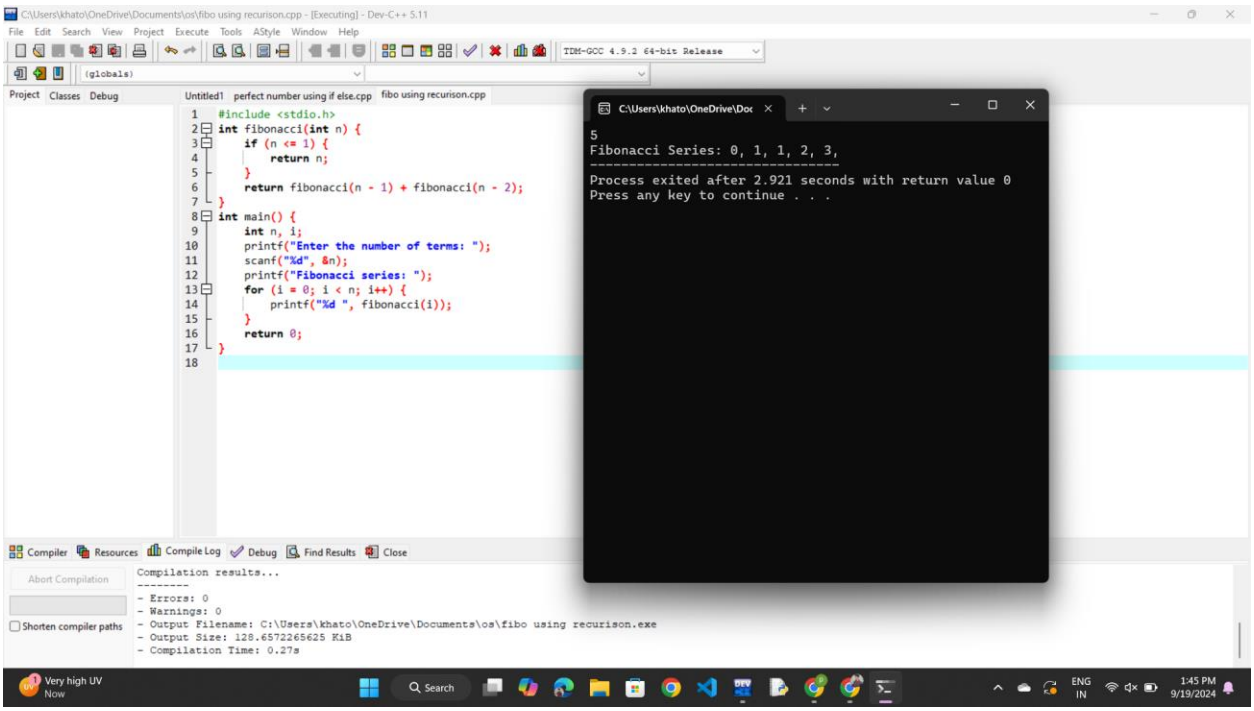
K PRASANTH (192211920) CSA0699 DAA

1.FIBONACICCI SERIES USING RECURSION

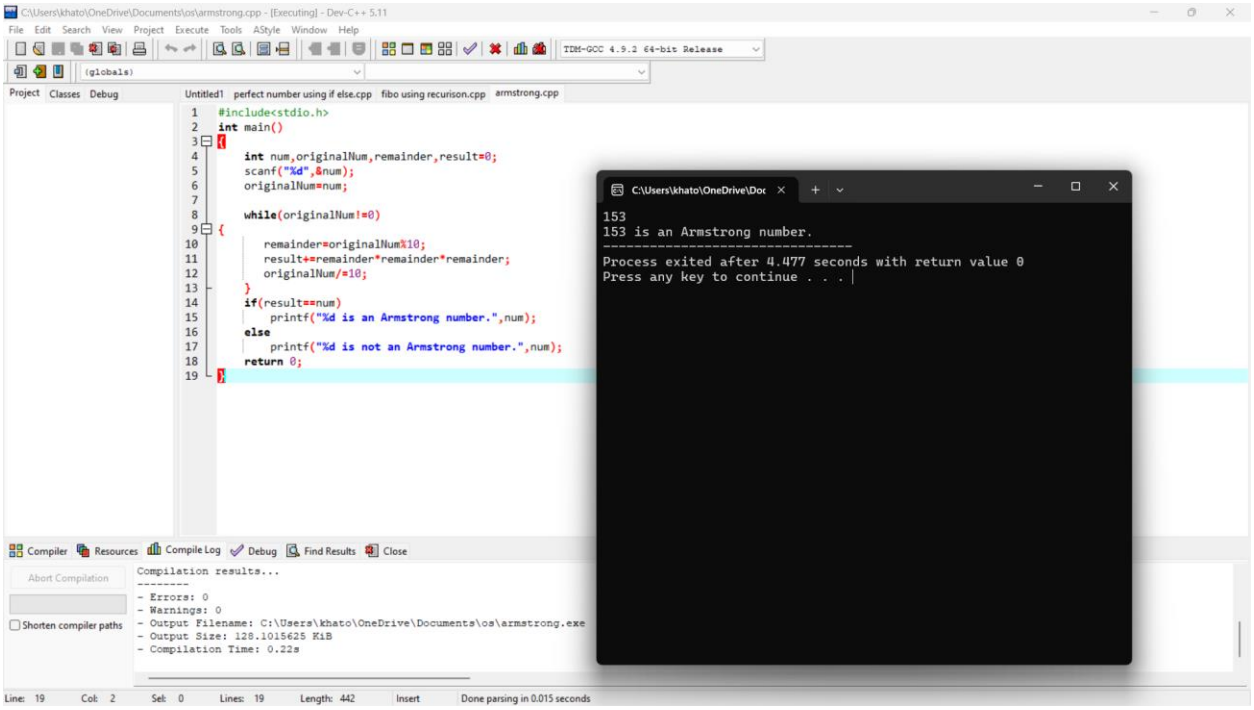
```
#include <stdio.h>
int fibonacci(int n) {
   if (n <= 1) {
      return n;
   }
   return fibonacci(n - 1) + fibonacci(n - 2);
}
int main() {
   int n, i;
   printf("Enter the number of terms: ");
   scanf("%d", &n);
   printf("Fibonacci series: ");
   for (i = 0; i < n; i++) {
      printf("%d ",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



2. ARMSTRONG NUMBER

```
#include<stdio.h>
int main()
{
   int num,originalNum,remainder,result=0;
   scanf("%d",&num);
   originalNum=num;

   while(originalNum!=0)
{
      remainder=originalNum%10;
      result+=remainder*remainder*remainder;
      originalNum/=10;
   }
```

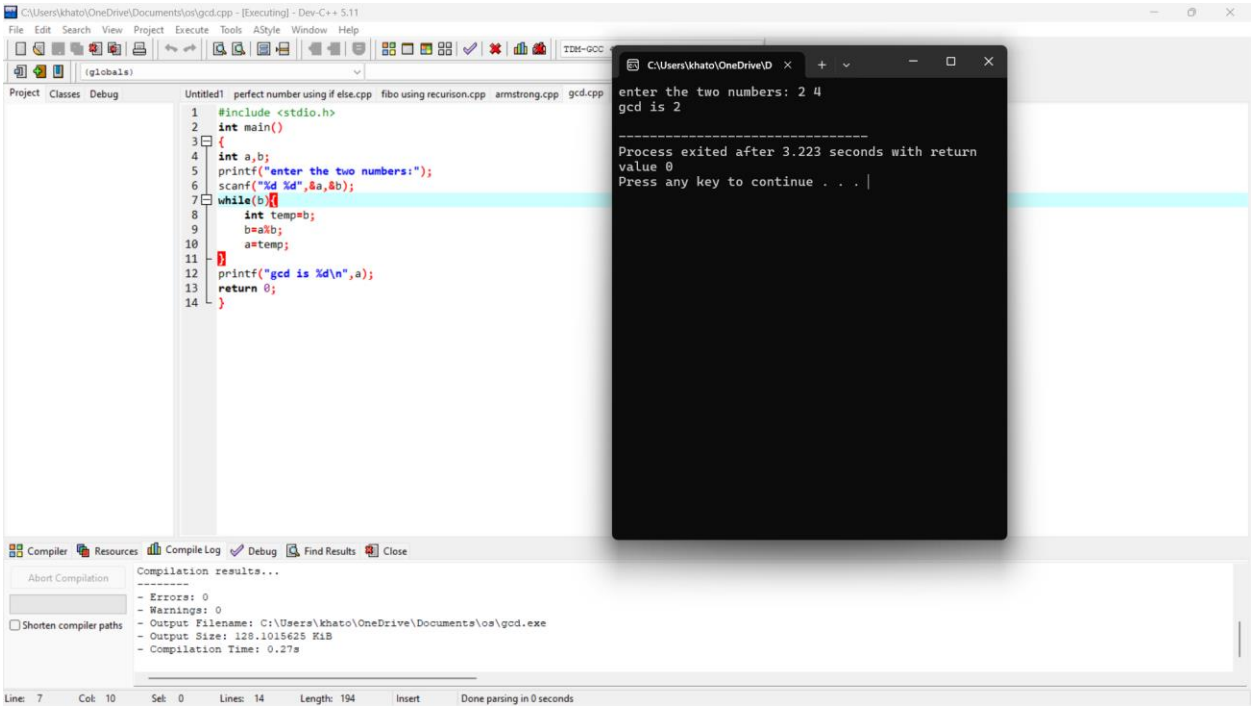
```
    if(result==num)
        printf("%d is an Armstrong number.",num);
    else
        printf("%d is not an Armstrong number.",num);
    return 0;
}
```



3. GCD OF TWO NUMBERS

