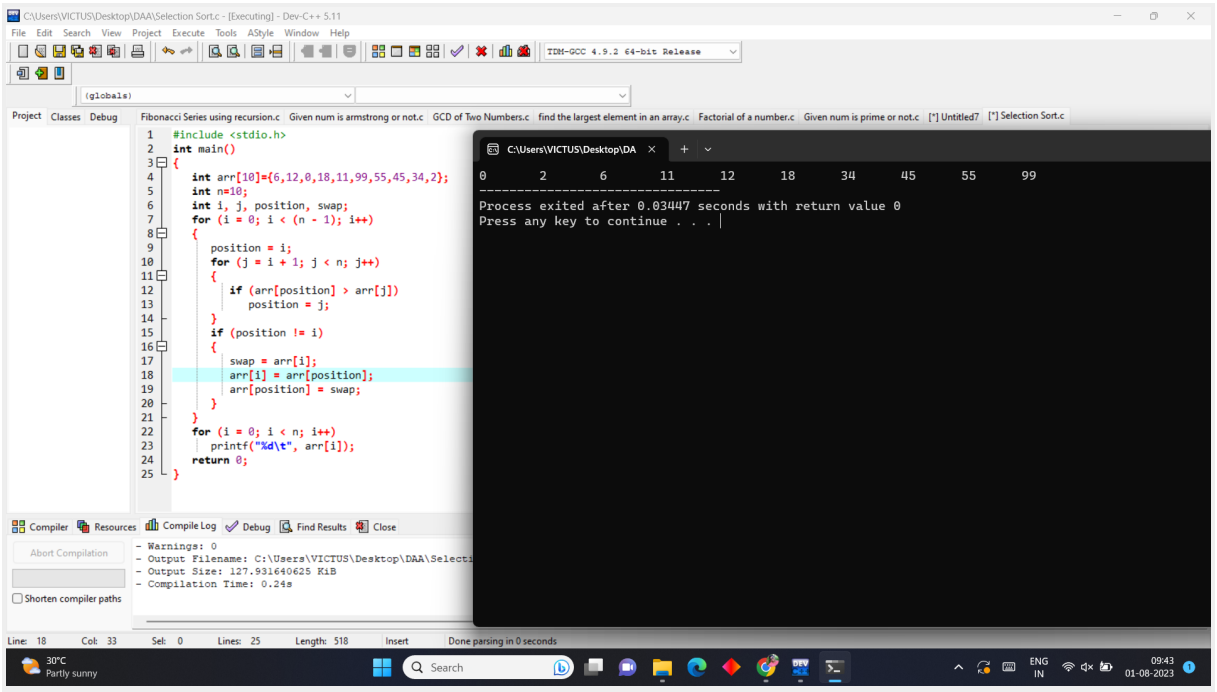


7. Write a C program to perform Selection sort.

PROGRAM

```
#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



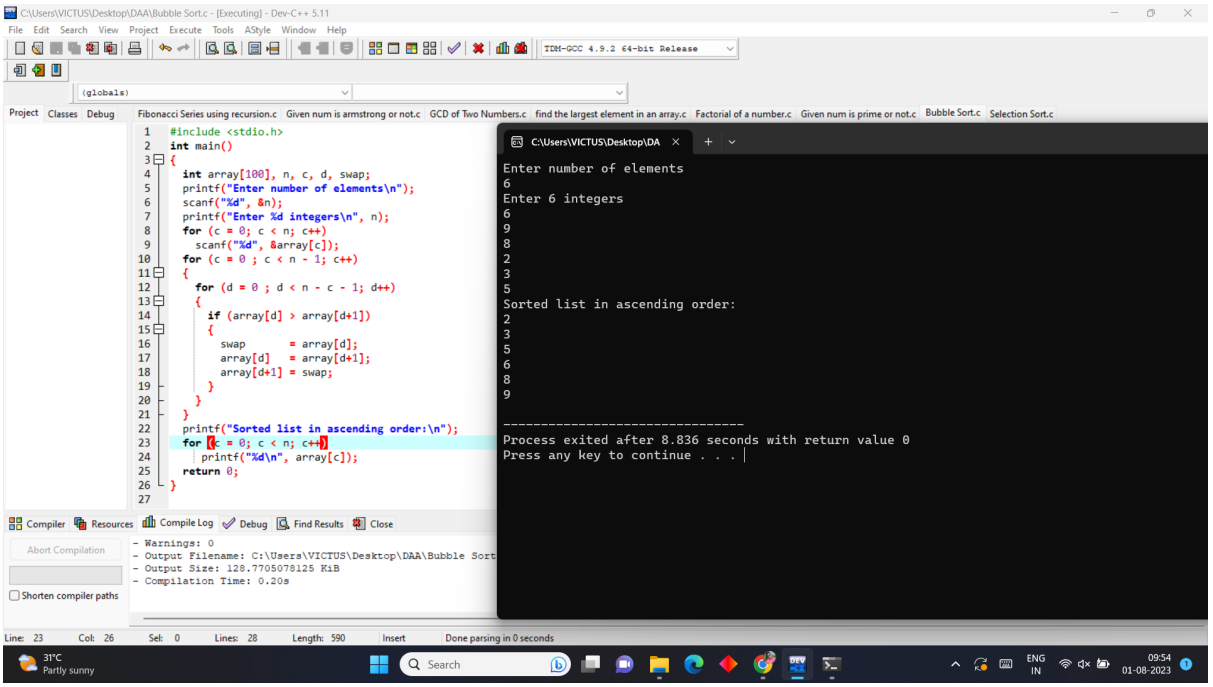
8. Write a C program to perform Bubble sort

PROGRAM

```
#include <stdio.h>
int main()
{
  int array[100], n, c, d, swap;
  printf("Enter number of elements\n");
  scanf("%d", &n);
  printf("Enter %d integers\n", n);
  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);
  for (c = 0 ; c < n - 1; c++)
  {
    for (d = 0 ; d < n - c - 1; d++)
    {
      if (array[d] > array[d+1])
      {
        swap       = array[d];
        array[d]   = array[d+1];
        array[d+1] = swap;
      }
    }
  }
  printf("Sorted list in ascending order:\n");
```

```
  for (c = 0; c < n; c++)
     printf("%d\n", array[c]);
  return 0;
}
```

OUTPUT



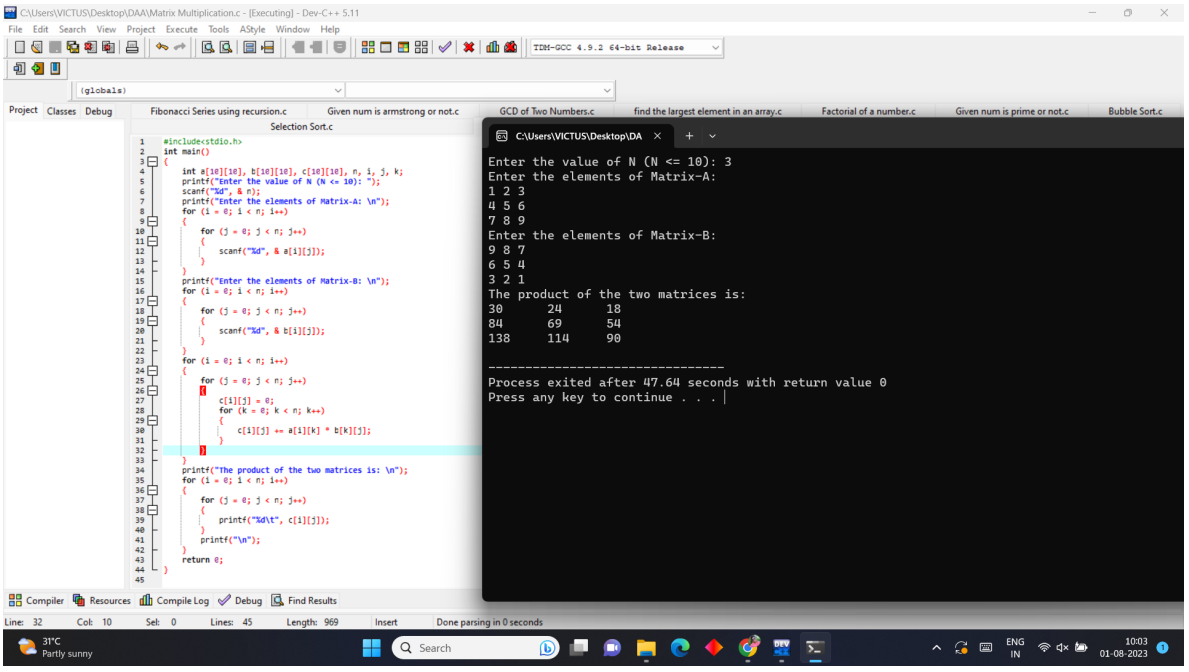
9. Write a C program to multiply two Matrix

PROGRAM

```
#include<stdio.h>
int main()
{
   int a[10][10], b[10][10], c[10][10], n, i, j, k;
   printf("Enter the value of N (N <= 10): ");
   scanf("%d", & n);
   printf("Enter the elements of Matrix-A: \n");
   for (i = 0; i < n; i++)
         {
      for (j = 0; j < n; j++)
                  {
         scanf("%d", & a[i][j]);
      }
   }
   printf("Enter the elements of Matrix-B: \n");
   for (i = 0; i < n; i++)
         {
      for (j = 0; j < n; j++)
```

```
                {
            scanf("%d", & b[i][j]);
        }
    }
    for (i = 0; i < n; i++)
        {
        for (j = 0; j < n; j++)
                {
            c[i][j] = 0;
            for (k = 0; k < n; k++)
                        {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
    printf("The product of the two matrices is: \n");
    for (i = 0; i < n; i++)
        {
        for (j = 0; j < n; j++)
                {
            printf("%d\t", c[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

OUTPUT

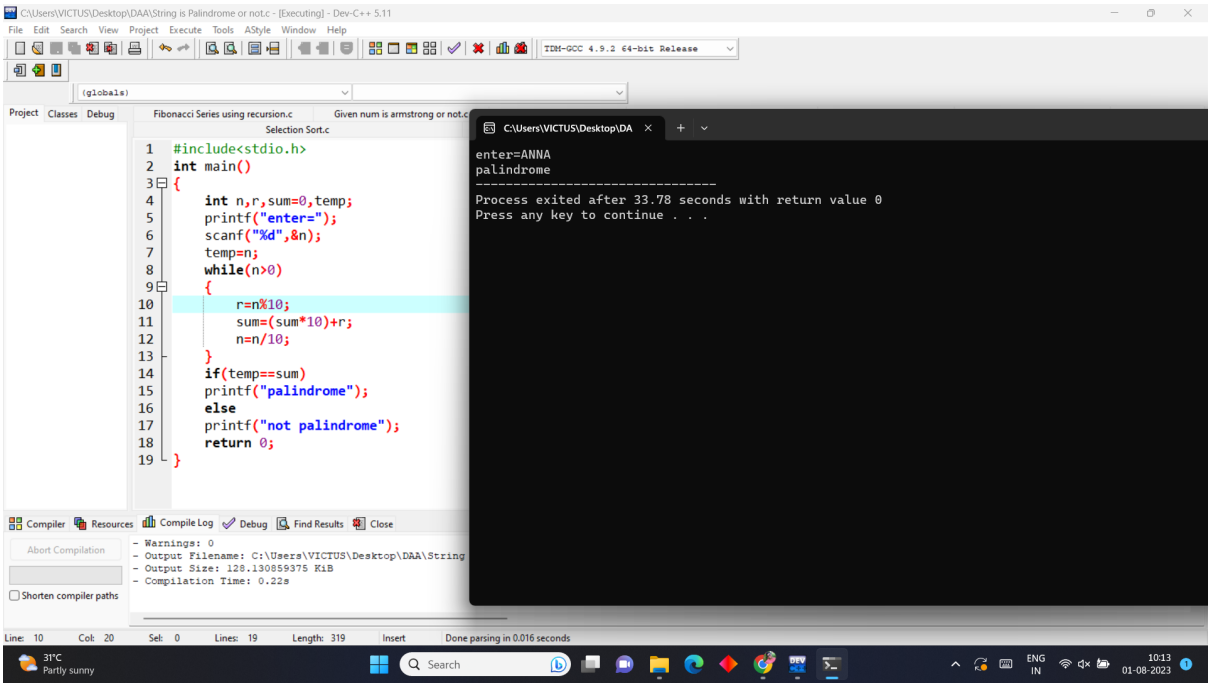


10. Write a C program for to check whether a given String is Palindrome or not

PROGRAM

```
#include<stdio.h>
int main()
{
	int n,r,sum=0,temp;
	printf("enter=");
	scanf("%d",&n);
	temp=n;
	while(n>0)
	{
		r=n%10;
		sum=(sum*10)+r;
		n=n/10;
	}
	if(temp==sum)
	printf("palindrome");
	else
	printf("not palindrome");
	return 0;
}
```

OUTPUT



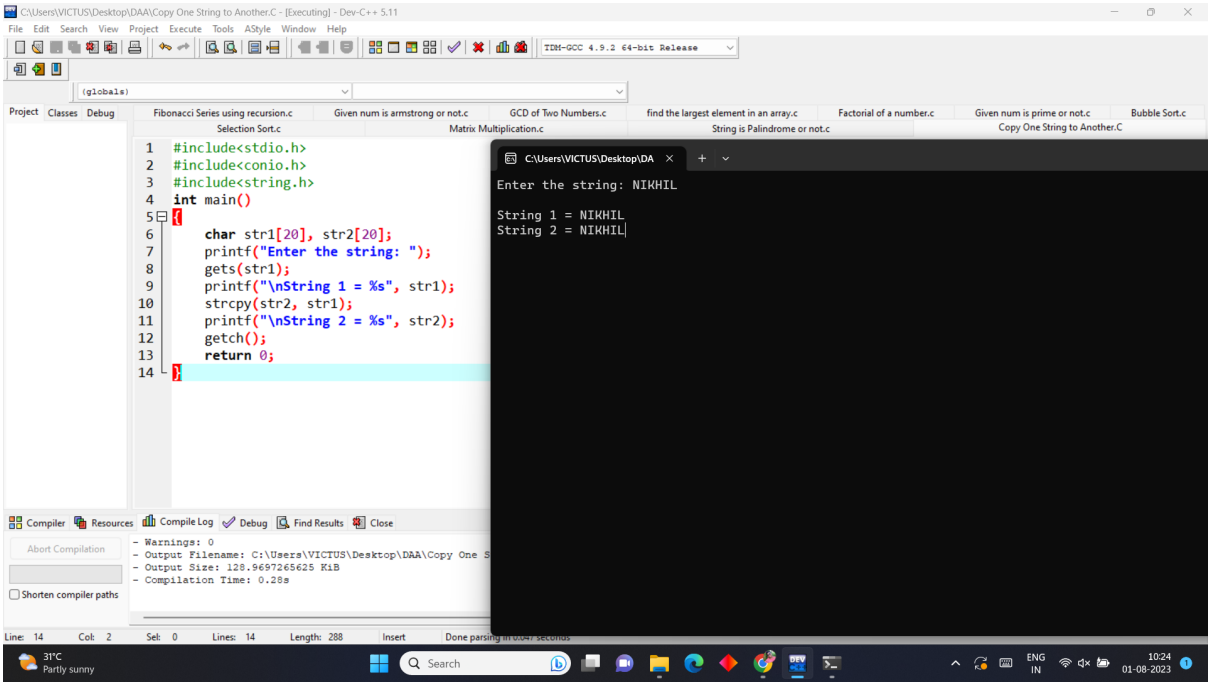
11.Write a C program to copy one string to another