```
#include <stdio.h>
int main()
{
int a,b;
printf("enter the two numbers:");
scanf("%d %d",&a,&b);
while(b){
```

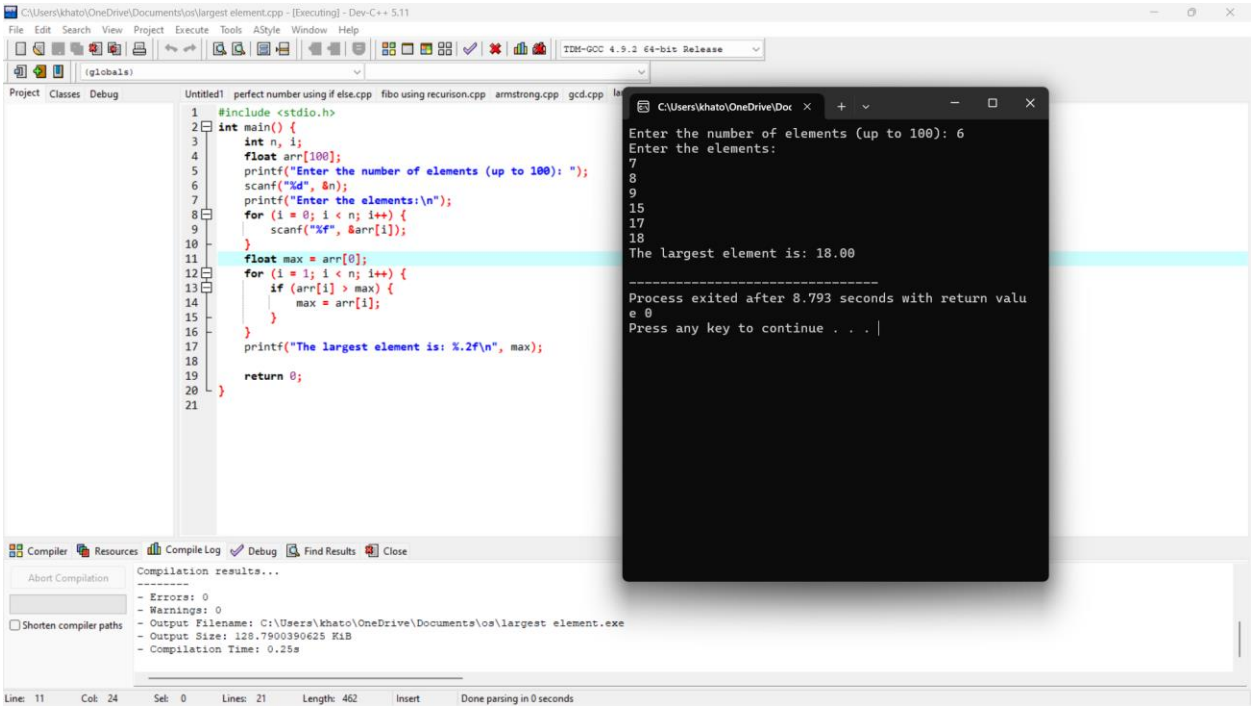
```
        int temp=b;
        b=a%b;
        a=temp;
}
printf("gcd is %d\n",a);
return 0;
}
```



4. LARGEST ELEMENT IN AN ARRAY

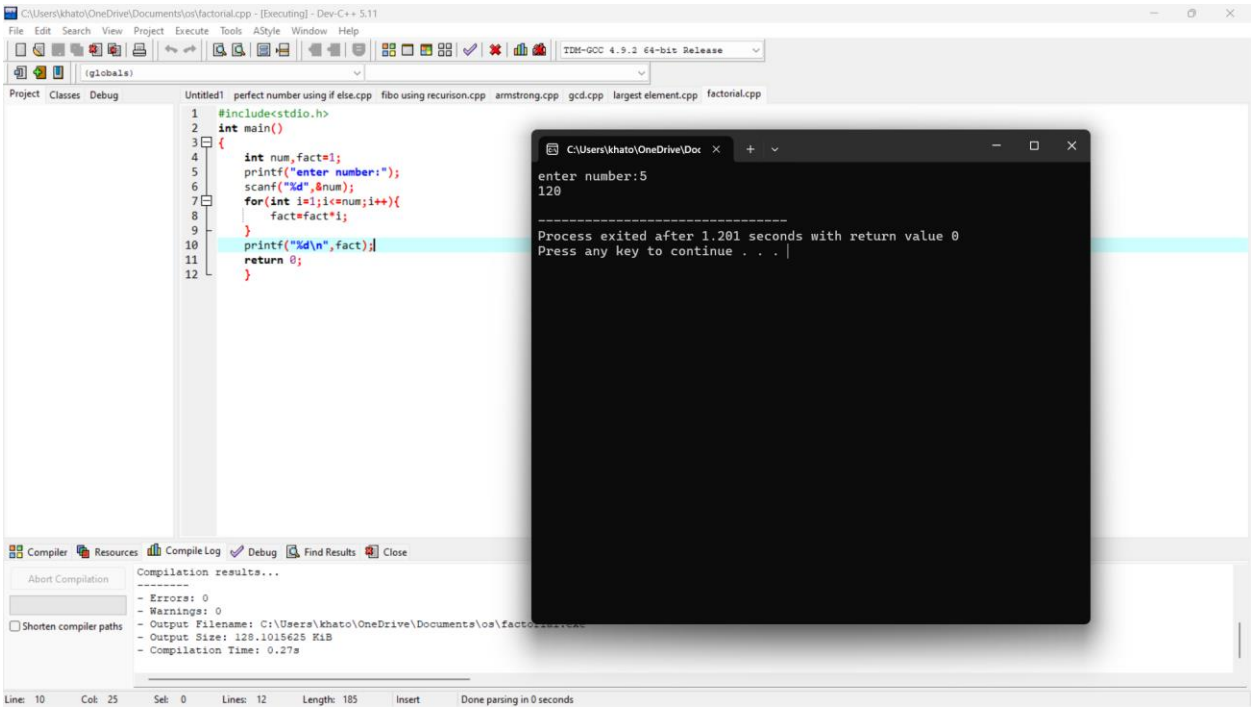
```
#include <stdio.h>
int main() {
    int n, i;
    float arr[100];
    printf("Enter the number of elements (up to 100): ");
    scanf("%d", &n);
```

```
    printf("Enter the elements:\n");
    for (i = 0; i < n; i++) {
        scanf("%f", &arr[i]);
    }
    float max = arr[0];
    for (i = 1; i < n; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    printf("The largest element is: %.2f\n", max);

    return 0;
}
```



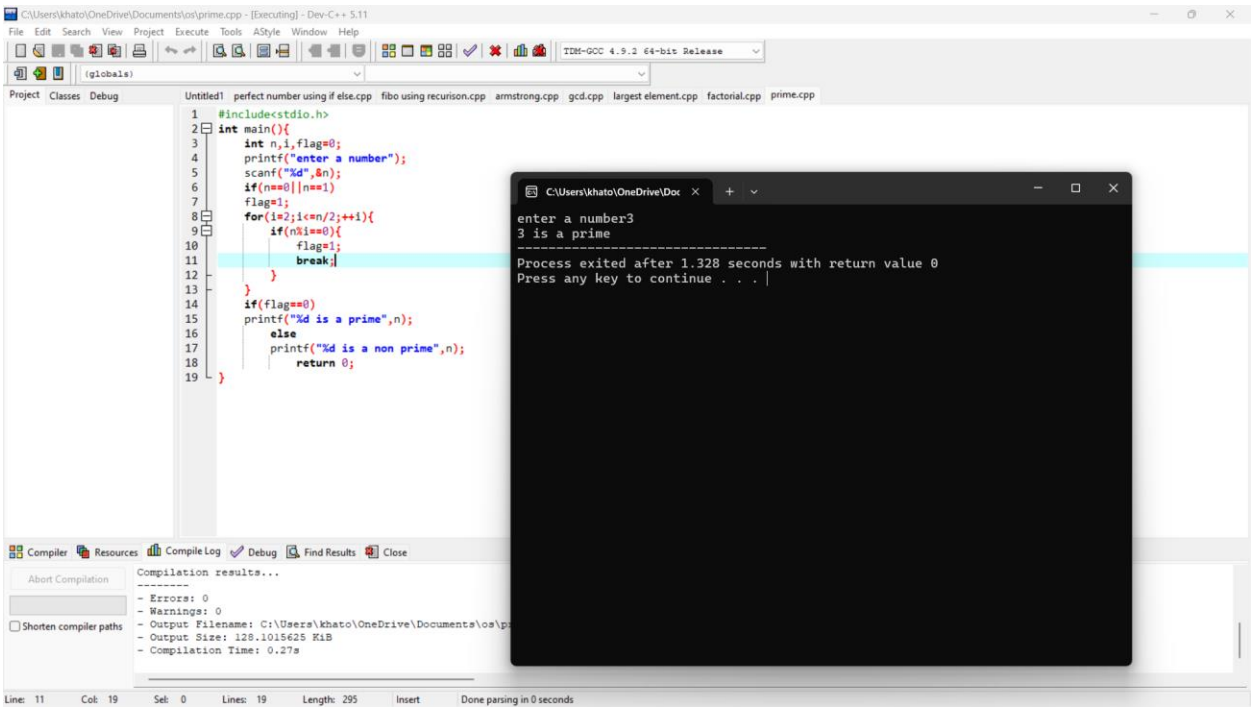
5. FACTORIAL

```
#include<stdio.h>
int main()
{
	int num,fact=1;
	printf("enter number:");
	scanf("%d",&num);
	for(int i=1;i<=num;i++){
		fact=fact*i;
	}
	printf("%d\n",fact);
	return 0;
	}
```



6. PRIME NUMBER

```
#include<stdio.h>
int main(){
	int n,i,flag=0;
	printf("enter a number");
	scanf("%d",&n);
	if(n==0||n==1)
	flag=1;
	for(i=2;i<=n/2;++i){
		if(n%i==0){
			flag=1;
			break;
		}
	}
	if(flag==0)
	printf("%d is a prime",n);
		else
		printf("%d is a non prime",n);
			return 0;
}
```

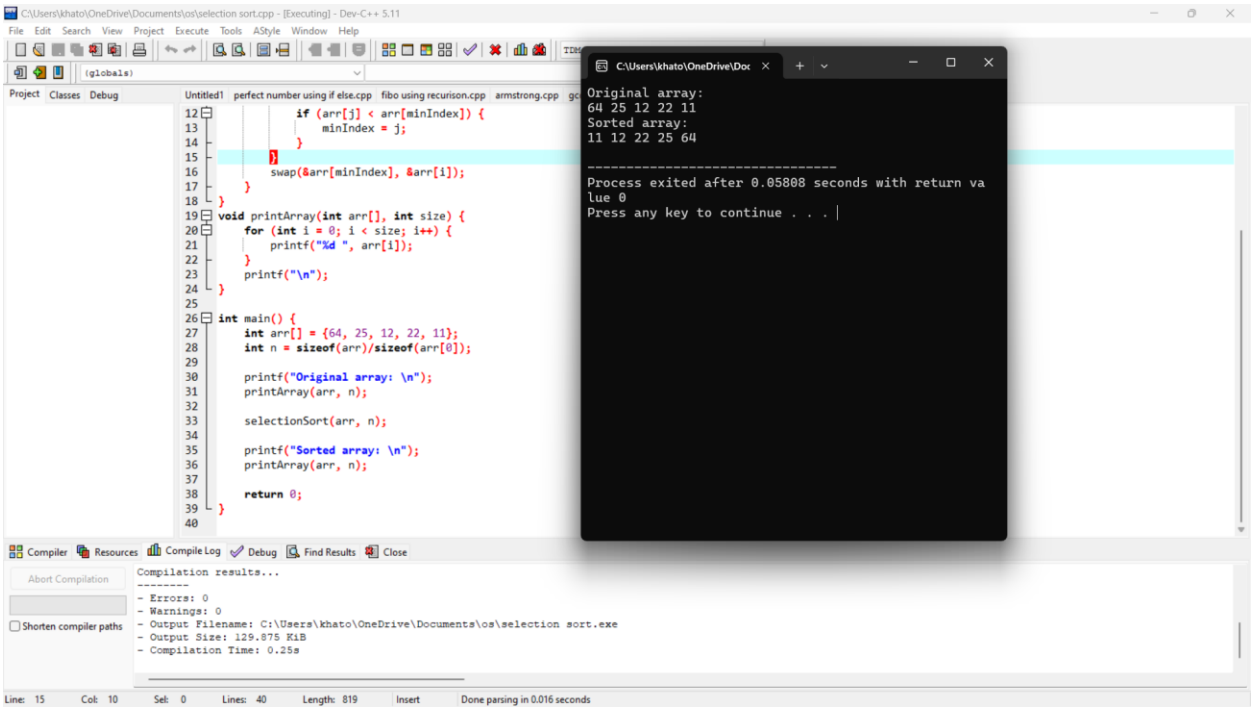


7. SELECTION SORT