PROGRAM

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

```
#include<conio.h>
#include<string.h>
int main()
{
    char str1[20], str2[20];
    printf("Enter the string: ");
    gets(str1);
    printf("\nString 1 = %s", str1);
    strcpy(str2, str1);
    printf("\nString 2 = %s", str2);
    getch();
    return 0;
}
```

OUTPUT



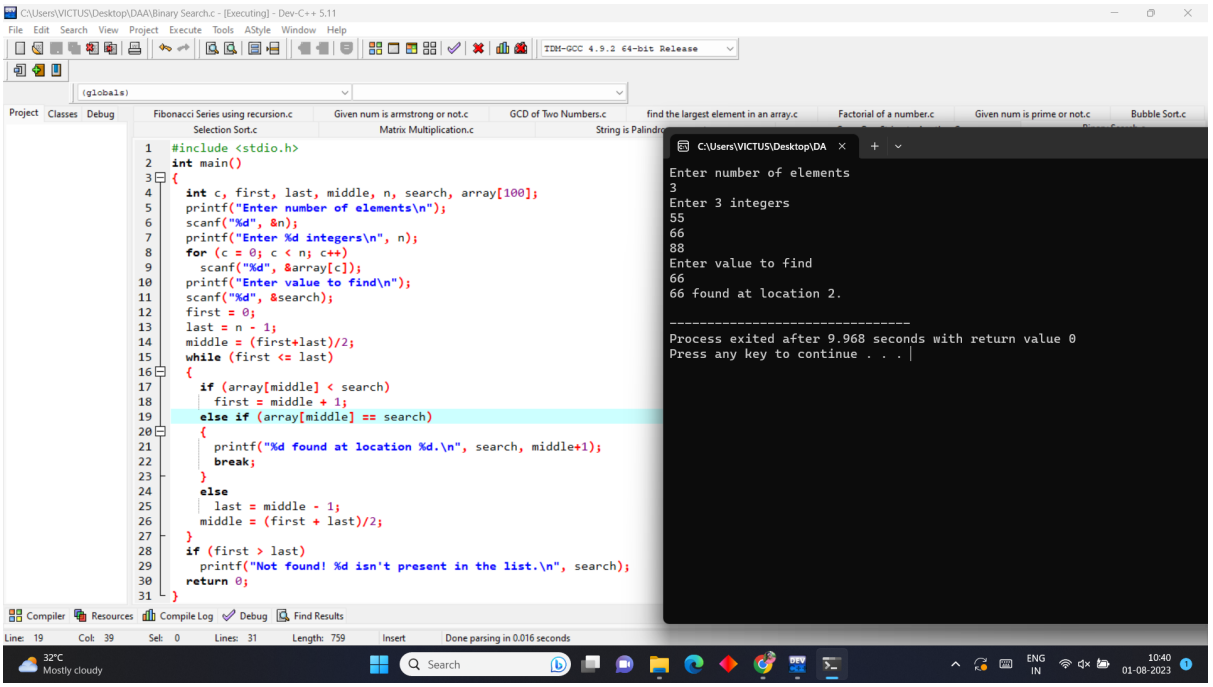
12. Write a Program to perform binary search.

PROGRAM

```
#include <stdio.h>
int main()
{
  int c, first, last, middle, n, search, array[100];
  printf("Enter number of elements\n");
  scanf("%d", &n);
  printf("Enter %d integers\n", n);
  for (c = 0; c < n; c++)
```

```
  scanf("%d", &array[c]);
 printf("Enter value to find\n");
 scanf("%d", &search);
 first = 0;
 last = n - 1;
 middle = (first+last)/2;
 while (first <= last)
 {
  if (array[middle] < search)
    first = middle + 1;
  else if (array[middle] == search)
        {
    printf("%d found at location %d.\n", search, middle+1);
    break;
  }
  else
    last = middle - 1;
  middle = (first + last)/2;
 }
 if (first > last)
  printf("Not found! %d isn't present in the list.\n", search);
 return 0;
}
```

OUTPUT

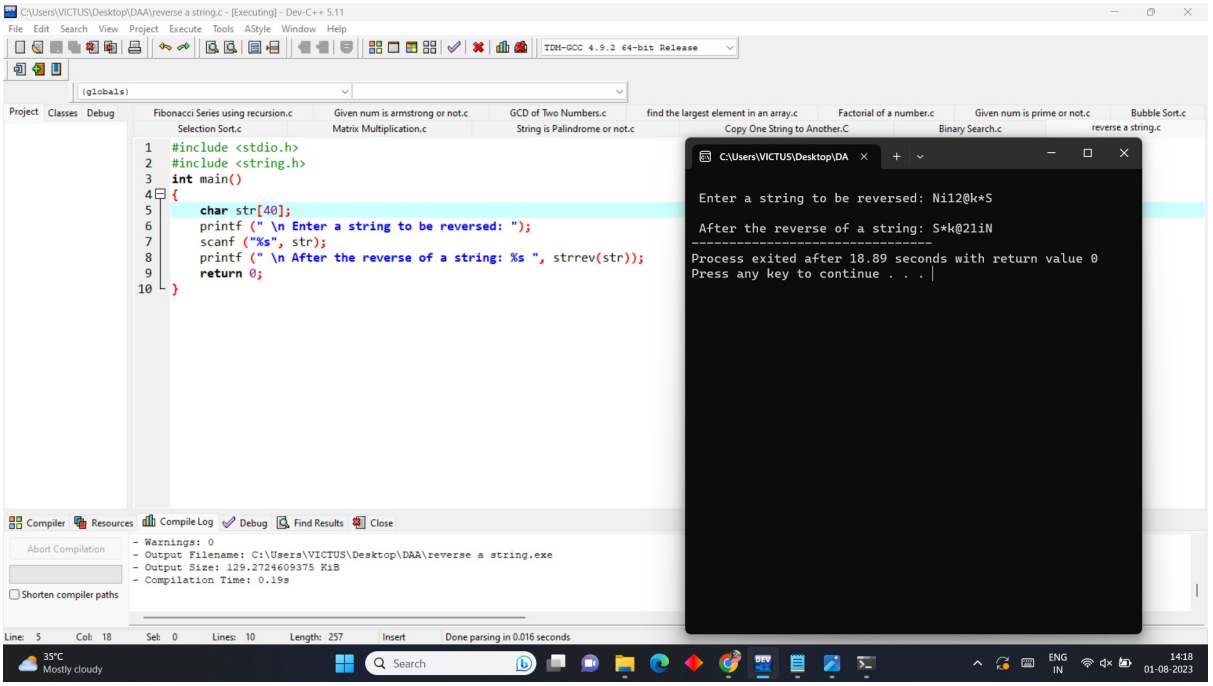


13. Write a program to print the reverse of a string

PROGRAM:

```
#include <stdio.h>
#include <string.h>
int main()
{
   char str[40];
   printf (" \n Enter a string to be reversed: ");
   scanf ("%s", str);
   printf (" \n After the reverse of a string: %s ", strrev(str));
   return 0;
}
```

OUTPUT

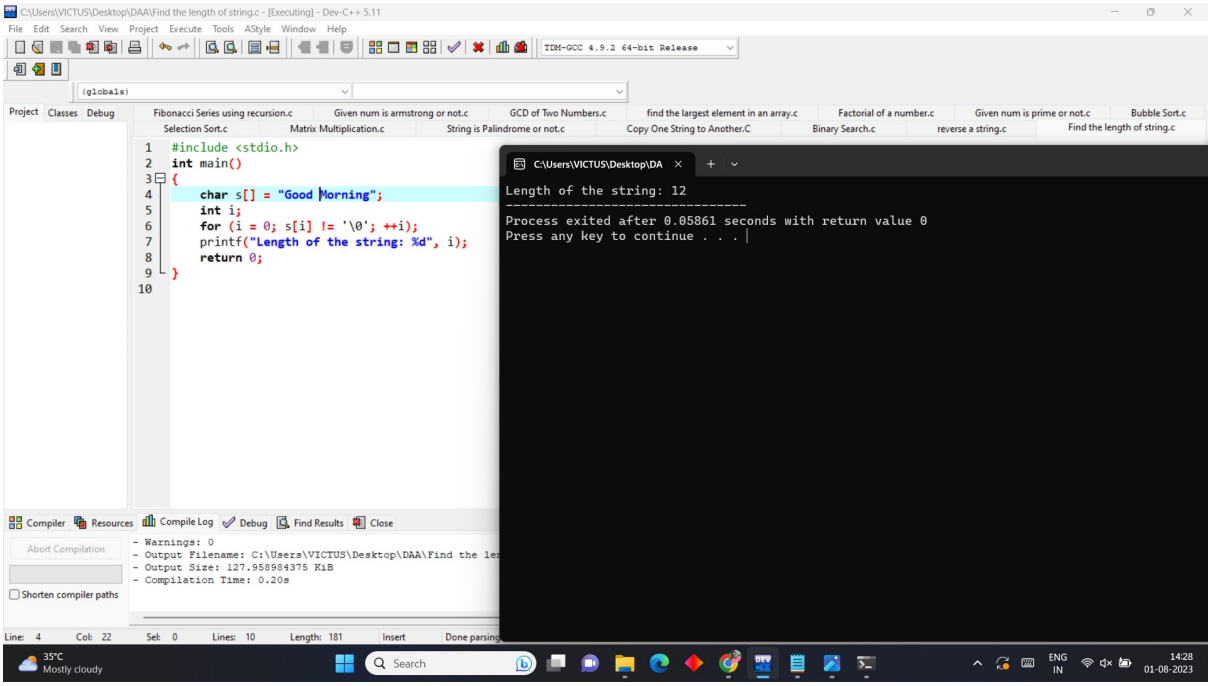


14. Write a program to find the length of a string.

PROGRAM

```
#include <stdio.h>
int main() {
   char s[] = "Good Morning";
   int i;

   for (i = 0; s[i] != '\0'; ++i);

   printf("Length of the string: %d", i);
   return 0;
}
```

OUTPUT



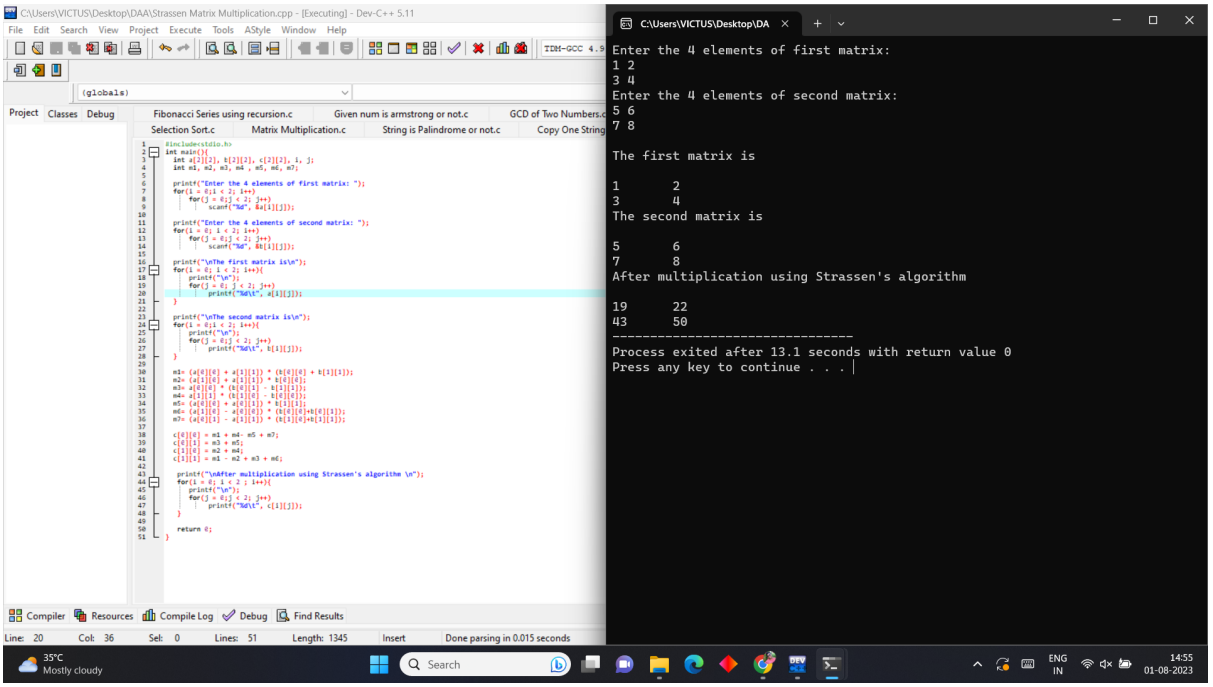
15. Write a program to perform Strassen's Matrix Multiplication.