```
#include <stdio.h>

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void selectionSort(int arr[], int n) {
    int i, j, minIndex;
    for (i = 0; i < n-1; i++) {
        minIndex = i;
        for (j = i+1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
```

```
            }
        }
        swap(&arr[minIndex], &arr[i]);
    }
}
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr)/sizeof(arr[0]);

    printf("Original array: \n");
    printArray(arr, n);

    selectionSort(arr, n);

    printf("Sorted array: \n");
    printArray(arr, n);
```

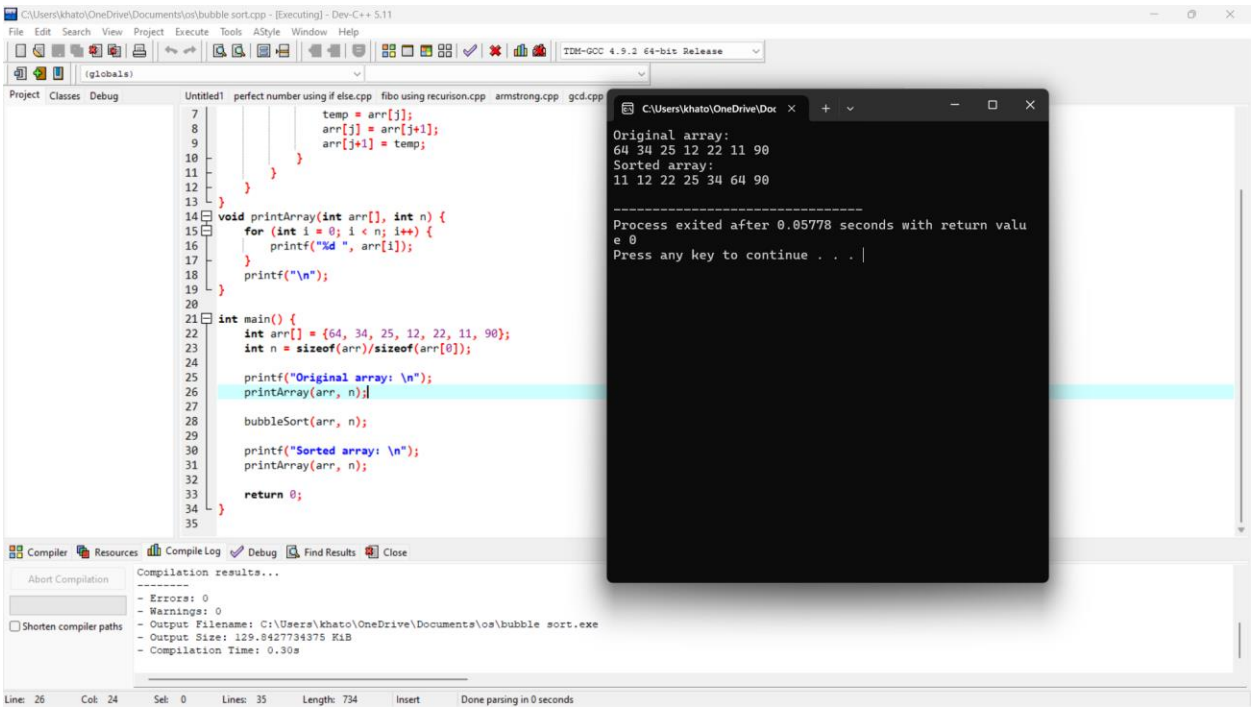
```
    return 0;
}
```



8. BUBBLE SORT

```
#include <stdio.h>

void bubbleSort(int arr[], int n) {
    int i, j, temp;
    for (i = 0; i < n-1; i++) {
        for (j = 0; j < n-i-1; j++) {
            if (arr[j] > arr[j+1]) {
                temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
```

```
    }
}
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);

    printf("Original array: \n");
    printArray(arr, n);

    bubbleSort(arr, n);

    printf("Sorted array: \n");
    printArray(arr, n);

    return 0;
}
```



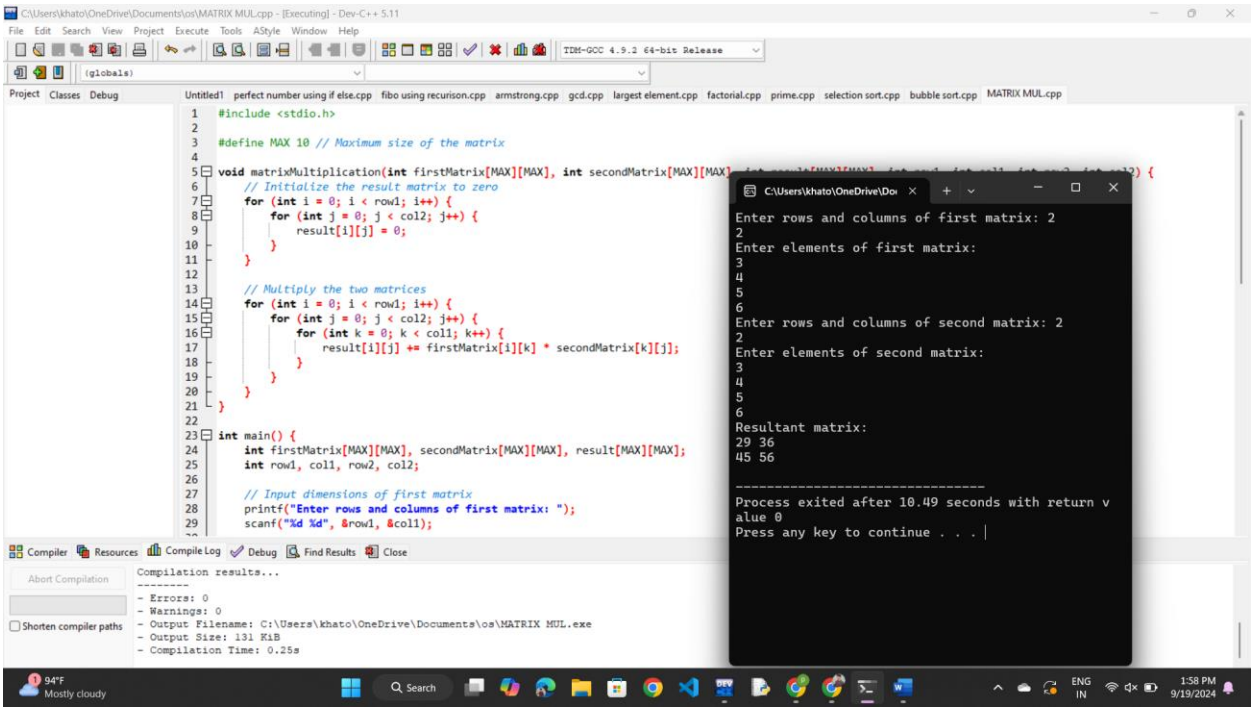
9. MATRIX MULTIPLICATION

```
#include <stdio.h>


#define MAX 10 // Maximum size of the matrix


void matrixMultiplication(int firstMatrix[MAX][MAX], int
secondMatrix[MAX][MAX], int result[MAX][MAX], int row1, int col1, int row2, int
col2) {
    // Initialize the result matrix to zero
    for (int i = 0; i < row1; i++) {
        for (int j = 0; j < col2; j++) {
            result[i][j] = 0;
        }
    }
```

```
    // Multiply the two matrices
    for (int i = 0; i < row1; i++) {
        for (int j = 0; j < col2; j++) {
            for (int k = 0; k < col1; k++) {
                result[i][j] += firstMatrix[i][k] * secondMatrix[k][j];
            }
        }
    }
}

int main() {
    int firstMatrix[MAX][MAX], secondMatrix[MAX][MAX], result[MAX][MAX];
    int row1, col1, row2, col2;

    // Input dimensions of first matrix
    printf("Enter rows and columns of first matrix: ");
    scanf("%d %d", &row1, &col1);

    // Input elements of first matrix
    printf("Enter elements of first matrix:\n");
    for (int i = 0; i < row1; i++) {
        for (int j = 0; j < col1; j++) {
            scanf("%d", &firstMatrix[i][j]);
        }
```

```
}

// Input dimensions of second matrix
printf("Enter rows and columns of second matrix: ");
scanf("%d %d", &row2, &col2);

// Matrix multiplication condition: col1 should be equal to row2
if (col1 != row2) {
    printf("Error! Matrix multiplication is not possible.\n");
    return -1;
}

// Input elements of second matrix
printf("Enter elements of second matrix:\n");
for (int i = 0; i < row2; i++) {
    for (int j = 0; j < col2; j++) {
        scanf("%d", &secondMatrix[i][j]);
    }
}

// Perform matrix multiplication
matrixMultiplication(firstMatrix, secondMatrix, result, row1, col1, row2, col2);

// Output the result matrix
```

```
    printf("Resultant matrix:\n");
    for (int i = 0; i < row1; i++) {
        for (int j = 0; j < col2; j++) {
            printf("%d ", result[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

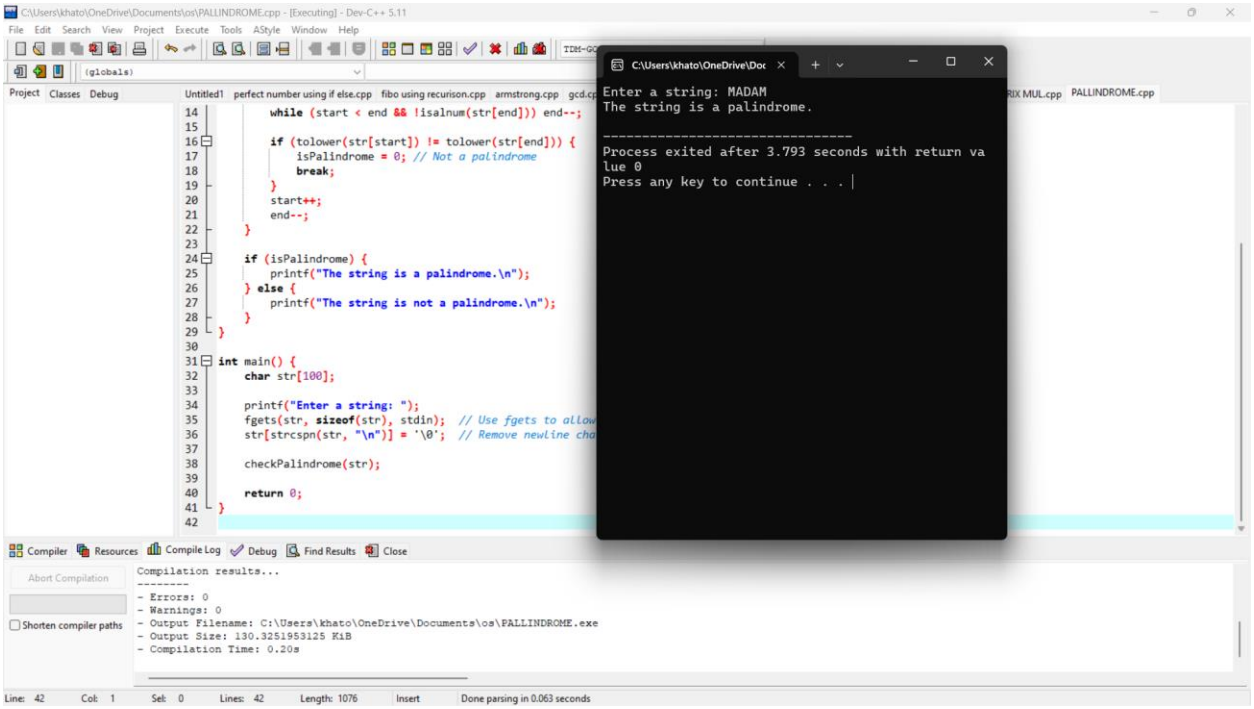


PALLINDROME

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

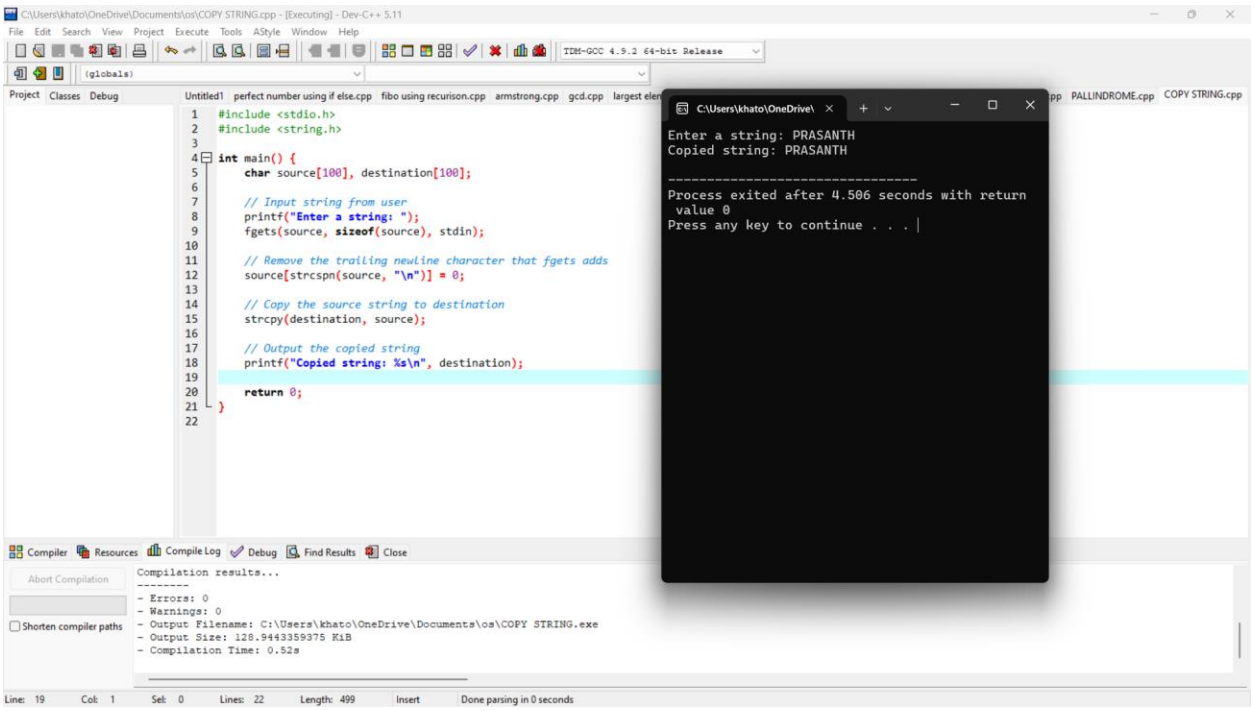
```
void checkPalindrome(char str[]) {
    int len = strlen(str);
    int start = 0;
    int end = len - 1;
    int isPalindrome = 1; // Assume the string is a palindrome

    while (start < end) {
        // Ignore non-alphanumeric characters and make it case-insensitive
        while (start < end && !isalnum(str[start])) start++;
        while (start < end && !isalnum(str[end])) end--;

        if (tolower(str[start]) != tolower(str[end])) {
            isPalindrome = 0; // Not a palindrome
            break;
        }
        start++;
        end--;
    }

    if (isPalindrome) {
        printf("The string is a palindrome.\n");
    } else {
        printf("The string is not a palindrome.\n");
```

```
    }
}

int main() {
    char str[100];

    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);  // Use fgets to allow spaces in the input
    str[strcspn(str, "\n")] = '\0';  // Remove newline character if present

    checkPalindrome(str);

    return 0;
}
```



COPY THE STRING

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

int main() {
    char source[100], destination[100];

    // Input string from user
    printf("Enter a string: ");
    fgets(source, sizeof(source), stdin);

    // Remove the trailing newline character that fgets adds
    source[strcspn(source, "\n")] = 0;

    // Copy the source string to destination
    strcpy(destination, source);

    // Output the copied string
    printf("Copied string: %s\n", destination);

    return 0;
}
```

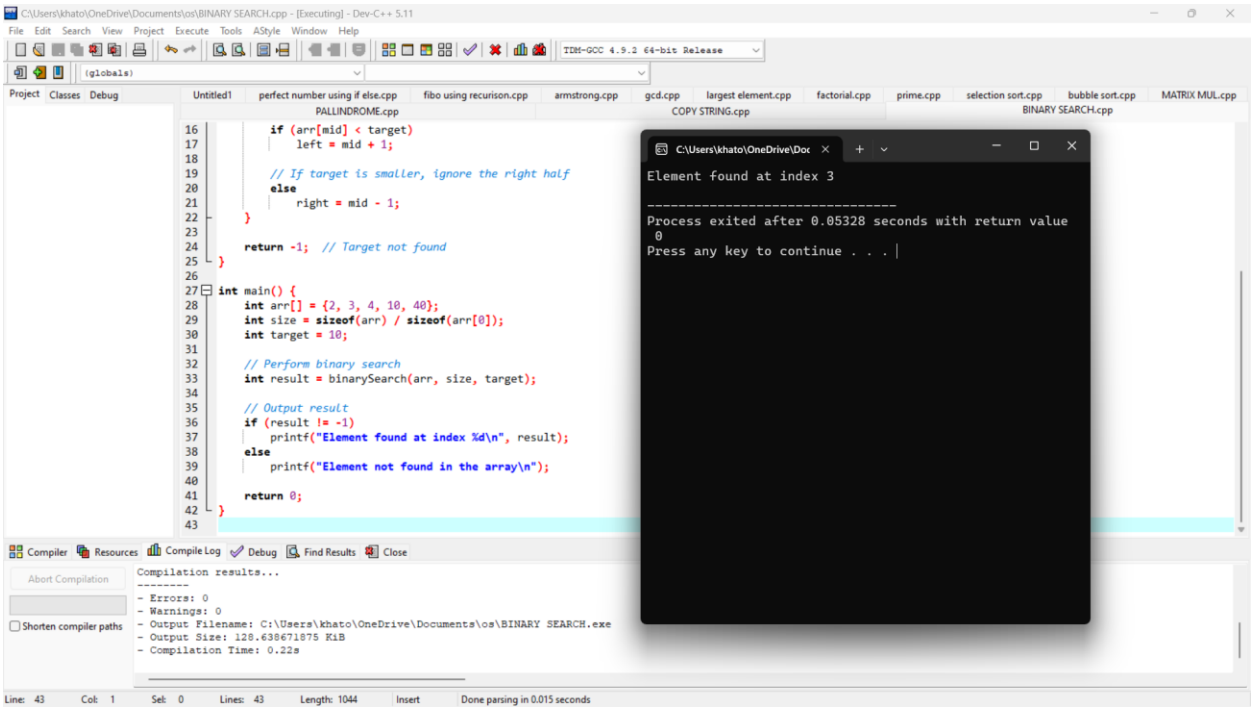


BINARY SEARCH

```
#include <stdio.h>


// Function to perform binary search
int binarySearch(int arr[], int size, int target) {
    int left = 0;
    int right = size - 1;


    while (left <= right) {
        int mid = left + (right - left) / 2; // Find the middle index


        // Check if target is present at mid
        if (arr[mid] == target)
            return mid;  // Target found
```

```
        // If target is greater, ignore the left half
        if (arr[mid] < target)
            left = mid + 1;

        // If target is smaller, ignore the right half
        else
            right = mid - 1;
    }

    return -1;  // Target not found
}

int main() {
    int arr[] = {2, 3, 4, 10, 40};
    int size = sizeof(arr) / sizeof(arr[0]);
    int target = 10;

    // Perform binary search
    int result = binarySearch(arr, size, target);

    // Output result
    if (result != -1)
        printf("Element found at index %d\n", result);
```

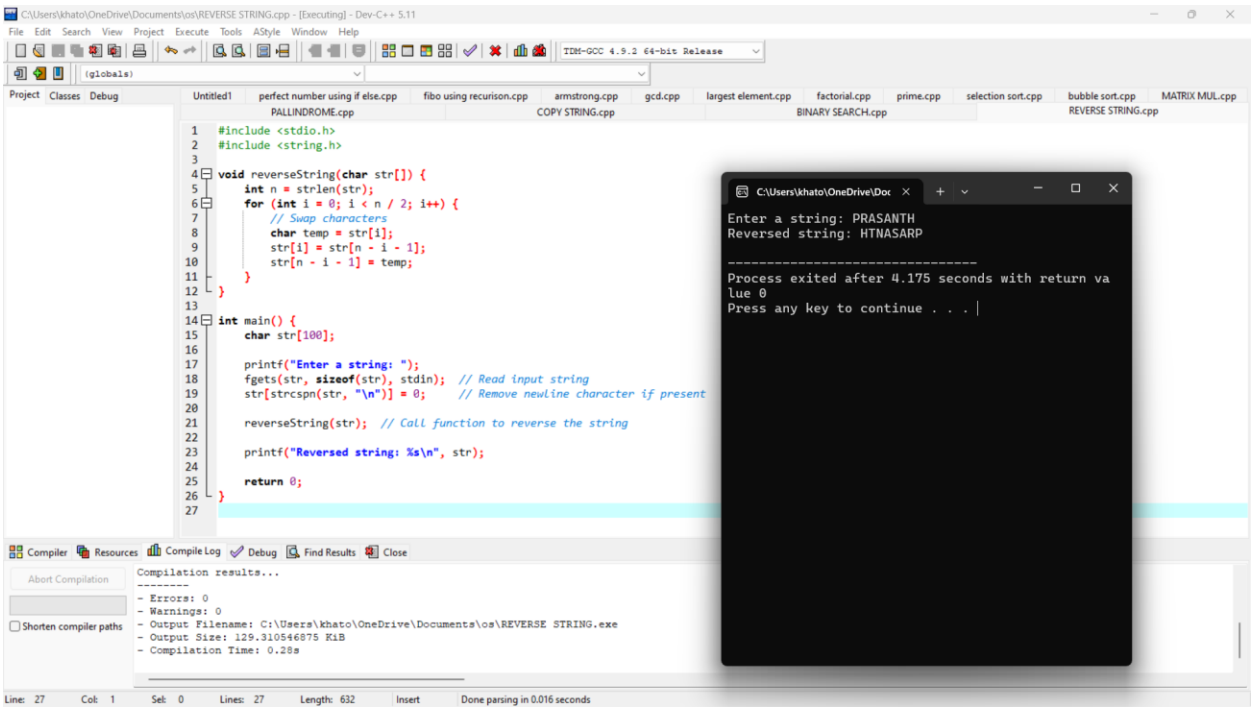
```
    else
        printf("Element not found in the array\n");

    return 0;
}
```



REVERSE STRING

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

void reverseString(char str[]) {
    int n = strlen(str);
    for (int i = 0; i < n / 2; i++) {
        // Swap characters
        char temp = str[i];
```