PROGRAM

```
#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;
}
```

OUTPUT


C:\Users\VICTUS\Desktop\DAA\Strassen Matrix Multiplication.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
Project Classes Debug
Fibonacci Series using recursion.c
Given num is armstrong or not.c
GCD of Two Numbers.c
Selection Sort.c
Matrix Multiplication.c
String is Palindrome or not.c
Copy One String
#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;
}
Compiler Resources Compile Log Debug Find Results
Line: 20 Col: 36 Sel: 0 Lines: 51 Length: 1345 Insert Done parsing in 0.015 seconds
C:\Users\VICTUS\Desktop\DA
Enter the 4 elements of first matrix:
1 2
3 4
Enter the 4 elements of second matrix:
5 6
7 8
The first matrix is
1 2
3 4
The second matrix is
5 6
7 8
After multiplication using Strassen's algorithm
19 22
43 50
Process exited after 13.1 seconds with return value 0
Press any key to continue . . .
35°C
Mostly cloudy
Search
ENG
IN
14:55
01-08-2023

CSA0683-DESIGN AND ANALYSIS OF ALGORITHMS FOR SEARCHING ALGORITHMS

B.Nikhil(192111054)

DAY-2

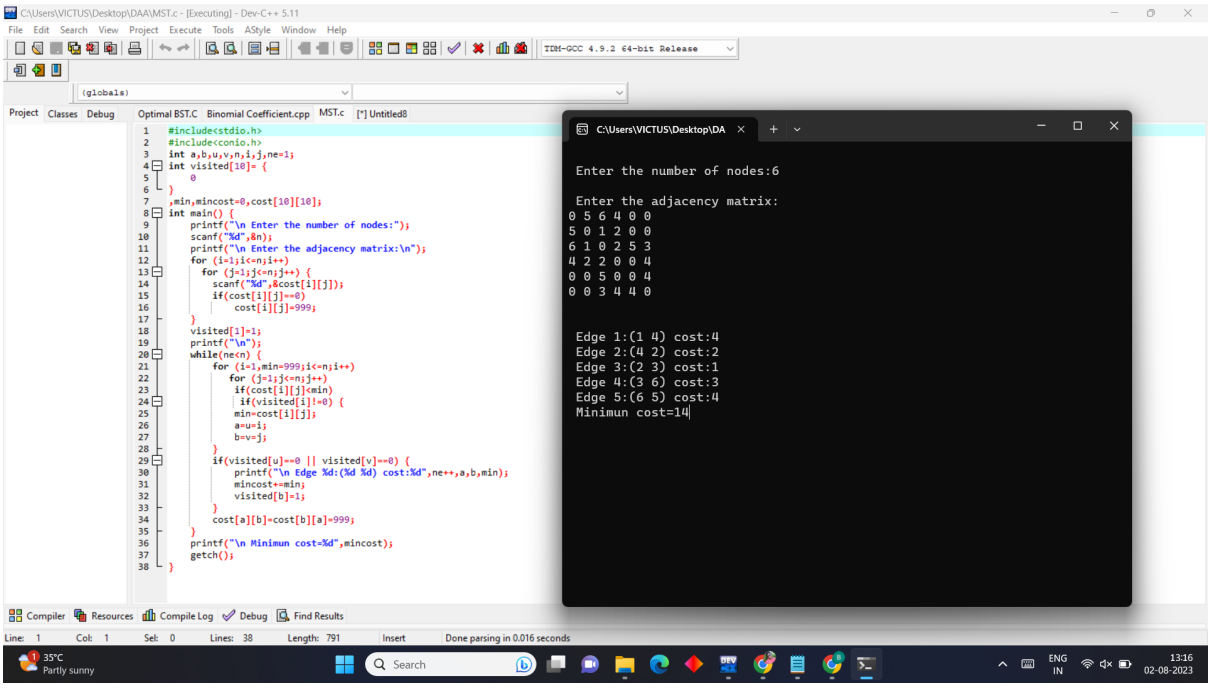
1. Write a program to perform MST using greedy techniques.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]= {
        0
}
,min,mincost=0,cost[10][10];
void main() {
        clrscr();
        printf("\n Enter the number of nodes:");
        scanf("%d",&n);
        printf("\n Enter the adjacency matrix:\n");
        for (i=1;i<=n;i++)
          for (j=1;j<=n;j++) {
                scanf("%d",&cost[i][j]);
                if(cost[i][j]==0)
                   cost[i][j]=999;
        }
        visited[1]=1;
        printf("\n");
        while(ne<n) {
                for (i=1,min=999;i<=n;i++)
                   for (j=1;j<=n;j++)
                    if(cost[i][j]<min)
                     if(visited[i]!=0) {
                          min=cost[i][j];
                          a=u=i;
                          b=v=j;
                }
                if(visited[u]==0 || visited[v]==0) {
                          printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
                          mincost+=min;
                          visited[b]=1;
                }
                cost[a][b]=cost[b][a]=999;
        }
```

```
        printf("\n Minimun cost=%d",mincost);
        getch();
}
```

OUTPUT



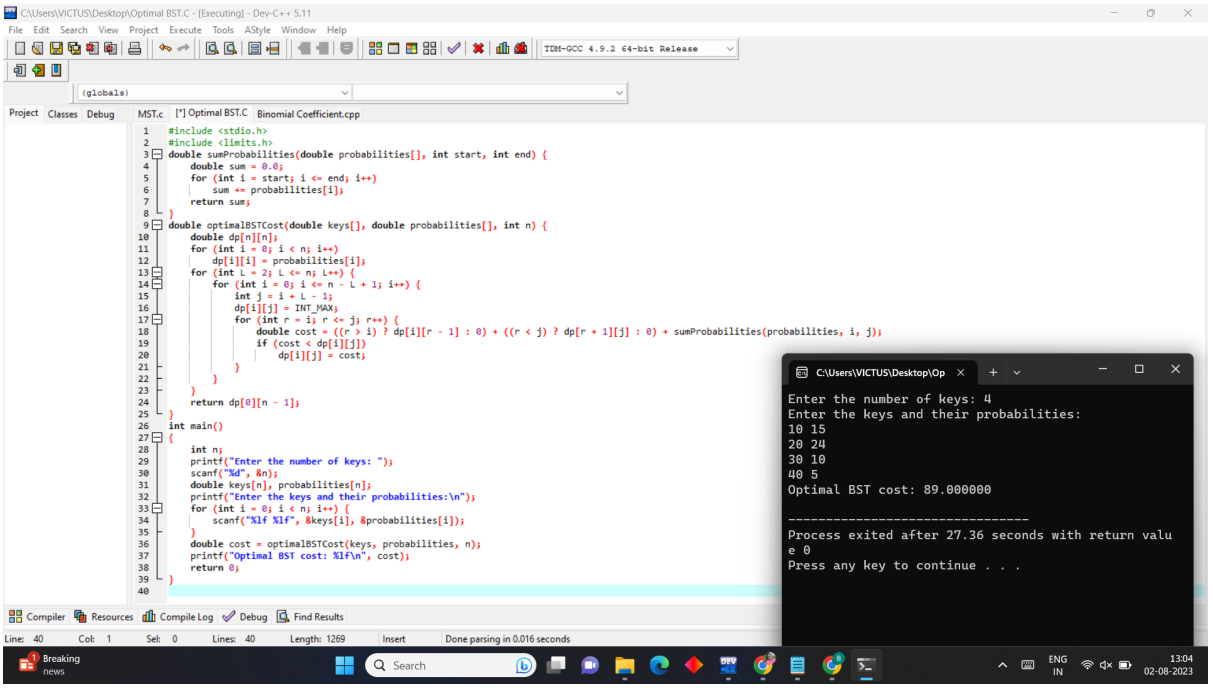
2. Using Dynamic programming concepts to find out the Optimal binary search tree

PROGRAM

```
#include <stdio.h>
#include <limits.h>
double sumProbabilities(double probabilities[], int start, int end) {
    double sum = 0.0;
    for (int i = start; i <= end; i++)
        sum += probabilities[i];
    return sum;
}
double optimalBSTCost(double keys[], double probabilities[], int n) {
    double dp[n][n];
    for (int i = 0; i < n; i++)
        dp[i][i] = probabilities[i];
    for (int L = 2; L <= n; L++) {
        for (int i = 0; i <= n - L + 1; i++) {
            int j = i + L - 1;
            dp[i][j] = INT_MAX;
            for (int r = i; r <= j; r++) {
                double cost = ((r > i) ? dp[i][r - 1] : 0) + ((r < j) ? dp[r + 1][j] : 0) +
sumProbabilities(probabilities, i, j);
                if (cost < dp[i][j])
                    dp[i][j] = cost;
```

```
            }
        }
    }
    return dp[0][n - 1];
}
int main()
{
    int n;
    printf("Enter the number of keys: ");
    scanf("%d", &n);
    double keys[n], probabilities[n];
    printf("Enter the keys and their probabilities:\n");
    for (int i = 0; i < n; i++) {
        scanf("%lf %lf", &keys[i], &probabilities[i]);
    }
    double cost = optimalBSTCost(keys, probabilities, n);
    printf("Optimal BST cost: %lf\n", cost);
    return 0;
}
```

OUTPUT



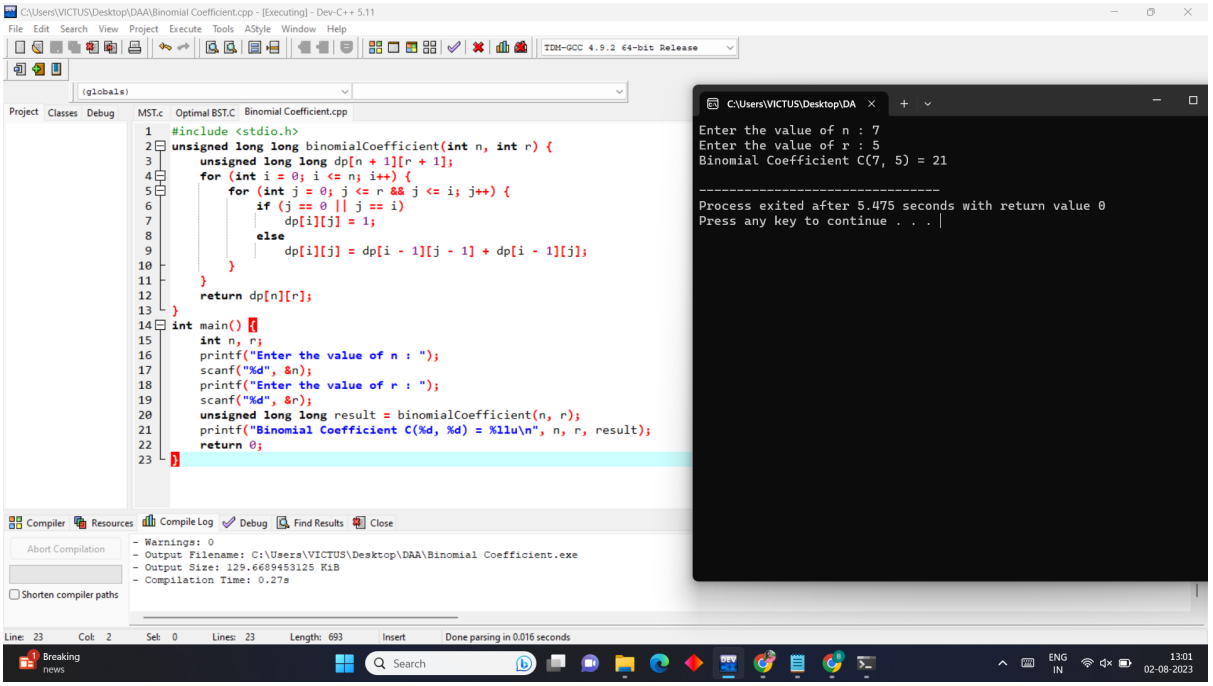
3. Using Dynamic programming techniques to find the binomial coefficient of a given number

PROGRAM

```
#include <stdio.h>
unsigned long long binomialCoefficient(int n, int r) {
    unsigned long long dp[n + 1][r + 1];
    for (int i = 0; i <= n; i++) {
```

```
        for (int j = 0; j <= r && j <= i; j++) {
            if (j == 0 || j == i)
                dp[i][j] = 1;
            else
                dp[i][j] = dp[i - 1][j - 1] + dp[i - 1][j];
        }
    }
    return dp[n][r];
}
int main() {
    int n, r;
    printf("Enter the value of n : ");
    scanf("%d", &n);
    printf("Enter the value of r : ");
    scanf("%d", &r);
    unsigned long long result = binomialCoefficient(n, r);
    printf("Binomial Coefficient C(%d, %d) = %llu\n", n, r, result);
    return 0;
}
```

OUTPUT



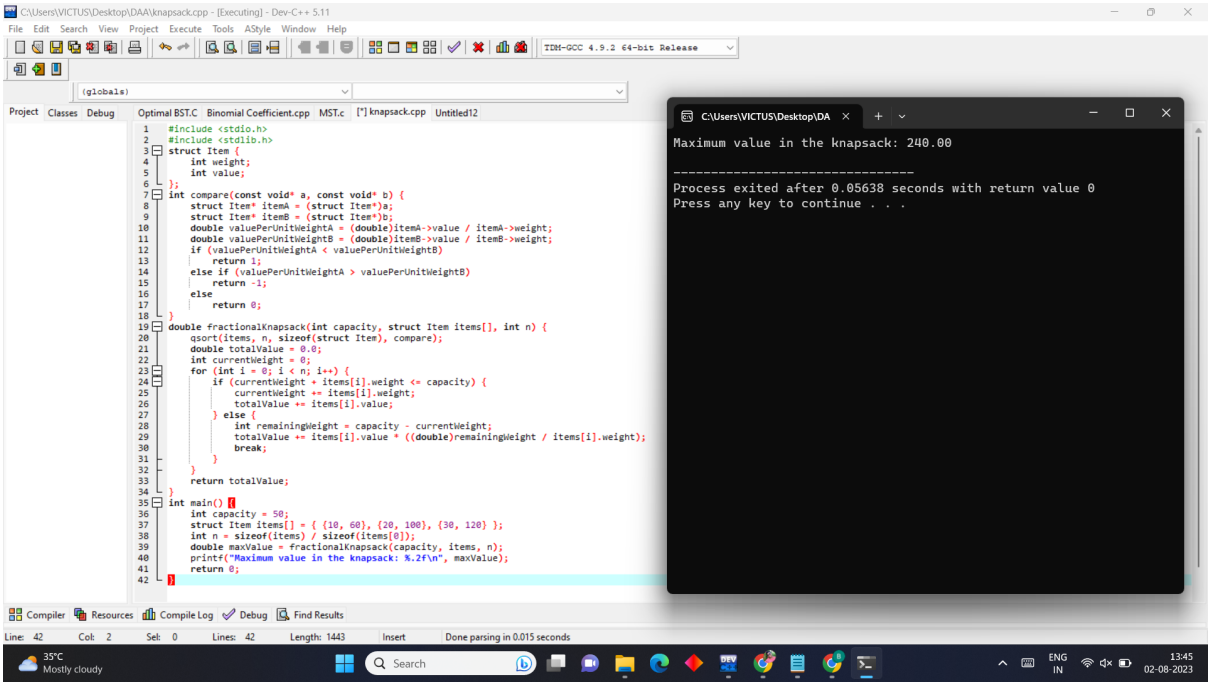
4.Write a program to perform the Knapsack problem using greedy techniques

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
struct Item {
    int weight;
```

```
    int value;
};
int compare(const void* a, const void* b) {
    struct Item* itemA = (struct Item*)a;
    struct Item* itemB = (struct Item*)b;
    double valuePerUnitWeightA = (double)itemA->value / itemA->weight;
    double valuePerUnitWeightB = (double)itemB->value / itemB->weight;
    if (valuePerUnitWeightA < valuePerUnitWeightB)
        return 1;
    else if (valuePerUnitWeightA > valuePerUnitWeightB)
        return -1;
    else
        return 0;
}
double fractionalKnapsack(int capacity, struct Item items[], int n) {
    qsort(items, n, sizeof(struct Item), compare);
    double totalValue = 0.0;
    int currentWeight = 0;
    for (int i = 0; i < n; i++) {
        if (currentWeight + items[i].weight <= capacity) {
            currentWeight += items[i].weight;
            totalValue += items[i].value;
        } else {
            int remainingWeight = capacity - currentWeight;
            totalValue += items[i].value * ((double)remainingWeight / items[i].weight);
            break;
        }
    }
    return totalValue;
}
int main() {
    int capacity = 50;
    struct Item items[] = { {10, 60}, {20, 100}, {30, 120} };
    int n = sizeof(items) / sizeof(items[0]);
    double maxValue = fractionalKnapsack(capacity, items, n);
    printf("Maximum value in the knapsack: %.2f\n", maxValue);
    return 0;
}
```

OUTPUT

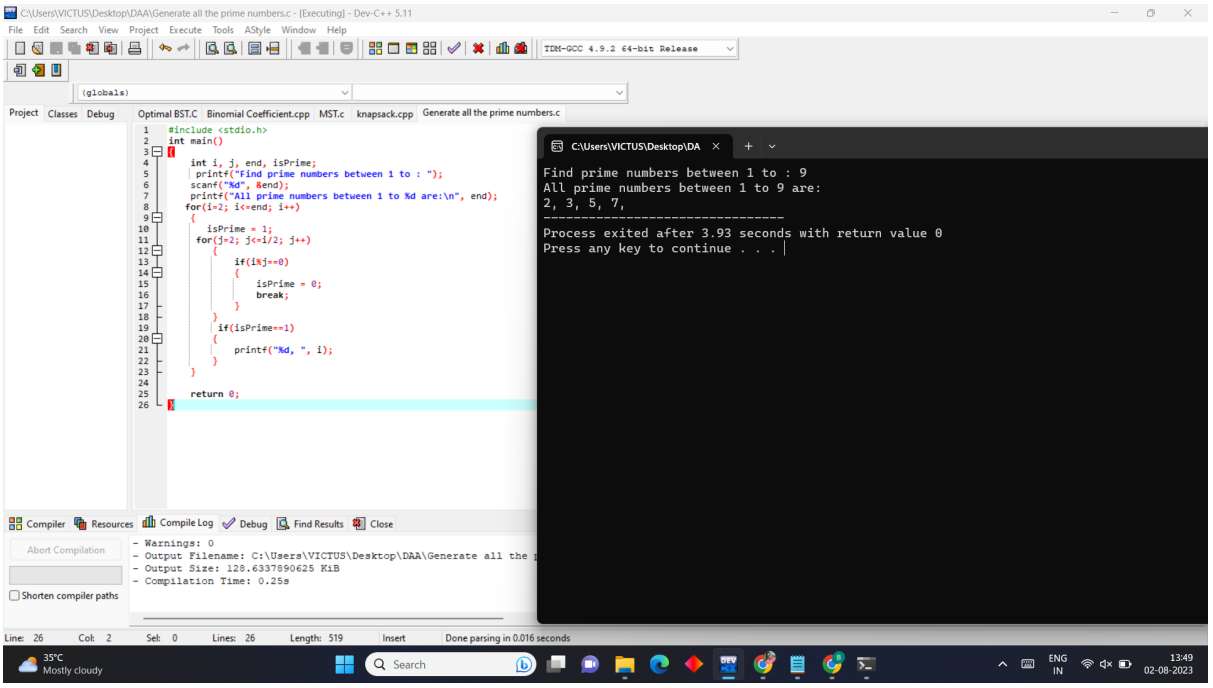


5.Write a program to generate all the prime numbers.

PROGRAM

```
#include <stdio.h>
int main()
{
   int i, j, end, isPrime;
           printf("Find prime numbers between 1 to : ");
   scanf("%d", &end);
   printf("All prime numbers between 1 to %d are:\n", end);
  for(i=2; i<=end; i++)
   {
     isPrime = 1;
    for(j=2; j<=i/2; j++)
      {
         if(i%j==0)
         {
            isPrime = 0;
            break;
         }
      }
       if(isPrime==1)
      {
         printf("%d, ", i);
      }
   }

   return 0;
}
```

OUTPUT



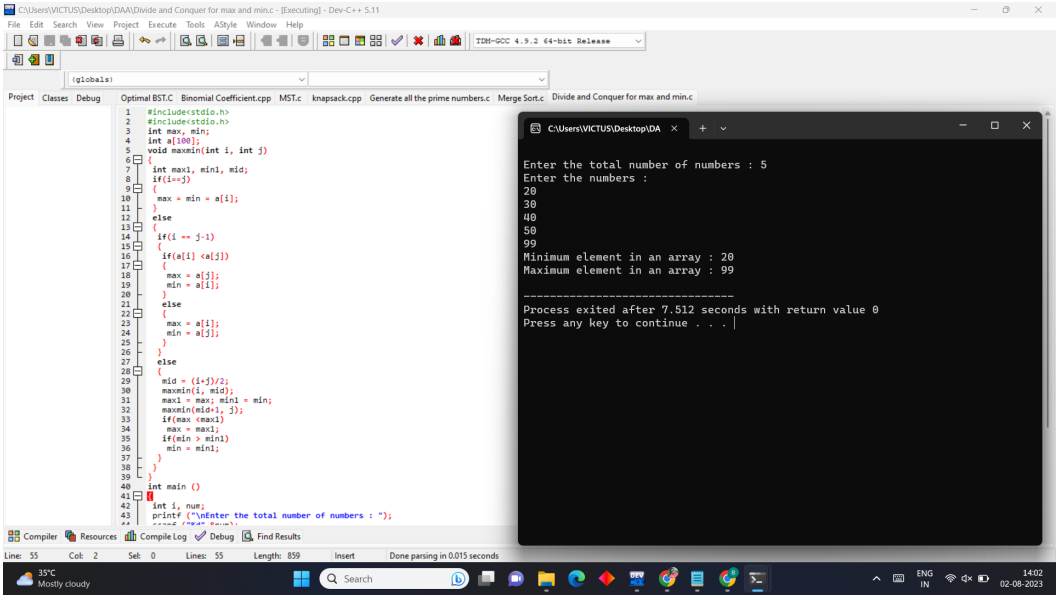
6.Using Divide and Conquer strategy to find Max and Min values in the list.

PROGRAM

```
#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;
}
```

OUTPUT



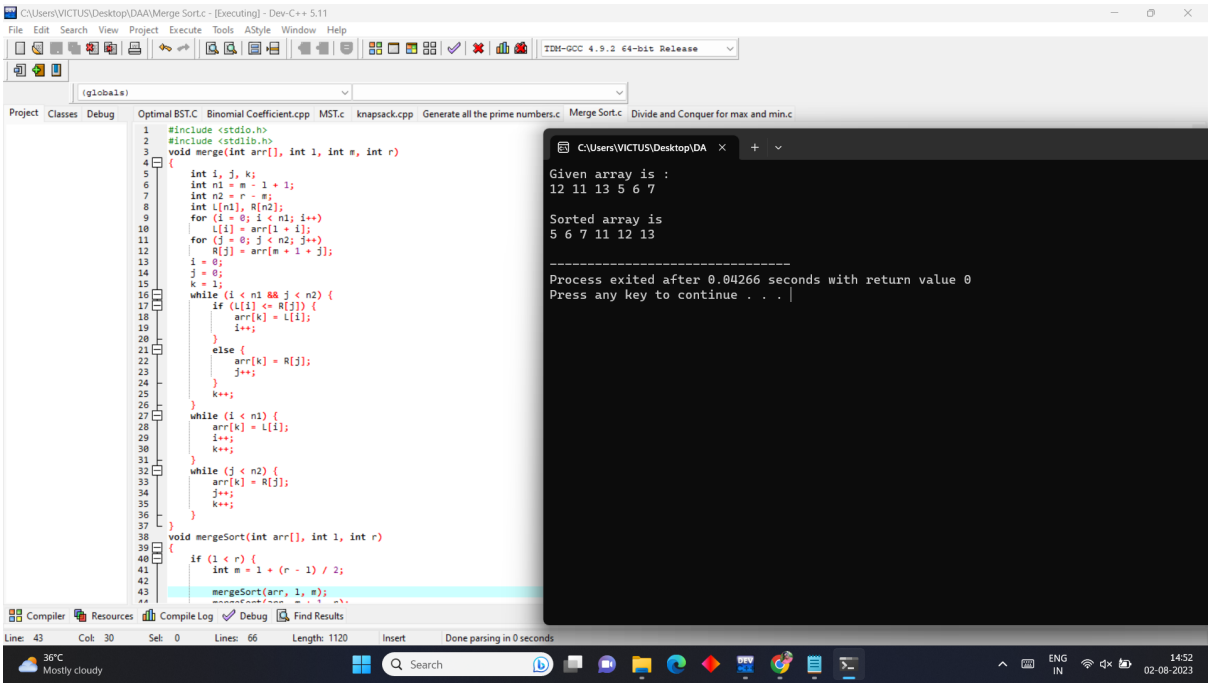
7.Write a program to perform Merge Sort.

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
void merge(int arr[], int l, int m, int r)
{
	int i, j, k;
	int n1 = m - l + 1;
	int n2 = r - m;
	int L[n1], R[n2];
	for (i = 0; i < n1; i++)
		L[i] = arr[l + i];
	for (j = 0; j < n2; j++)
		R[j] = arr[m + 1 + j];
	i = 0;
	j = 0;
	k = l;
	while (i < n1 && j < n2) {
		if (L[i] <= R[j]) {
			arr[k] = L[i];
			i++;
		}
		else {
			arr[k] = R[j];
			j++;
		}
		k++;
	}
	while (i < n1) {
		arr[k] = L[i];
		i++;
		k++;
	}
	while (j < n2) {
		arr[k] = R[j];
		j++;
		k++;
	}
}
void mergeSort(int arr[], int l, int r)
{
	if (l < r) {
		int m = l + (r - l) / 2;

		mergeSort(arr, l, m);
		mergeSort(arr, m + 1, r);
```

```
                merge(arr, l, m, r);
        }
}
void printArray(int A[], int size)
{
        int i;
        for (i = 0; i < size; i++)
                printf("%d ", A[i]);
        printf("\n");
}
int main()
{
        int arr[] = { 12, 11, 13, 5, 6, 7 };
        int arr_size = sizeof(arr) / sizeof(arr[0]);
        printf("Given array is :\n");
        printArray(arr, arr_size);
        mergeSort(arr, 0, arr_size - 1);
        printf("\nSorted array is \n");
        printArray(arr, arr_size);
        return 0;
}
```

OUTPUT



CSA0683-DESIGN AND ANALYSIS OF ALGORITHMS FOR SEARCHING ALGORITHMS

B.Nikhil(192111054)

DAY-3

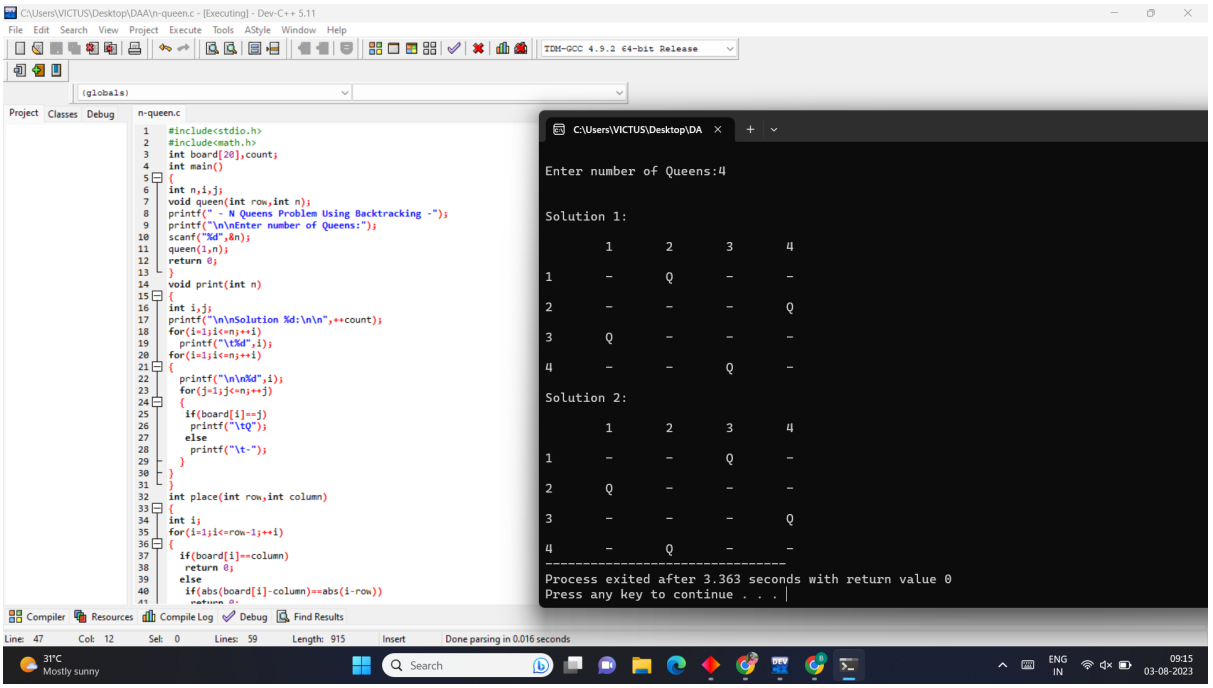
1.Write C program to perform n Queens Problem using backtracking

PROGRAM

```
#include<stdio.h>
#include<math.h>
int board[20],count;
int main()
{
int n,i,j;
void queen(int row,int n);
printf(" - N Queens Problem Using Backtracking -");
printf("\n\nEnter number of Queens:");
scanf("%d",&n);
queen(1,n);
return 0;
}
void print(int n)
{
int i,j;
printf("\n\nSolution %d:\n\n",++count);
for(i=1;i<=n;++i)
  printf("\t%d",i);
for(i=1;i<=n;++i)
{
  printf("\n\n%d",i);
  for(j=1;j<=n;++j)
  {
   if(board[i]==j)
    printf("\tQ");
   else
    printf("\t-");
  }
}
}
int place(int row,int column)
{
int i;
```