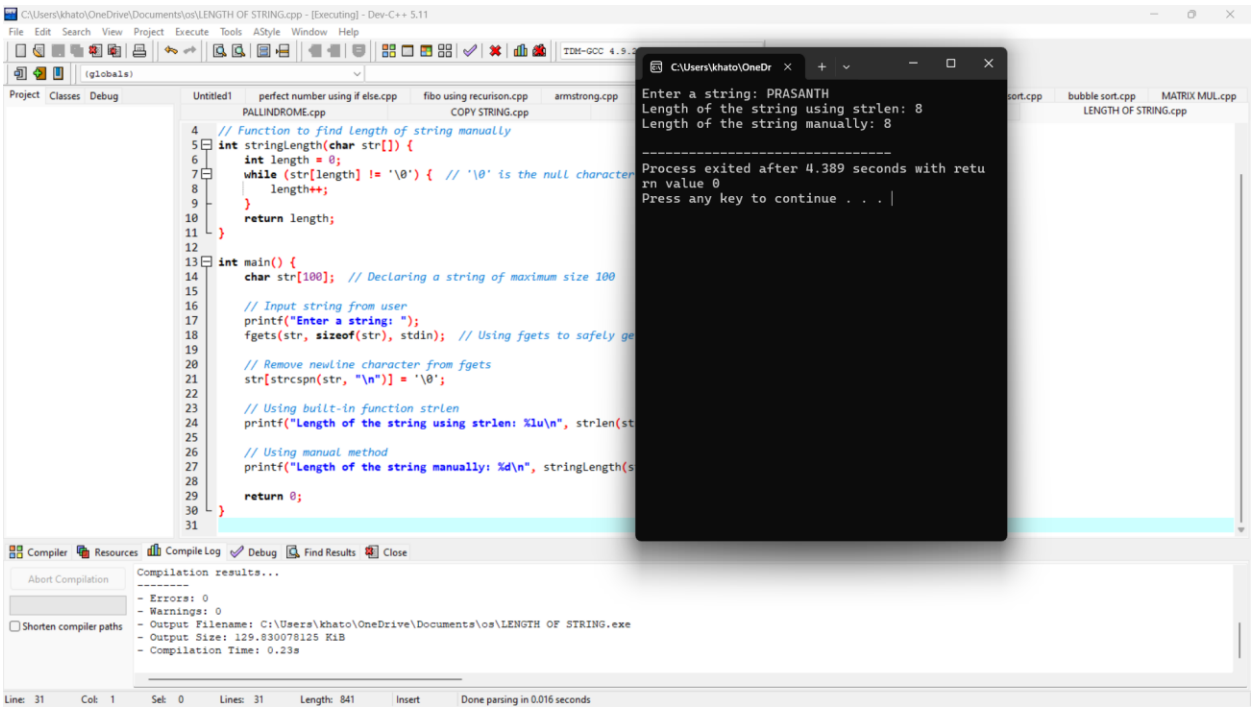
```
        str[i] = str[n - i - 1];
        str[n - i - 1] = temp;
    }
}

int main() {
    char str[100];

    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);  // Read input string
    str[strcspn(str, "\n")] = 0;     // Remove newline character if present

    reverseString(str);  // Call function to reverse the string

    printf("Reversed string: %s\n", str);

    return 0;
}
```



LENGTH OF STRING

```
#include <stdio.h>
#include <string.h>  // For strlen

// Function to find length of string manually
int stringLength(char str[]) {
    int length = 0;
    while (str[length] != '\0') {  // '\0' is the null character indicating the end of the string
        length++;
    }
    return length;
}
```

```
int main() {
    char str[100];  // Declaring a string of maximum size 100

    // Input string from user
    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);  // Using fgets to safely get input

    // Remove newline character from fgets
    str[strcspn(str, "\n")] = '\0';

    // Using built-in function strlen
    printf("Length of the string using strlen: %lu\n", strlen(str));

    // Using manual method
    printf("Length of the string manually: %d\n", stringLength(str));

    return 0;
}
```



STRASSEN MATRIX

```
#include <stdio.h>

#include <stdlib.h>


// Function to add two matrices

void addMatrix(int n, int A[][n], int B[][n], int result[][n]) {

    for (int i = 0; i < n; i++)

        for (int j = 0; j < n; j++)

            result[i][j] = A[i][j] + B[i][j];

}


// Function to subtract two matrices

void subtractMatrix(int n, int A[][n], int B[][n], int result[][n]) {

    for (int i = 0; i < n; i++)
```

```
        for (int j = 0; j < n; j++)
            result[i][j] = A[i][j] - B[i][j];
}

// Strassen's matrix multiplication function
void strassenMultiply(int n, int A[][n], int B[][n], int C[][n]) {
    // Base case: 2x2 matrix multiplication
    if (n == 2) {
        int P1 = (A[0][0] + A[1][1]) * (B[0][0] + B[1][1]);
        int P2 = (A[1][0] + A[1][1]) * B[0][0];
        int P3 = A[0][0] * (B[0][1] - B[1][1]);
        int P4 = A[1][1] * (B[1][0] - B[0][0]);
        int P5 = (A[0][0] + A[0][1]) * B[1][1];
        int P6 = (A[1][0] - A[0][0]) * (B[0][0] + B[0][1]);
        int P7 = (A[0][1] - A[1][1]) * (B[1][0] + B[1][1]);

        C[0][0] = P1 + P4 - P5 + P7;
        C[0][1] = P3 + P5;
        C[1][0] = P2 + P4;
        C[1][1] = P1 + P3 - P2 + P6;
        return;
    }

    int newSize = n / 2;
```

```
    int A11[newSize][newSize], A12[newSize][newSize], A21[newSize][newSize], A22[newSize][newSize];

    int B11[newSize][newSize], B12[newSize][newSize], B21[newSize][newSize], B22[newSize][newSize];

    int C11[newSize][newSize], C12[newSize][newSize], C21[newSize][newSize], C22[newSize][newSize];

    int P1[newSize][newSize], P2[newSize][newSize], P3[newSize][newSize], P4[newSize][newSize], P5[newSize][newSize], P6[newSize][newSize], P7[newSize][newSize];

    int temp1[newSize][newSize], temp2[newSize][newSize];


    // Divide A and B matrices into 4 sub-matrices
    for (int i = 0; i < newSize; i++) {
        for (int j = 0; j < newSize; j++) {
            A11[i][j] = A[i][j];
            A12[i][j] = A[i][j + newSize];
            A21[i][j] = A[i + newSize][j];
            A22[i][j] = A[i + newSize][j + newSize];

            B11[i][j] = B[i][j];
            B12[i][j] = B[i][j + newSize];
            B21[i][j] = B[i + newSize][j];
            B22[i][j] = B[i + newSize][j + newSize];
        }
    }
```

```
// P1 = (A11 + A22) * (B11 + B22)
addMatrix(newSize, A11, A22, temp1);
addMatrix(newSize, B11, B22, temp2);
strassenMultiply(newSize, temp1, temp2, P1);

// P2 = (A21 + A22) * B11
addMatrix(newSize, A21, A22, temp1);
strassenMultiply(newSize, temp1, B11, P2);

// P3 = A11 * (B12 - B22)
subtractMatrix(newSize, B12, B22, temp1);
strassenMultiply(newSize, A11, temp1, P3);

// P4 = A22 * (B21 - B11)
subtractMatrix(newSize, B21, B11, temp1);
strassenMultiply(newSize, A22, temp1, P4);

// P5 = (A11 + A12) * B22
addMatrix(newSize, A11, A12, temp1);
strassenMultiply(newSize, temp1, B22, P5);

// P6 = (A21 - A11) * (B11 + B12)
subtractMatrix(newSize, A21, A11, temp1);
addMatrix(newSize, B11, B12, temp2);
```

```
strassenMultiply(newSize, temp1, temp2, P6);

// P7 = (A12 - A22) * (B21 + B22)
subtractMatrix(newSize, A12, A22, temp1);
addMatrix(newSize, B21, B22, temp2);
strassenMultiply(newSize, temp1, temp2, P7);

// C11 = P1 + P4 - P5 + P7
addMatrix(newSize, P1, P4, temp1);
subtractMatrix(newSize, temp1, P5, temp2);
addMatrix(newSize, temp2, P7, C11);

// C12 = P3 + P5
addMatrix(newSize, P3, P5, C12);

// C21 = P2 + P4
addMatrix(newSize, P2, P4, C21);

// C22 = P1 + P3 - P2 + P6
addMatrix(newSize, P1, P3, temp1);
subtractMatrix(newSize, temp1, P2, temp2);
addMatrix(newSize, temp2, P6, C22);

// Combine C11, C12, C21, C22 into result matrix C
```

```
    for (int i = 0; i < newSize; i++) {
        for (int j = 0; j < newSize; j++) {
            C[i][j] = C11[i][j];
            C[i][j + newSize] = C12[i][j];
            C[i + newSize][j] = C21[i][j];
            C[i + newSize][j + newSize] = C22[i][j];
        }
    }
}

int main() {
    int n = 2; // Matrix size 2x2 (can be adjusted for larger sizes)

    int A[2][2] = { {1, 2}, {3, 4} };
    int B[2][2] = { {5, 6}, {7, 8} };
    int C[2][2]; // Result matrix

    strassenMultiply(n, A, B, C);

    // Print the result
    printf("Result of Strassen Matrix Multiplication:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", C[i][j]);
```

```
        }
        printf("\n");
    }

    return 0;
}
```

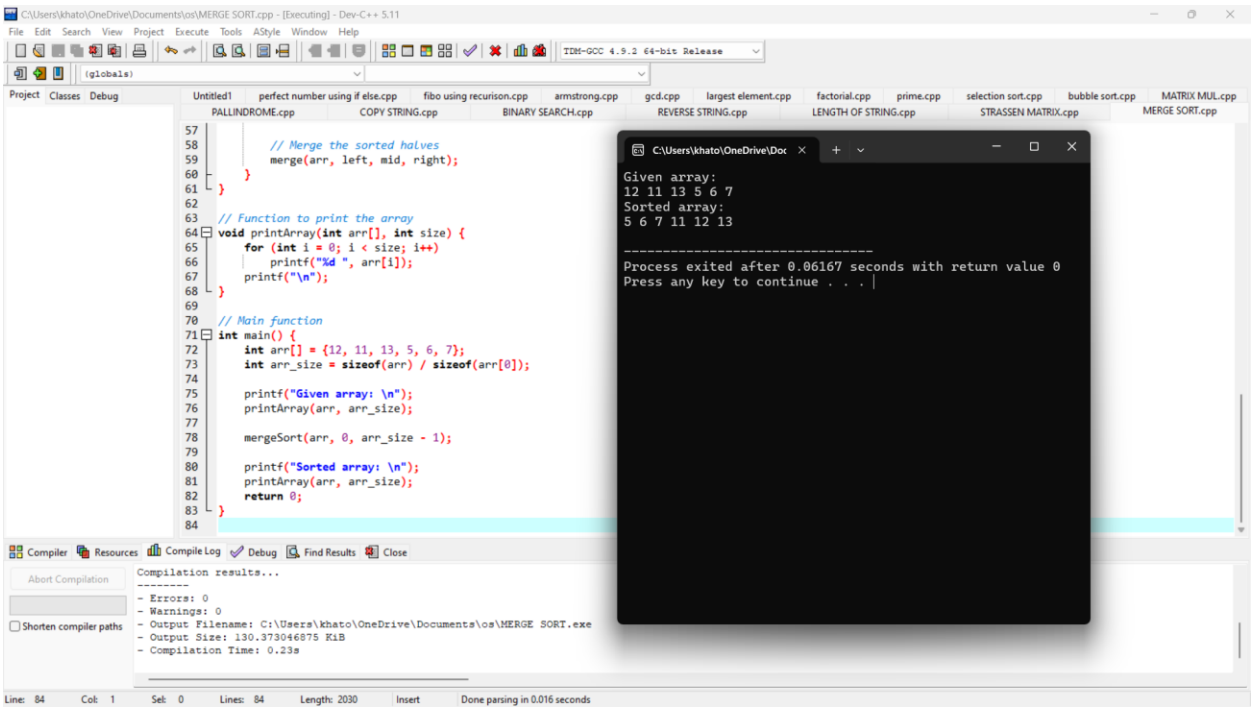
MERGE SORT

```
#include <stdio.h>

// Function to merge two halves of the array
void merge(int arr[], int left, int mid, int right) {
    int i, j, k;
    int n1 = mid - left + 1;
    int n2 = right - mid;

    // Temporary arrays to hold the left and right halves
    int leftArr[n1], rightArr[n2];

    // Copy data to temporary arrays
    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;  // Initial index of the left array
j = 0;  // Initial index of the right array
k = left;  // Initial index of merged array

// Merge the two halves
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++;
}

// Copy the remaining elements of rightArr[], if any
```

```
    while (j < n2) {
        arr[k] = rightArr[j];
        j++;
        k++;
    }
}

// Function to implement Merge Sort
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        // Sort the first and second halves
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        // Merge the sorted halves
        merge(arr, left, mid, right);
    }
}

// Function to print the array
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++)
```

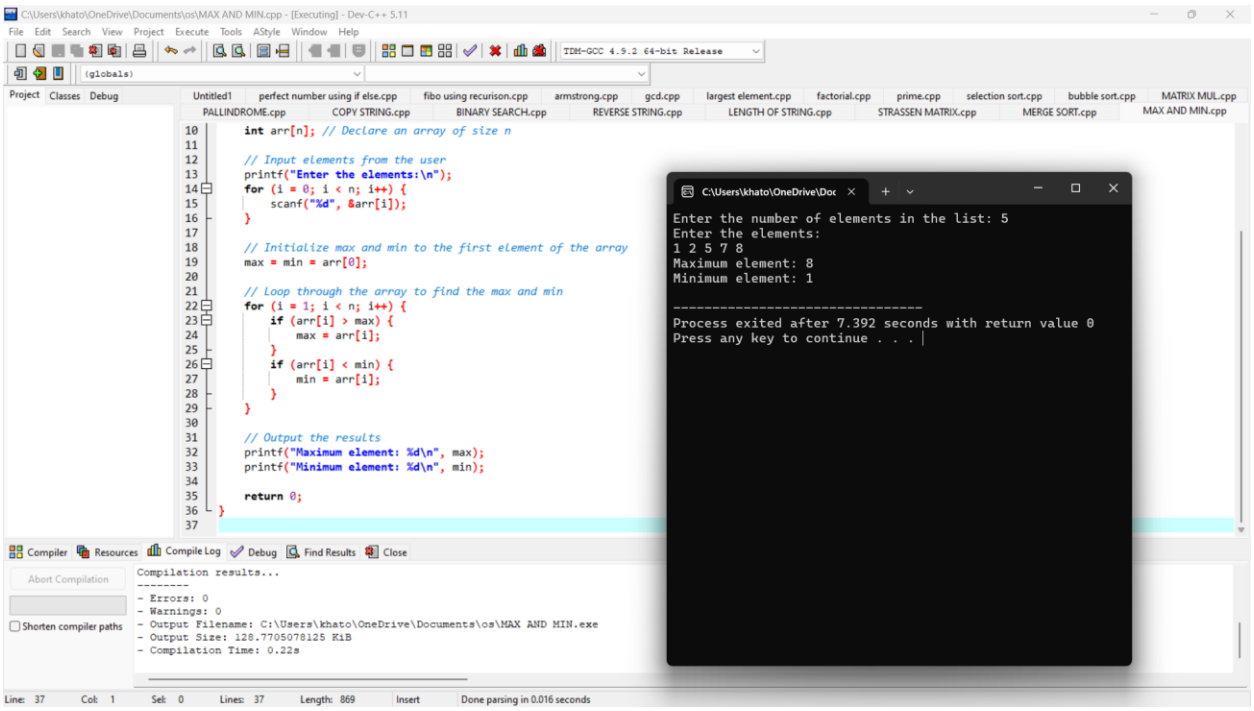
```
        printf("%d ", arr[i]);
    printf("\n");
}

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

    printf("Given array: \n");
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    printf("Sorted array: \n");
    printArray(arr, arr_size);
    return 0;
}
```



MAX AND MIN

```
#include <stdio.h>


int main() {
    int n, i, max, min;

    // Ask the user for the number of elements
    printf("Enter the number of elements in the list: ");
    scanf("%d", &n);

    int arr[n]; // Declare an array of size n

    // Input elements from the user
    printf("Enter the elements:\n");
```

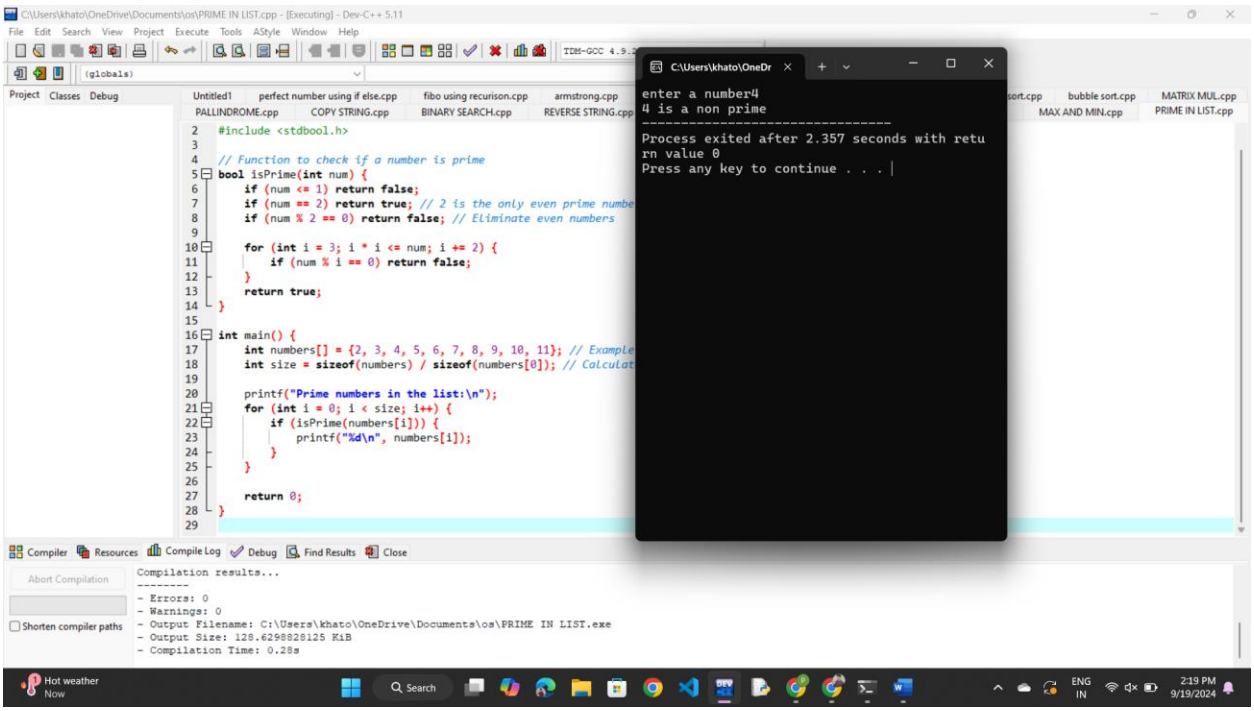
```
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Initialize max and min to the first element of the array
    max = min = arr[0];

    // Loop through the array to find the max and min
    for (i = 1; i < n; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
        if (arr[i] < min) {
            min = arr[i];
        }
    }

    // Output the results
    printf("Maximum element: %d\n", max);
    printf("Minimum element: %d\n", min);

    return 0;
}
```