```
for(i=1;i<=row-1;++i)
{
  if(board[i]==column)
   return 0;
  else
   if(abs(board[i]-column)==abs(i-row))
    return 0;
}
return 1;
}
void queen(int row,int n)
{
int column;
for(column=1;column<=n;++column)
{
  if(place(row,column))
  {
   board[row]=column;
   if(row==n)
    print(n);
   else
    queen(row+1,n);
  }
}
}
```

OUTPUT

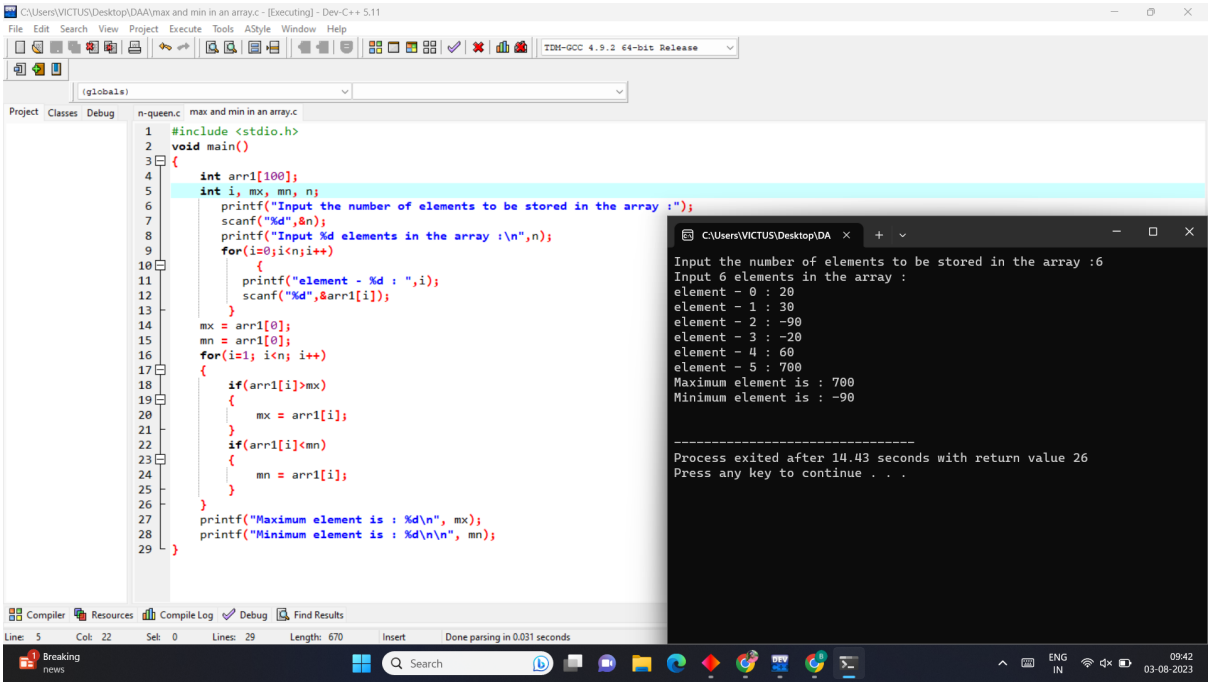


2.Write C program to print minimum and maximum value sequence for all the numbers in a list.

PROGRAM

```
#include <stdio.h>
void main()
{
   int arr1[100];
   int i, mx, mn, n;
     printf("Input the number of elements to be stored in the array :");
     scanf("%d",&n);
     printf("Input %d elements in the array :\n",n);
     for(i=0;i<n;i++)
         {
             printf("element - %d : ",i);
             scanf("%d",&arr1[i]);
           }
   mx = arr1[0];
   mn = arr1[0];
   for(i=1; i<n; i++)
   {
      if(arr1[i]>mx)
      {
         mx = arr1[i];
      }
      if(arr1[i]<mn)
      {
         mn = arr1[i];
      }
   }
   printf("Maximum element is : %d\n", mx);
   printf("Minimum element is : %d\n\n", mn);
}
```

OUTPUT



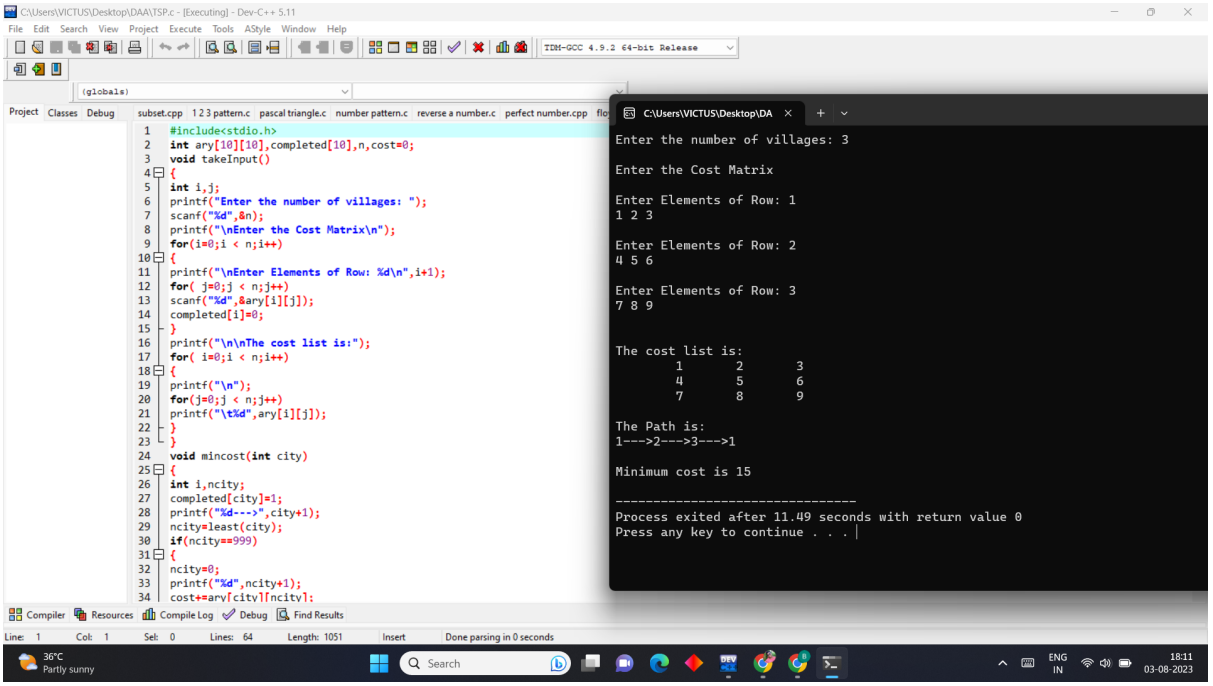
3.Write C program to perform traveling salesman problem using dynamic programming

PROGRAM

```
#include<stdio.h>
int ary[10][10],completed[10],n,cost=0;
void takeInput()
{
int i,j;
printf("Enter the number of villages: ");
scanf("%d",&n);
printf("\nEnter the Cost Matrix\n");
for(i=0;i < n;i++)
{
printf("\nEnter Elements of Row: %d\n",i+1);
for( j=0;j < n;j++)
scanf("%d",&ary[i][j]);
completed[i]=0;
}
printf("\n\nThe cost list is:");
for( i=0;i < n;i++)
{
printf("\n");
for(j=0;j < n;j++)
printf("\t%d",ary[i][j]);
}
```

```
}
void mincost(int city)
{
int i,ncity;
completed[city]=1;
printf("%d--->",city+1);
ncity=least(city);
if(ncity==999)
{
ncity=0;
printf("%d",ncity+1);
cost+=ary[city][ncity];
return;
}
mincost(ncity);
}
int least(int c)
{
int i,nc=999;
int min=999,kmin;
for(i=0;i < n;i++)
{
if((ary[c][i]!=0)&&(completed[i]==0))
if(ary[c][i]+ary[i][c] < min)
{
min=ary[i][0]+ary[c][i];
kmin=ary[c][i];
nc=i;
}
}
if(min!=999)
cost+=kmin;
return nc;
}
int main()
{
takeInput();
printf("\n\nThe Path is:\n");
mincost(0);
printf("\n\nMinimum cost is %d\n ",cost);
return 0;
}
```

OUTPUT



4.Write C program for the given pattern

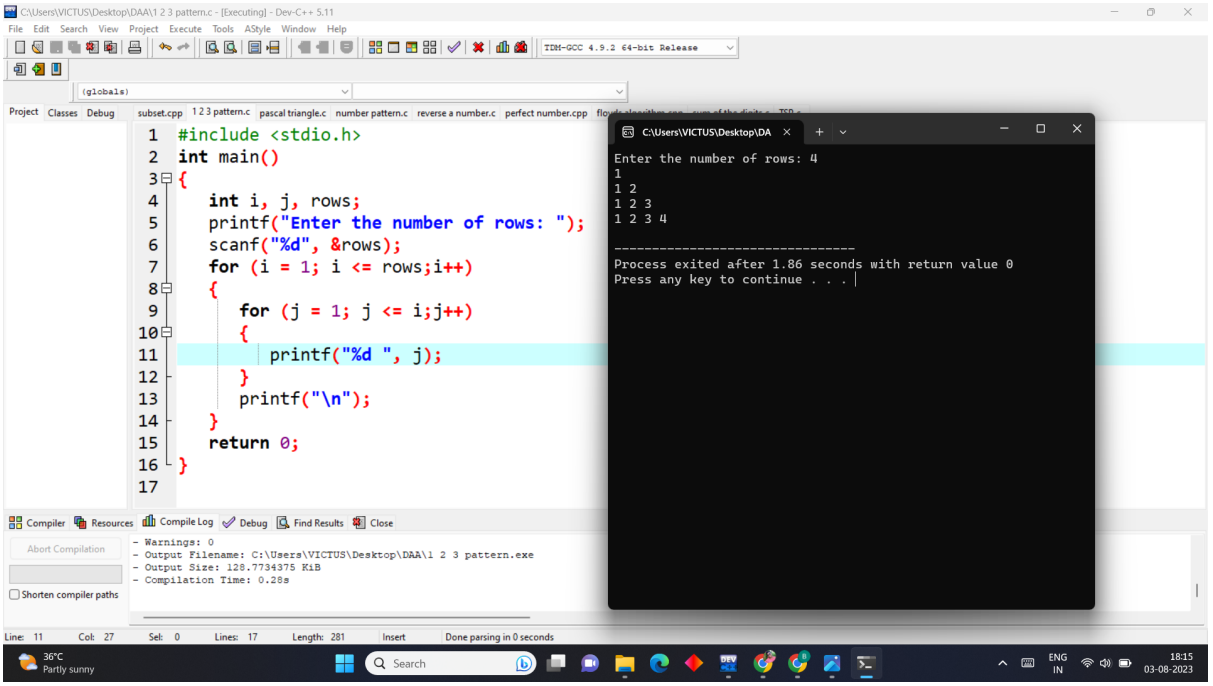
If n=4

1
1 2
1 2 3
1 2 3 4

PROGRAM:

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

OUTPUT



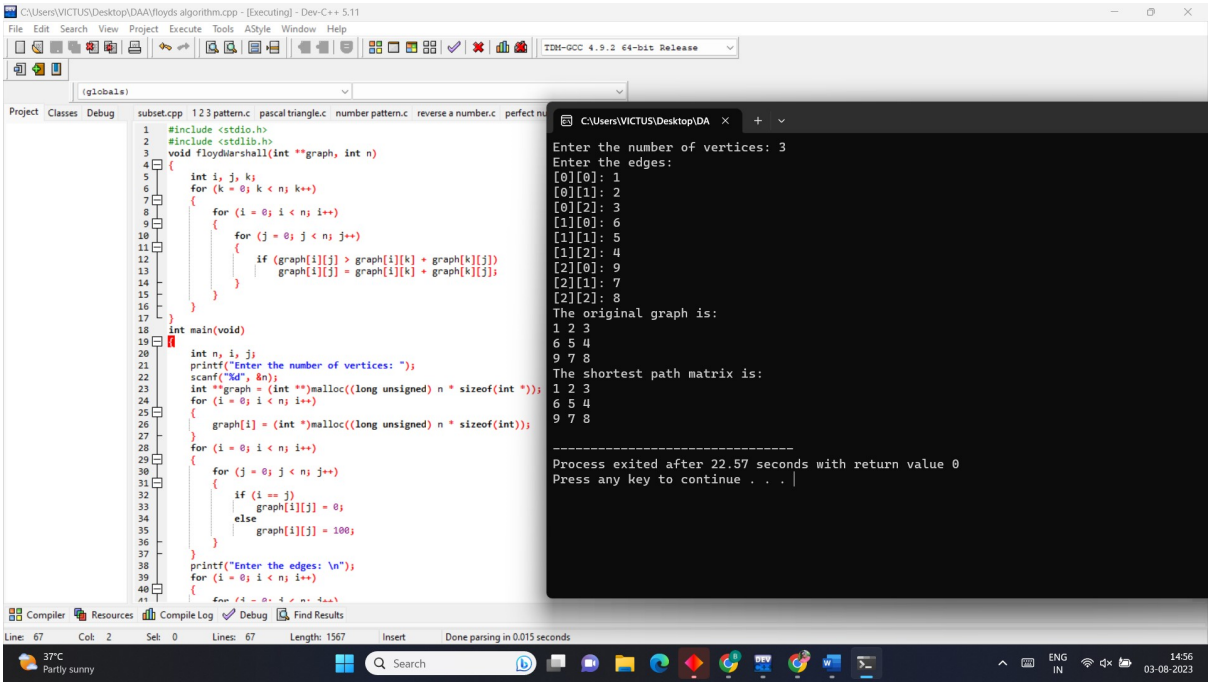
5.Write C program to perform Floyd's algorithm

PROGRAM

```
#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;
}
```

OUTPUT

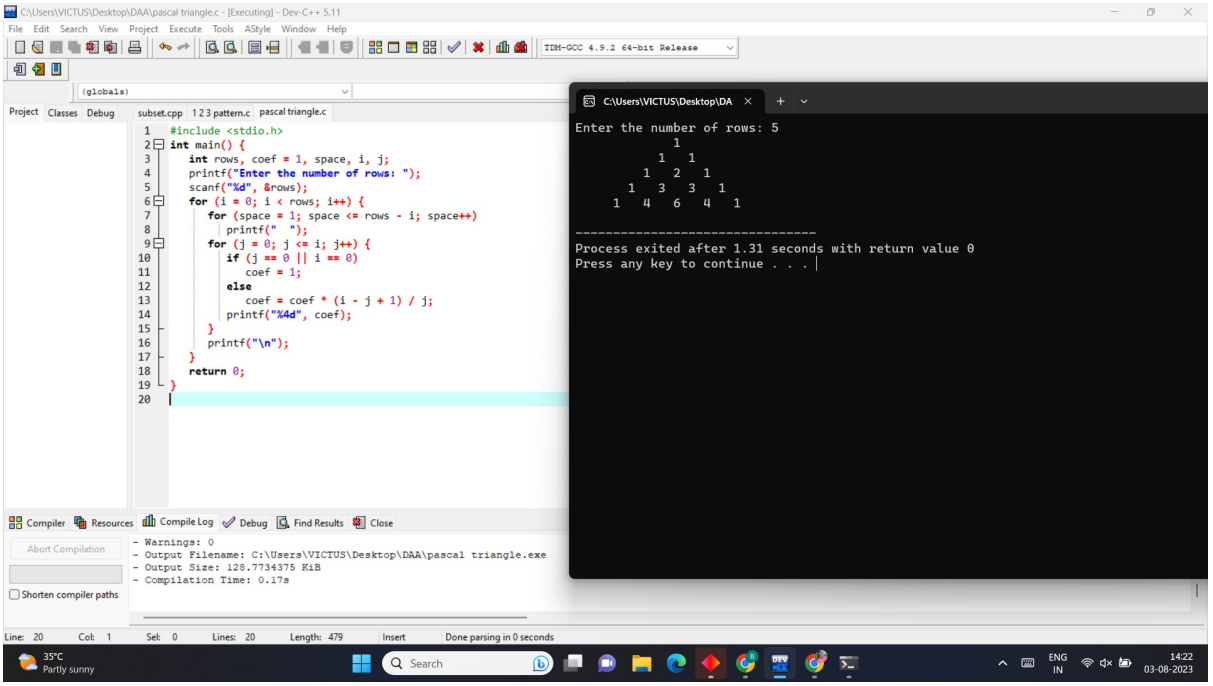


6. Write C program for pascal triangle.

PROGRAM