PRIME NUMBERS IN LIMIT

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

// Function to check if a number is prime
bool isPrime(int num) {
    if (num <= 1) return false;
    if (num == 2) return true; // 2 is the only even prime number
    if (num % 2 == 0) return false; // Eliminate even numbers

    for (int i = 3; i * i <= num; i += 2) {
        if (num % i == 0) return false;
    }
```

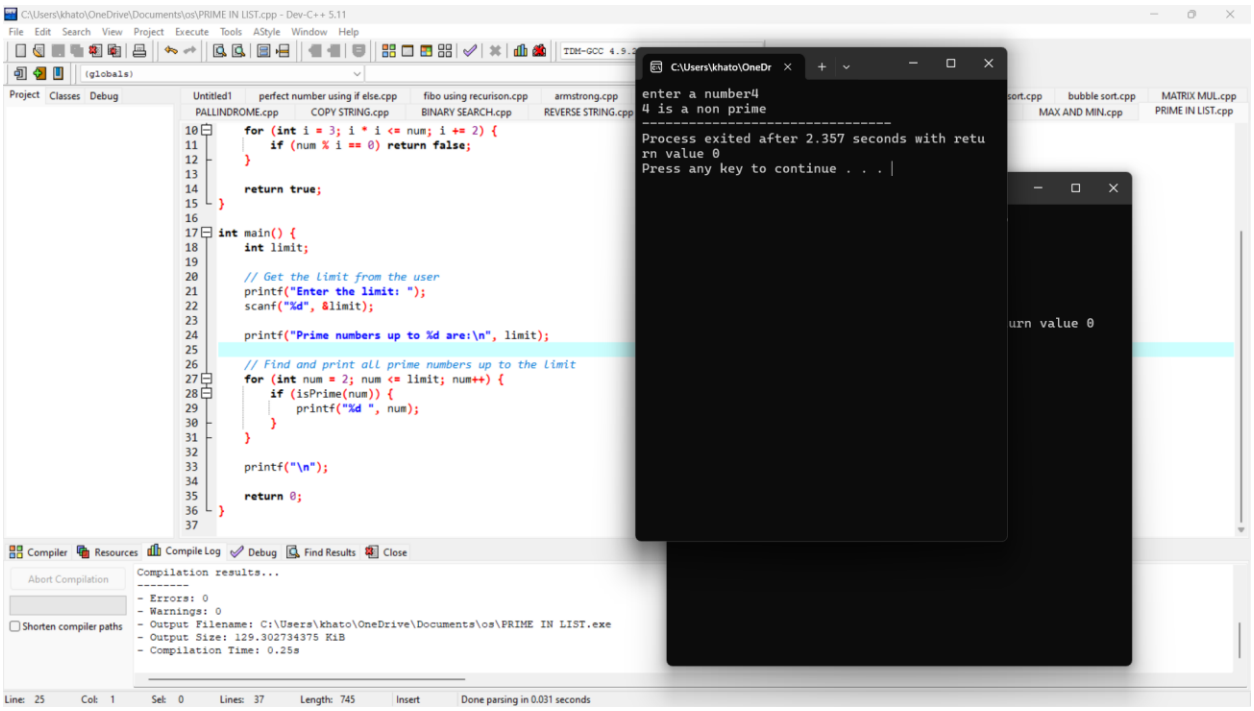
```
    return true;
}

int main() {
    int numbers[] = {2, 3, 4, 5, 6, 7, 8, 9, 10, 11}; // Example list of numbers
    int size = sizeof(numbers) / sizeof(numbers[0]); // Calculate size of the array

    printf("Prime numbers in the list:\n");
    for (int i = 0; i < size; i++) {
        if (isPrime(numbers[i])) {
            printf("%d\n", numbers[i]);
        }
    }

    return 0;
}
```



PRIME NUMBERS UPTO

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

// Function to check if a number is prime
bool isPrime(int num) {
    if (num <= 1) return false;
    if (num == 2) return true;
    if (num % 2 == 0) return false;

    for (int i = 3; i * i <= num; i += 2) {
        if (num % i == 0) return false;
    }
```

```
    return true;
}

int main() {
    int limit;

    // Get the limit from the user
    printf("Enter the limit: ");
    scanf("%d", &limit);

    printf("Prime numbers up to %d are:\n", limit);

    // Find and print all prime numbers up to the limit
    for (int num = 2; num <= limit; num++) {
        if (isPrime(num)) {
            printf("%d ", num);
        }
    }

    printf("\n");

    return 0;
}
```



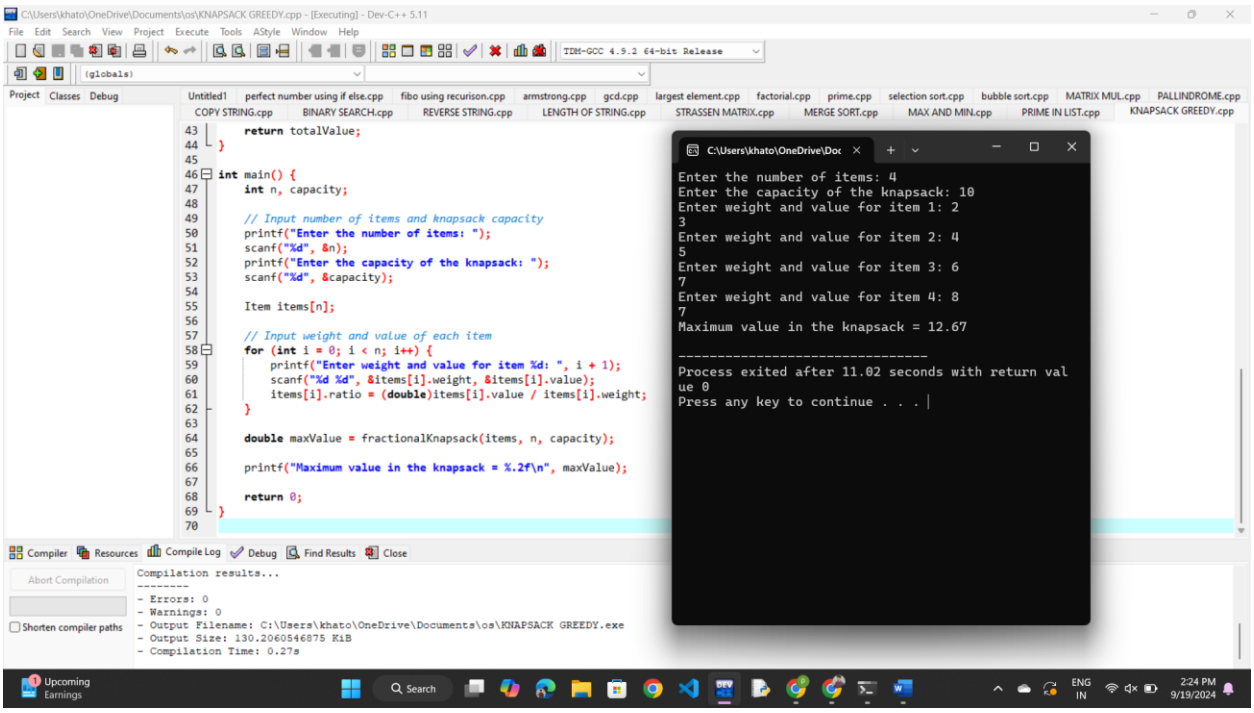
KNAPSACK USING GREEDY TECHNIQUE

```
#include <stdio.h>


// Structure to represent an item
typedef struct {
    int weight;
    int value;
    double ratio;
} Item;


// Function to sort items based on their value-to-weight ratio in descending order
void sortItems(Item items[], int n) {
    Item temp;
    for (int i = 0; i < n-1; i++) {
```

```
        for (int j = 0; j < n-i-1; j++) {
            if (items[j].ratio < items[j+1].ratio) {
                // Swap items[j] and items[j+1]
                temp = items[j];
                items[j] = items[j+1];
                items[j+1] = temp;
            }
        }
    }
}

// Function to solve the fractional knapsack problem
double fractionalKnapsack(Item items[], int n, int capacity) {
    sortItems(items, n);

    double totalValue = 0.0;
    int remainingCapacity = capacity;

    for (int i = 0; i < n; i++) {
        if (items[i].weight <= remainingCapacity) {
            // Take the whole item
            remainingCapacity -= items[i].weight;
            totalValue += items[i].value;
        } else {
```

```
            // Take the fraction of the item
            totalValue += items[i].value * ((double)remainingCapacity / items[i].weight);
            break;
        }
    }
    return totalValue;
}

int main() {
    int n, capacity;

    // Input number of items and knapsack capacity
    printf("Enter the number of items: ");
    scanf("%d", &n);
    printf("Enter the capacity of the knapsack: ");
    scanf("%d", &capacity);

    Item items[n];

    // Input weight and value of each item
    for (int i = 0; i < n; i++) {
        printf("Enter weight and value for item %d: ", i + 1);
        scanf("%d %d", &items[i].weight, &items[i].value);
        items[i].ratio = (double)items[i].value / items[i].weight;
```

```
    }

    double maxValue = fractionalKnapsack(items, n, capacity);

    printf("Maximum value in the knapsack = %.2f\n", maxValue);

    return 0;
}
```



MST USING GREEDY

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

// Define the maximum number of vertices
#define MAX_VERTICES 100
```

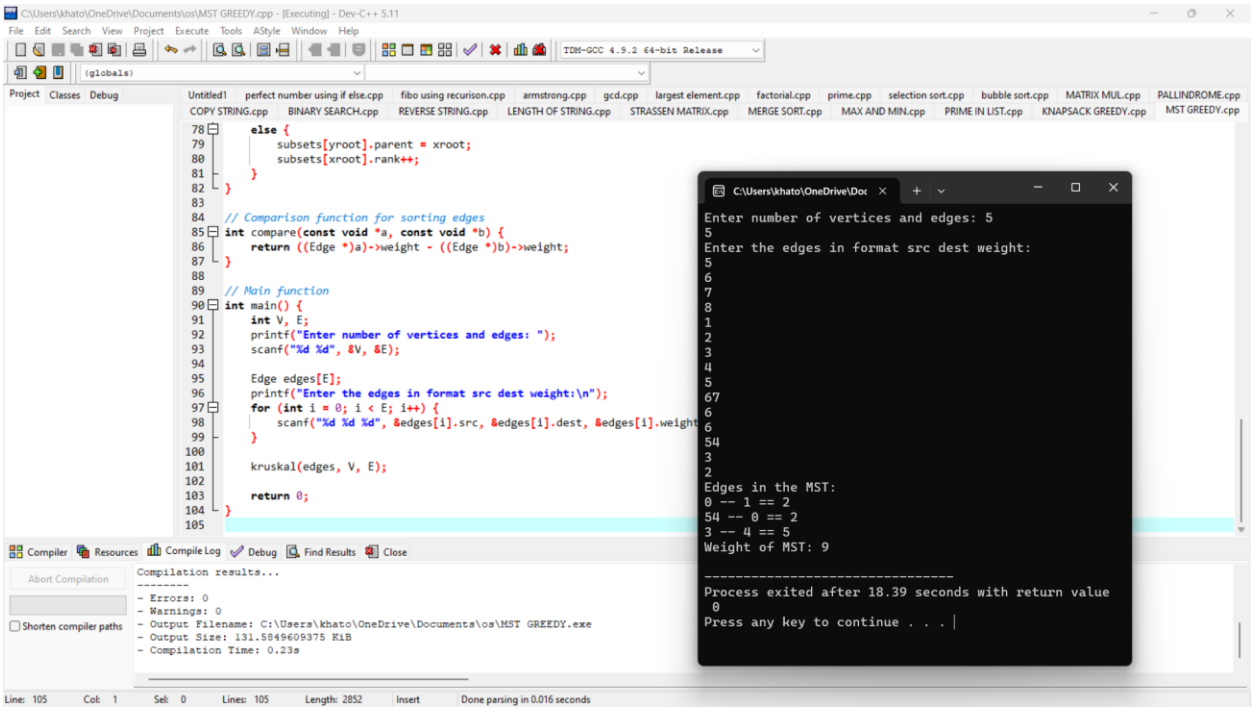
```
#define INF 999999

// Structure to represent an edge
typedef struct {
    int src, dest, weight;
} Edge;

// Structure to represent a subset for Union-Find
typedef struct {
    int parent, rank;
} Subset;

// Function prototypes
int find(Subset subsets[], int i);
void unionSets(Subset subsets[], int x, int y);
int compare(const void *a, const void *b);