```
#include <stdio.h>
int main() {
  int rows, coef = 1, space, i, j;
  printf("Enter the number of rows: ");
  scanf("%d", &rows);
  for (i = 0; i < rows; i++) {
    for (space = 1; space <= rows - i; space++)
      printf("  ");
    for (j = 0; j <= i; j++) {
      if (j == 0 || i == 0)
        coef = 1;
      else
        coef = coef * (i - j + 1) / j;
      printf("%4d", coef);
    }
    printf("\n");
  }
  return 0;
}
```

OUTPUT



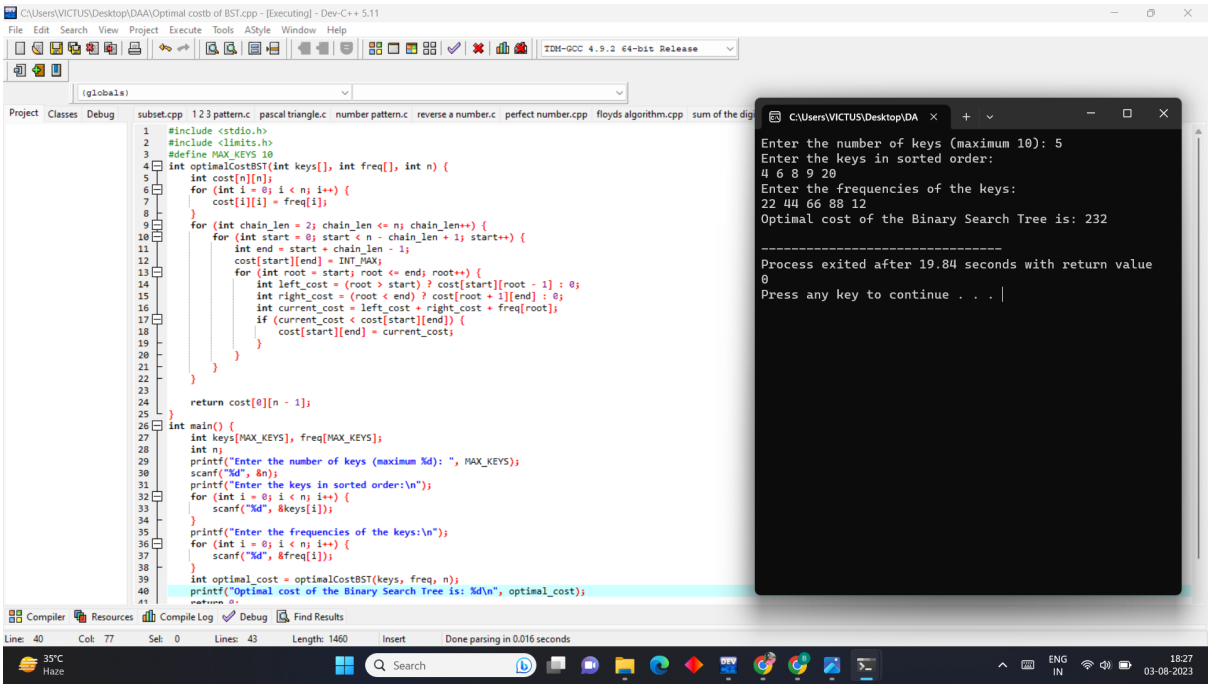
7.Write a program to find the optimal cost by using appropriate algorithm

PROGRAM

```
#include <stdio.h>
#include <limits.h>
#define MAX_KEYS 10
int optimalCostBST(int keys[], int freq[], int n) {
    int cost[n][n];
    for (int i = 0; i < n; i++) {
        cost[i][i] = freq[i];
    }
    for (int chain_len = 2; chain_len <= n; chain_len++) {
        for (int start = 0; start < n - chain_len + 1; start++) {
            int end = start + chain_len - 1;
            cost[start][end] = INT_MAX;
            for (int root = start; root <= end; root++) {
                int left_cost = (root > start) ? cost[start][root - 1] : 0;
                int right_cost = (root < end) ? cost[root + 1][end] : 0;
                int current_cost = left_cost + right_cost + freq[root];
                if (current_cost < cost[start][end]) {
                    cost[start][end] = current_cost;
                }
            }
        }
    }
```

```
    return cost[0][n - 1];
}
int main() {
    int keys[MAX_KEYS], freq[MAX_KEYS];
    int n;
    printf("Enter the number of keys (maximum %d): ", MAX_KEYS);
    scanf("%d", &n);
    printf("Enter the keys in sorted order:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &keys[i]);
    }
    printf("Enter the frequencies of the keys:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &freq[i]);
    }
    int optimal_cost = optimalCostBST(keys, freq, n);
    printf("Optimal cost of the Binary Search Tree is: %d\n", optimal_cost);
    return 0;
}
```

OUTPUT



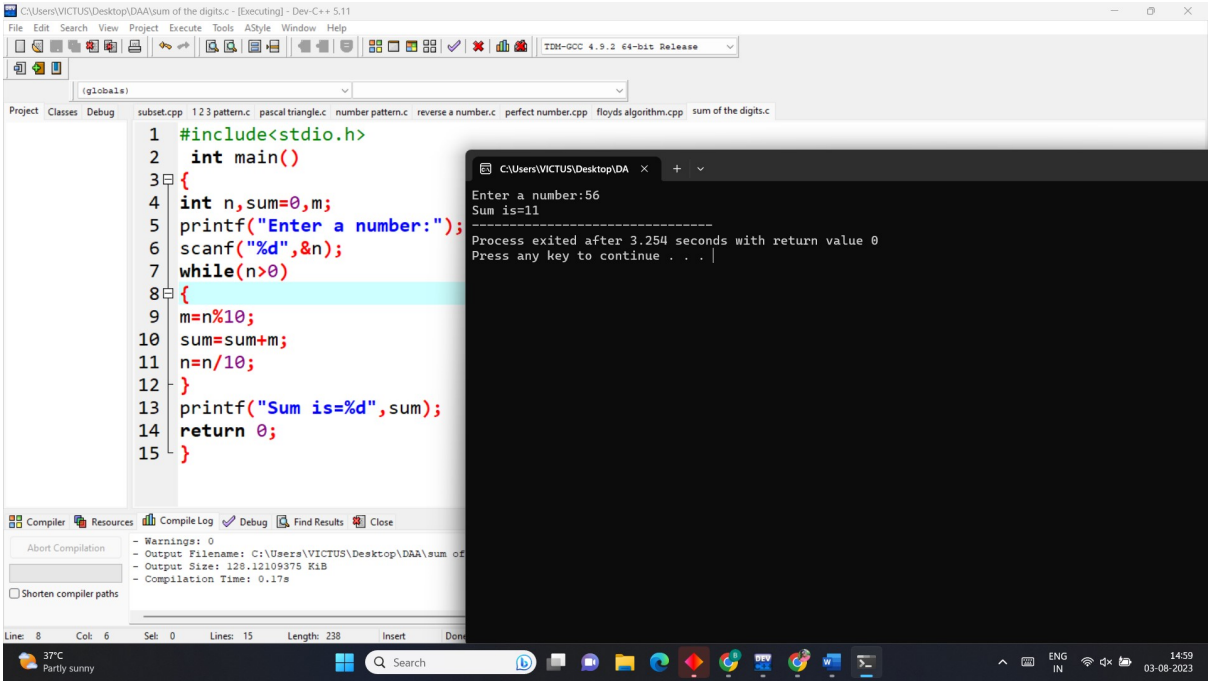
8.Write a program to find the sum of digits.

PROGRAM

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

```
 int main()
{
int n,sum=0,m;
printf("Enter a number:");
scanf("%d",&n);
while(n>0)
{
m=n%10;
sum=sum+m;
n=n/10;
}
printf("Sum is=%d",sum);
return 0;
}
```

OUTPUT



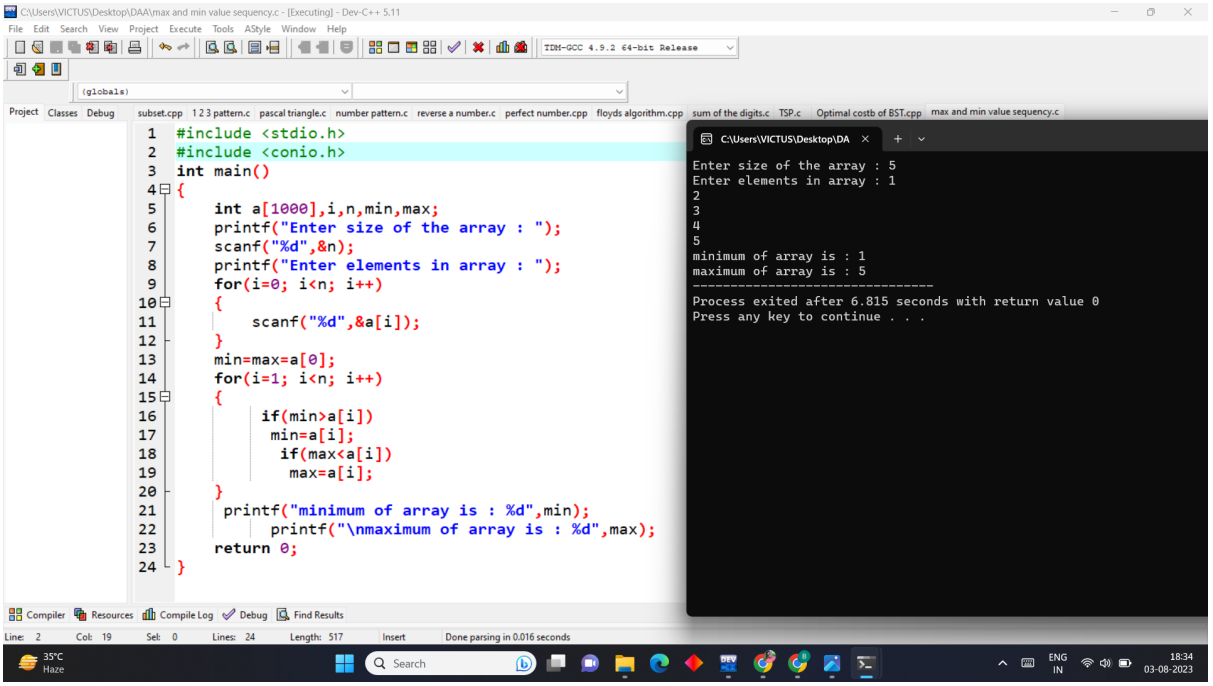
9.Write a program to print minimum and maximum value sequency for all the numbers in a list.

PROGRAM

```
#include <stdio.h>
#include <conio.h>
int main()
{
   int a[1000],i,n,min,max;
   printf("Enter size of the array : ");
```

```
    scanf("%d",&n);
    printf("Enter elements in array : ");
    for(i=0; i<n; i++)
    {
       scanf("%d",&a[i]);
    }
    min=max=a[0];
    for(i=1; i<n; i++)
    {
        if(min>a[i])
                    min=a[i];
                     if(max<a[i])
                      max=a[i];
    }
     printf("minimum of array is : %d",min);
          printf("\nmaximum of array is : %d",max);
    return 0;
}
```

OUTPUT



10.Write a program to perform n Queens Problem using backtracking.

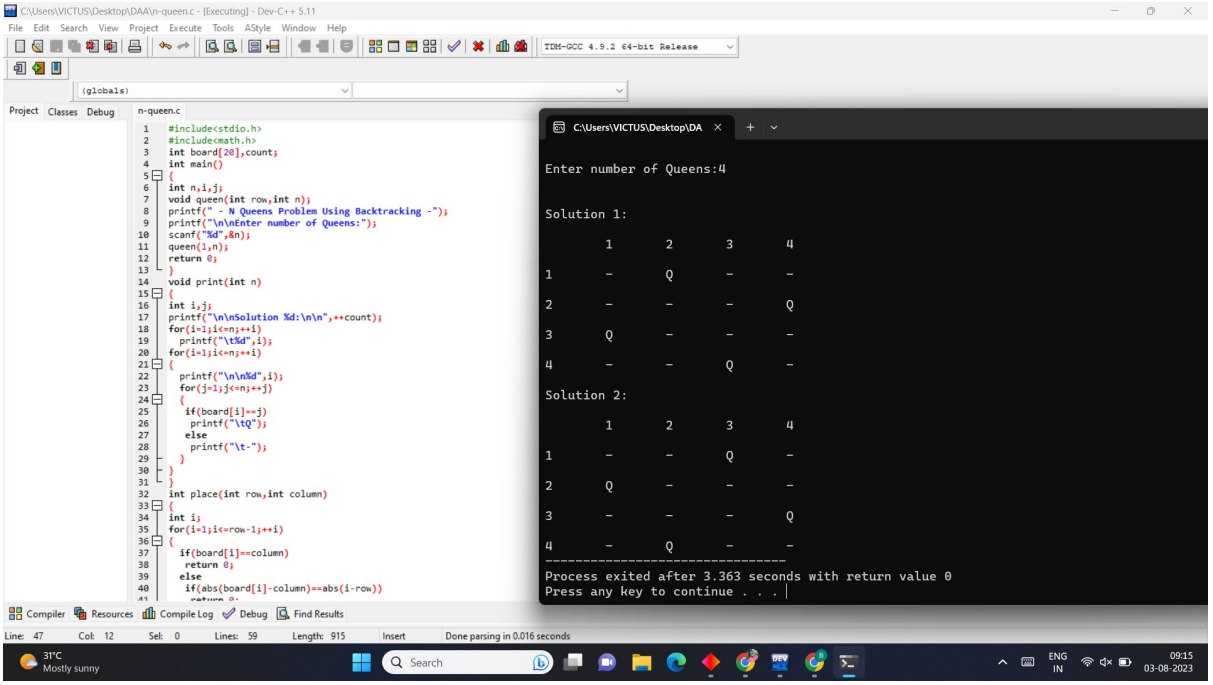
PROGRAM

```
#include<stdio.h>
#include<math.h>
int board[20],count;
```

```
int main()
{
int n,i,j;
void queen(int row,int n);
printf(" - N Queens Problem Using Backtracking -");
printf("\n\nEnter number of Queens:");
scanf("%d",&n);
queen(1,n);
return 0;
}
void print(int n)
{
int i,j;
printf("\n\nSolution %d:\n\n",++count);
for(i=1;i<=n;++i)
  printf("\t%d",i);
for(i=1;i<=n;++i)
{
  printf("\n\n%d",i);
  for(j=1;j<=n;++j)
  {
   if(board[i]==j)
    printf("\tQ");
   else
    printf("\t-");
  }
}
}
int place(int row,int column)
{
int i;
for(i=1;i<=row-1;++i)
{
  if(board[i]==column)
   return 0;
  else
   if(abs(board[i]-column)==abs(i-row))
    return 0;
}
return 1;
}
void queen(int row,int n)
{
int column;
```

```
for(column=1;column<=n;++column)
{
 if(place(row,column))
  {
   board[row]=column;
   if(row==n)
    print(n);
   else
    queen(row+1,n);
  }
}
}
```

OUTPUT



CSA0683-DESIGN AND ANALYSIS OF ALGORITHMS FOR SEARCHING ALGORITHMS

B.Nikhil(192111054)

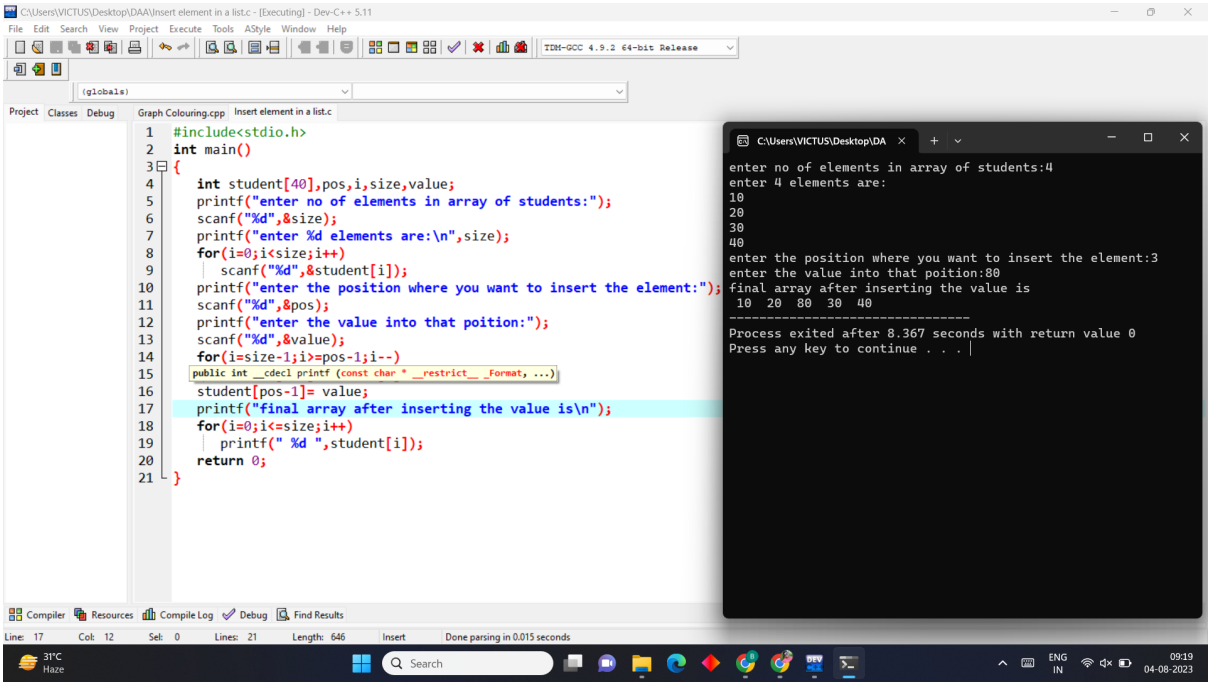
DAY-4

1.Write C program to insert a number in a list.

PROGRAM

```
#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 position:");
  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 ",student[i]);
  return 0;
}
```

OUTPUT



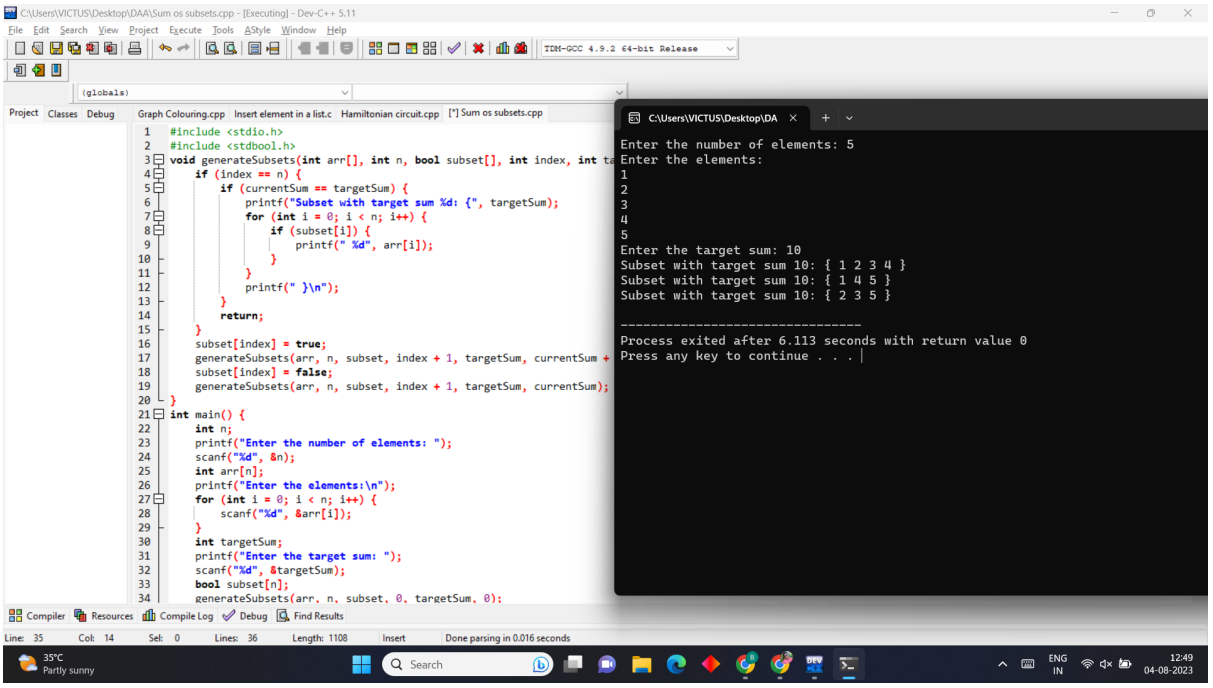
2.Write C program to perform a sum of subsets problem using backtracking.

PROGRAM

```
#include <stdio.h>
#include <stdbool.h>
void generateSubsets(int arr[], int n, bool subset[], int index, int targetSum, int currentSum) {
    if (index == n) {
        if (currentSum == targetSum) {
            printf("Subset with target sum %d: {", targetSum);
            for (int i = 0; i < n; i++) {
                if (subset[i]) {
                    printf(" %d", arr[i]);
                }
            }
            printf(" }\n");
        }
        return;
    }
    subset[index] = true;
    generateSubsets(arr, n, subset, index + 1, targetSum, currentSum + arr[index]);
    subset[index] = false;
    generateSubsets(arr, n, subset, index + 1, targetSum, currentSum);
}
int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter the elements:\n");
```

```
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    int targetSum;
    printf("Enter the target sum: ");
    scanf("%d", &targetSum);
    bool subset[n];
    generateSubsets(arr, n, subset, 0, targetSum, 0);
    return 0;
}
```

OUTPUT



3.Write C program to perform graph colouring problem using backtracking.

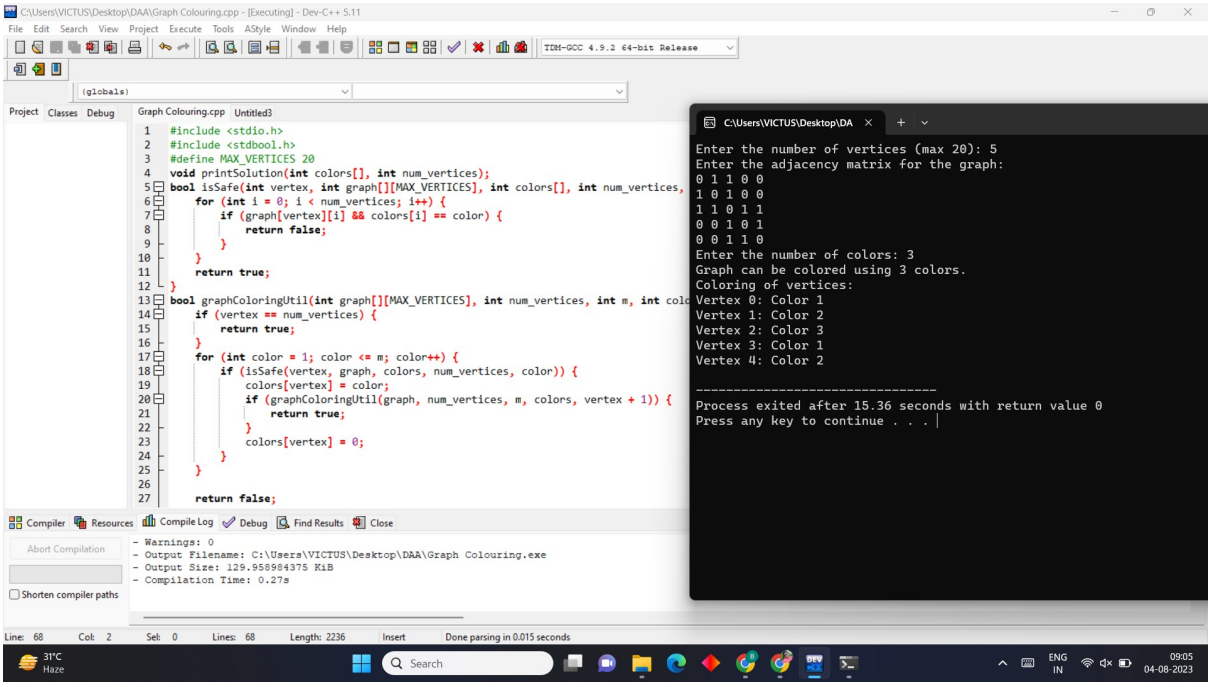
PROGRAM

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_VERTICES 20
void printSolution(int colors[], int num_vertices);
bool isSafe(int vertex, int graph[][MAX_VERTICES], int colors[], int num_vertices, int color) {
    for (int i = 0; i < num_vertices; i++) {
        if (graph[vertex][i] && colors[i] == color) {
            return false;
        }
    }
    return true;
}
```

```
bool graphColoringUtil(int graph[][MAX_VERTICES], int num_vertices, int m, int colors[], int
vertex) {
    if (vertex == num_vertices) {
        return true;
    }
    for (int color = 1; color <= m; color++) {
        if (isSafe(vertex, graph, colors, num_vertices, color)) {
            colors[vertex] = color;
            if (graphColoringUtil(graph, num_vertices, m, colors, vertex + 1)) {
                return true;
            }
            colors[vertex] = 0;
        }
    }

    return false;
}
bool graphColoring(int graph[][MAX_VERTICES], int num_vertices, int m) {
    int colors[MAX_VERTICES];
    for (int i = 0; i < num_vertices; i++) {
        colors[i] = 0;
    }
    if (!graphColoringUtil(graph, num_vertices, m, colors, 0)) {
        return false;
    }
    printf("Graph can be colored using %d colors.\n", m);
    printf("Coloring of vertices:\n");
    printSolution(colors, num_vertices);
    return true;
}
void printSolution(int colors[], int num_vertices) {
    for (int i = 0; i < num_vertices; i++) {
        printf("Vertex %d: Color %d\n", i, colors[i]);
    }
}
int main() {
    int num_vertices, m;
    printf("Enter the number of vertices (max %d): ", MAX_VERTICES);
    scanf("%d", &num_vertices);
    int graph[MAX_VERTICES][MAX_VERTICES];
    printf("Enter the adjacency matrix for the graph:\n");
    for (int i = 0; i < num_vertices; i++) {
        for (int j = 0; j < num_vertices; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
    printf("Enter the number of colors: ");
    scanf("%d", &m);
```

```
    if (m < 1) {
        printf("Number of colors should be at least 1.\n");
        return 1;
    }
    if (!graphColoring(graph, num_vertices, m)) {
        printf("Graph cannot be colored with the given constraints.\n");
    }
    return 0;
}
```

OUTPUT



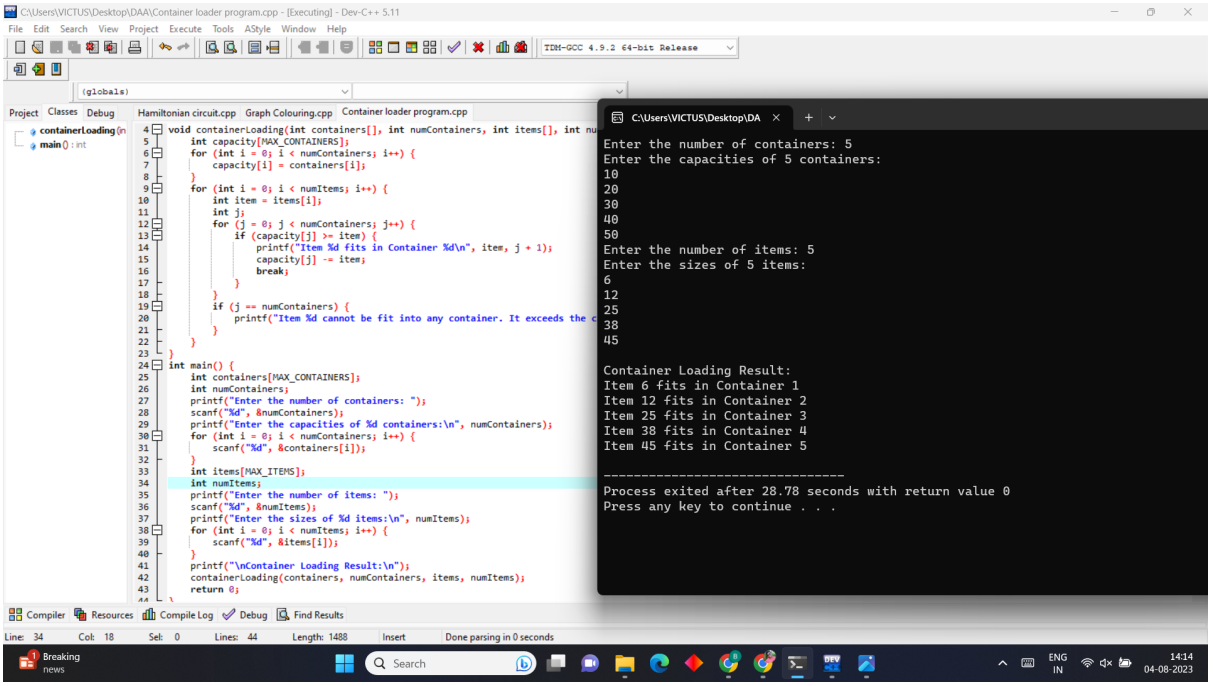
4.Write C program to compute container loader Problem.

PROGRAM

```
#include <stdio.h>
#define MAX_CONTAINERS 100
#define MAX_ITEMS 100
void containerLoading(int containers[], int numContainers, int items[], int numItems) {
    int capacity[MAX_CONTAINERS];
    for (int i = 0; i < numContainers; i++) {
        capacity[i] = containers[i];
    }
    for (int i = 0; i < numItems; i++) {
        int item = items[i];
        int j;
        for (j = 0; j < numContainers; j++) {
```

```
            if (capacity[j] >= item) {
                printf("Item %d fits in Container %d\n", item, j + 1);
                capacity[j] -= item;
                break;
            }
        }
        if (j == numContainers) {
            printf("Item %d cannot be fit into any container. It exceeds the capacity.\n", item);
        }
    }
}
int main() {
    int containers[MAX_CONTAINERS];
    int numContainers;
    printf("Enter the number of containers: ");
    scanf("%d", &numContainers);
    printf("Enter the capacities of %d containers:\n", numContainers);
    for (int i = 0; i < numContainers; i++) {
        scanf("%d", &containers[i]);
    }
    int items[MAX_ITEMS];
    int numItems;
    printf("Enter the number of items: ");
    scanf("%d", &numItems);
    printf("Enter the sizes of %d items:\n", numItems);
    for (int i = 0; i < numItems; i++) {
        scanf("%d", &items[i]);
    }
    printf("\nContainer Loading Result:\n");
    containerLoading(containers, numContainers, items, numItems);
    return 0;
}
```

OUTPUT

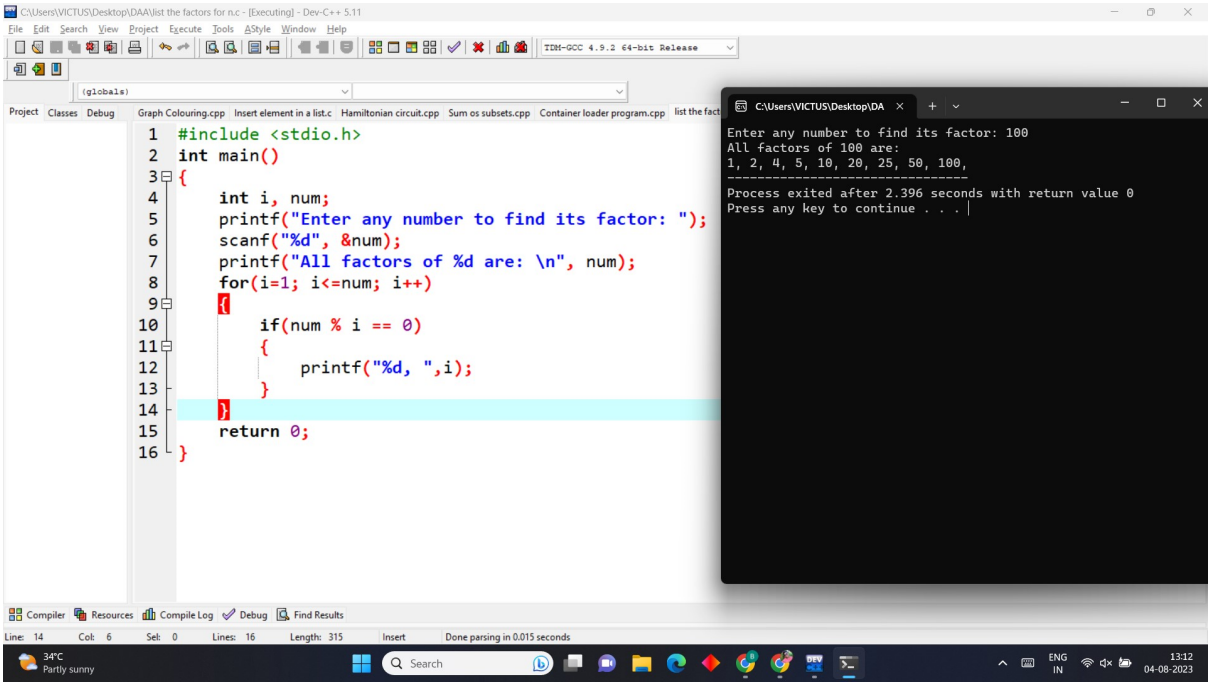


5.Write a program to generate the list of all factors for n value.

PROGRAM

```
#include <stdio.h>
int main()
{
   int i, num;
   printf("Enter any number to find its factor: ");
   scanf("%d", &num);
   printf("All factors of %d are: \n", num);
   for(i=1; i<=num; i++)
   {
      if(num % i == 0)
      {
         printf("%d, ",i);
      }
   }
   return 0;
}
```

OUTPUT



6.Write a program to perform Assignment problem using branch and bound

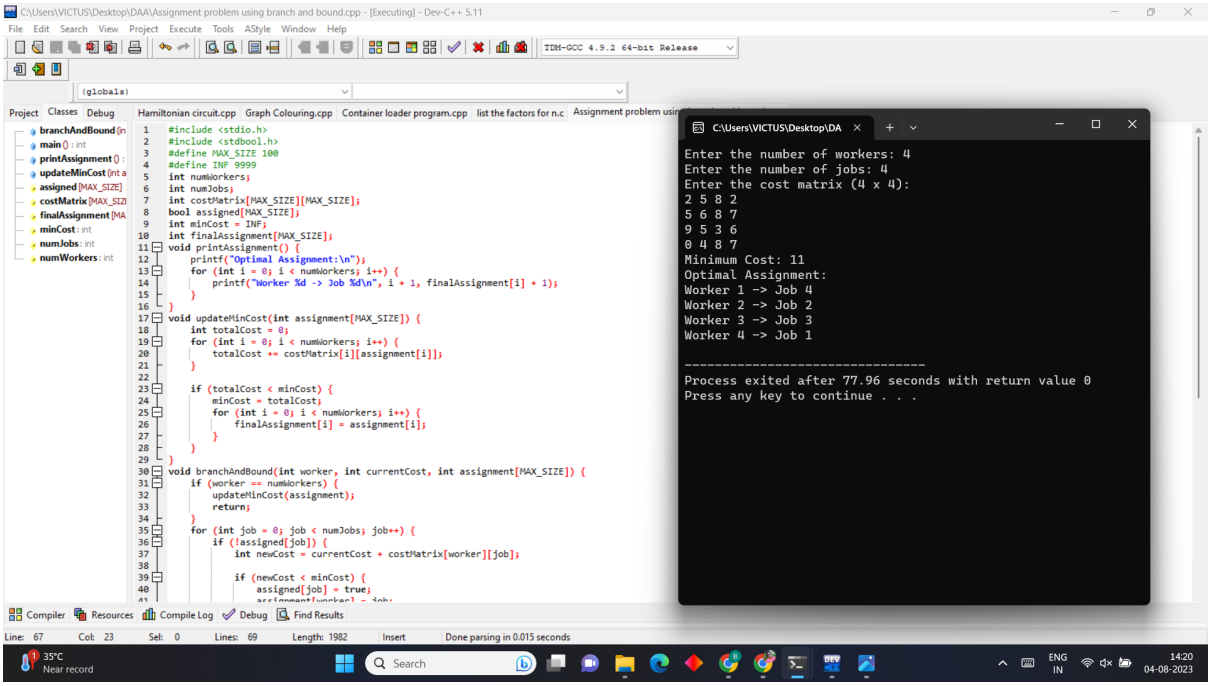
PROGRAM

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_SIZE 100
#define INF 9999
int numWorkers;
int numJobs;
int costMatrix[MAX_SIZE][MAX_SIZE];
bool assigned[MAX_SIZE];
int minCost = INF;
int finalAssignment[MAX_SIZE];
void printAssignment() {
    printf("Optimal Assignment:\n");
    for (int i = 0; i < numWorkers; i++) {
        printf("Worker %d -> Job %d\n", i + 1, finalAssignment[i] + 1);
    }
}
void updateMinCost(int assignment[MAX_SIZE]) {
    int totalCost = 0;
    for (int i = 0; i < numWorkers; i++) {
        totalCost += costMatrix[i][assignment[i]];
    }

    if (totalCost < minCost) {
```

```
        minCost = totalCost;
        for (int i = 0; i < numWorkers; i++) {
            finalAssignment[i] = assignment[i];
        }
    }
}
void branchAndBound(int worker, int currentCost, int assignment[MAX_SIZE]) {
    if (worker == numWorkers) {
        updateMinCost(assignment);
        return;
    }
    for (int job = 0; job < numJobs; job++) {
        if (!assigned[job]) {
            int newCost = currentCost + costMatrix[worker][job];

            if (newCost < minCost) {
                assigned[job] = true;
                assignment[worker] = job;

                branchAndBound(worker + 1, newCost, assignment);

                assigned[job] = false;
            }
        }
    }
}
int main() {
    printf("Enter the number of workers: ");
    scanf("%d", &numWorkers);
    printf("Enter the number of jobs: ");
    scanf("%d", &numJobs);
    printf("Enter the cost matrix (%d x %d):\n", numWorkers, numJobs);
    for (int i = 0; i < numWorkers; i++) {
        for (int j = 0; j < numJobs; j++) {
            scanf("%d", &costMatrix[i][j]);
        }
    }
    int assignment[MAX_SIZE];
    for (int i = 0; i < numWorkers; i++) {
        assigned[i] = false;
    }
    branchAndBound(0, 0, assignment);
    printf("Minimum Cost: %d\n", minCost);
    printAssignment();
```

```
    return 0;
}
```

OUTPUT



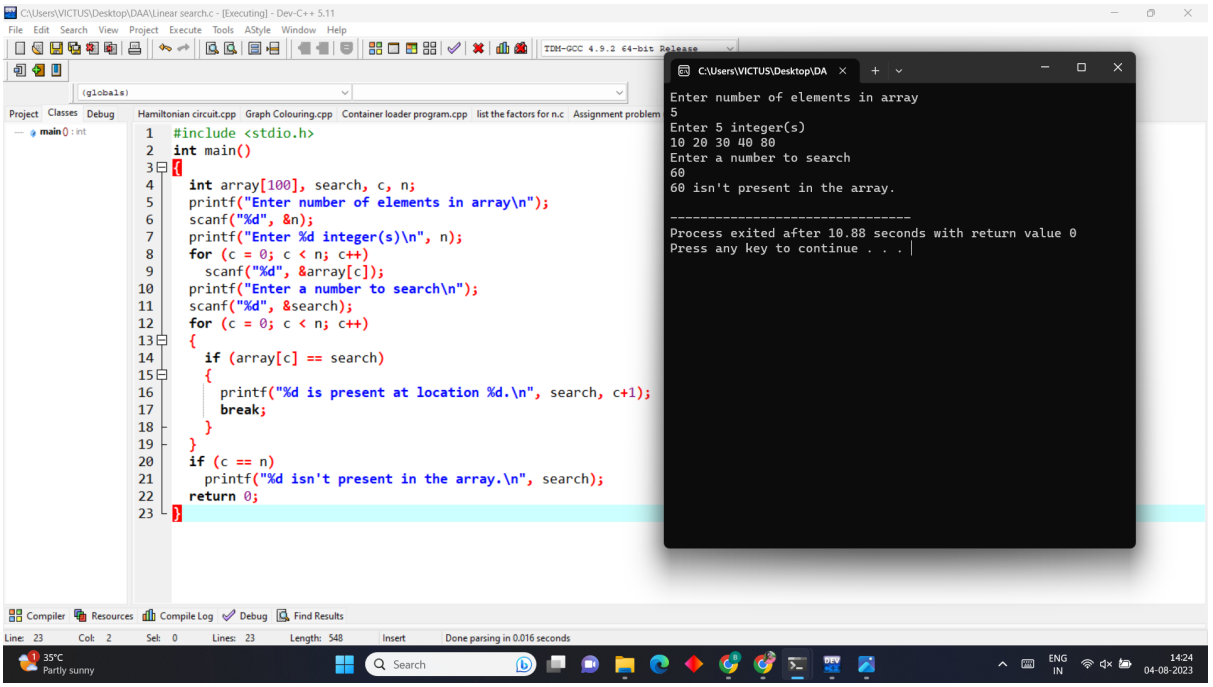
7.Write a program to perform linear search.

PROGRAM

```
#include <stdio.h>
int main()
{
  int array[100], search, c, n;
  printf("Enter number of elements in array\n");
  scanf("%d", &n);
  printf("Enter %d integer(s)\n", n);
  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);
  printf("Enter a number to search\n");
  scanf("%d", &search);
  for (c = 0; c < n; c++)
  {
    if (array[c] == search)
    {
      printf("%d is present at location %d.\n", search, c+1);
      break;
    }
```

```
  }
  if (c == n)
    printf("%d isn't present in the array.\n", search);
  return 0;
}
```

OUTPUT



8.Write a program to find out Hamiltonian circuit Using backtracking method

PROGRAM

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_VERTICES 100
int graph[MAX_VERTICES][MAX_VERTICES];
int numVertices;
int hamiltonianPath[MAX_VERTICES];
bool isSafe(int v, int pos, int path[MAX_VERTICES]) {
    if (graph[path[pos - 1]][v] == 0) {
        return false;
    }
    for (int i = 0; i < pos; i++) {
        if (path[i] == v) {
            return false;
        }
    }
```

```
    return true;
}
bool hamiltonianUtil(int path[MAX_VERTICES], int pos) {
    if (pos == numVertices) {
        if (graph[path[pos - 1]][path[0]] == 1) {
            return true;
        }
        return false;
    }
    for (int v = 1; v < numVertices; v++) {
        if (isSafe(v, pos, path)) {
            path[pos] = v;
            if (hamiltonianUtil(path, pos + 1)) {
                return true;
            }
            path[pos] = -1;
        }
    }
    return false;
}
bool hamiltonianCircuit() {
    int path[MAX_VERTICES];
    for (int i = 0; i < numVertices; i++) {
        path[i] = -1;
    }
    path[0] = 0;
    if (hamiltonianUtil(path, 1)) {
        for (int i = 0; i < numVertices; i++) {
            hamiltonianPath[i] = path[i];
        }
        hamiltonianPath[numVertices] = path[0];
        return true;
    }
    return false;
}
int main() {
    printf("Enter the number of vertices in the graph: ");
    scanf("%d", &numVertices);
    printf("Enter the adjacency matrix of the graph (1 if there's an edge, 0 otherwise):\n");
    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
```

```
    if (hamiltonianCircuit()) {
        printf("Hamiltonian Circuit Found:\n");
        for (int i = 0; i <= numVertices; i++) {
            printf("%d ", hamiltonianPath[i]);
        }
        printf("\n");
    } else {
        printf("Hamiltonian Circuit Not Found.\n");
    }
    return 0;
}
```

OUTPUT