// Kruskal's algorithm to find MST
void kruskal(Edge edges[], int V, int E) {
    Edge result[V]; // To store the resulting MST
    Subset subsets[V]; // To represent the Union-Find structure
    int e = 0; // Count of edges in MST
    int i = 0; // Initial index of sorted edges
```

```
// Initialize subsets
for (int v = 0; v < V; ++v) {
    subsets[v].parent = v;
    subsets[v].rank = 0;
}

// Sort all edges by weight
qsort(edges, E, sizeof(edges[0]), compare);

// Process each edge in sorted order
while (e < V - 1 && i < E) {
    Edge next_edge = edges[i++];
    int x = find(subsets, next_edge.src);
    int y = find(subsets, next_edge.dest);

    // If including this edge does not cause a cycle
    if (x != y) {
        result[e++] = next_edge;
        unionSets(subsets, x, y);
    }
}

// Print the MST
printf("Edges in the MST:\n");
```

```
    int mst_weight = 0;
    for (i = 0; i < e; ++i) {
        printf("%d -- %d == %d\n", result[i].src, result[i].dest, result[i].weight);
        mst_weight += result[i].weight;
    }
    printf("Weight of MST: %d\n", mst_weight);
}

// Find function for Union-Find
int find(Subset subsets[], int i) {
    if (subsets[i].parent != i)
        subsets[i].parent = find(subsets, subsets[i].parent);
    return subsets[i].parent;
}

// Union function for Union-Find
void unionSets(Subset subsets[], int x, int y) {
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);

    if (subsets[xroot].rank < subsets[yroot].rank)
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
        subsets[yroot].parent = xroot;
```

```
    else {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}

// Comparison function for sorting edges
int compare(const void *a, const void *b) {
    return ((Edge *)a)->weight - ((Edge *)b)->weight;
}

// Main function
int main() {
    int V, E;
    printf("Enter number of vertices and edges: ");
    scanf("%d %d", &V, &E);

    Edge edges[E];
    printf("Enter the edges in format src dest weight:\n");
    for (int i = 0; i < E; i++) {
        scanf("%d %d %d", &edges[i].src, &edges[i].dest, &edges[i].weight);
    }

    kruskal(edges, V, E);
```

```
    return 0;
}
```



OPTIMAL BINARY SEARCH TREE

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

// Function prototypes
void printOptimalBST(int i, int j, int s[][j+1], int keys[], int n);
void optimalBST(int keys[], int freq[], int n);

int main() {
    int keys[] = {10, 12, 20}; // Keys in the BST
```

```
    int freq[] = {34, 8, 50};  // Frequencies of the keys
    int n = sizeof(keys)/sizeof(keys[0]);

    optimalBST(keys, freq, n);

    return 0;
}

// Function to print the optimal BST
void printOptimalBST(int i, int j, int s[][j+1], int keys[], int n) {
    if (i <= j) {
        printf("Key %d (root) with keys in range [%d-%d]\n", keys[s[i][j]], i, j);
        printOptimalBST(i, s[i][j] - 1, s, keys, n);
        printOptimalBST(s[i][j] + 1, j, s, keys, n);
    }
}

// Function to construct and print the optimal BST
void optimalBST(int keys[], int freq[], int n) {
    int i, j, len, r;
    int cost[n][n], s[n][n];

    // Initialize cost and s tables
    for (i = 0; i < n; i++) {
```

```
    cost[i][i] = freq[i];
    s[i][i] = i;
    for (j = i + 1; j < n; j++) {
        cost[i][j] = INT_MAX;
    }
}

// Build the cost and s tables
for (len = 2; len <= n; len++) {
    for (i = 0; i <= n - len; i++) {
        j = i + len - 1;
        int sum = 0;
        for (r = i; r <= j; r++) {
            sum += freq[r];
        }

        for (r = i; r <= j; r++) {
            int c = (r > i ? cost[i][r - 1] : 0) + (r < j ? cost[r + 1][j] : 0) + sum;
            if (c < cost[i][j]) {
                cost[i][j] = c;
                s[i][j] = r;
            }
        }
    }
```

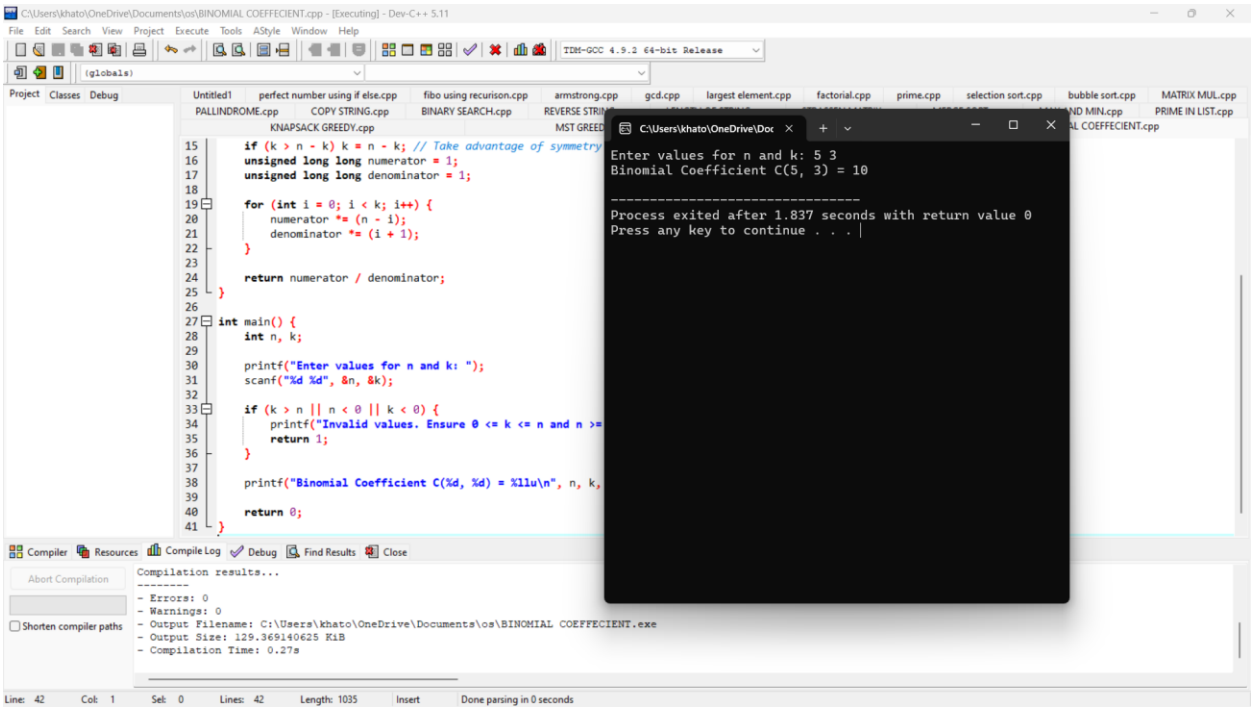
```
    }

    // Print the cost of the optimal BST
    printf("Cost of Optimal BST: %d\n", cost[0][n - 1]);

    // Print the optimal BST structure
    printf("Optimal BST structure:\n");
    printOptimalBST(0, n - 1, s, keys, n);
}
```

BINOMIAL COEFFECIENT

```
#include <stdio.h>

// Function to calculate factorial of a number
unsigned long long factorial(int n) {
    if (n == 0 || n == 1) return 1;
    unsigned long long fact = 1;
    for (int i = 2; i <= n; i++) {
        fact *= i;
    }
    return fact;
}

// Function to calculate binomial coefficient
unsigned long long binomialCoefficient(int n, int k) {
```

```
    if (k > n - k) k = n - k; // Take advantage of symmetry
    unsigned long long numerator = 1;
    unsigned long long denominator = 1;

    for (int i = 0; i < k; i++) {
        numerator *= (n - i);
        denominator *= (i + 1);
    }

    return numerator / denominator;
}

int main() {
    int n, k;

    printf("Enter values for n and k: ");
    scanf("%d %d", &n, &k);

    if (k > n || n < 0 || k < 0) {
        printf("Invalid values. Ensure 0 <= k <= n and n >= 0.\n");
        return 1;
    }

    printf("Binomial Coefficient C(%d, %d) = %llu\n", n, k, binomialCoefficient(n, k));
```

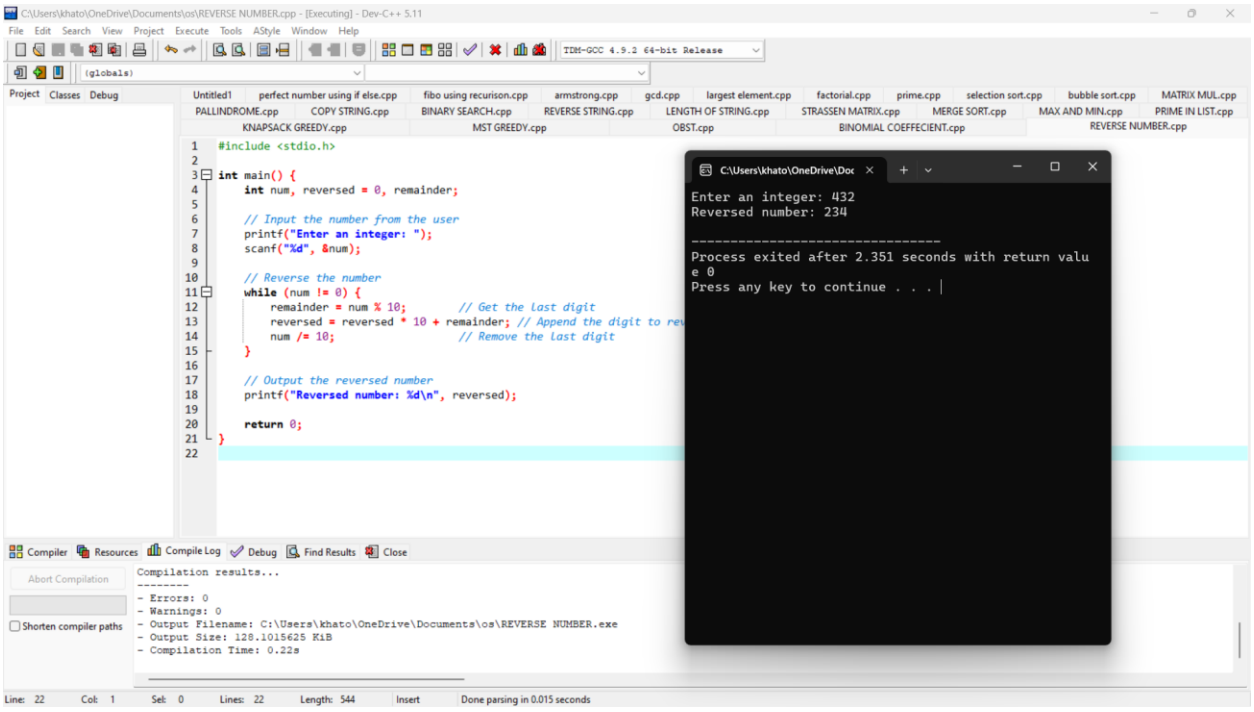
```
    return 0;
}
```



REVERSE A NUMBER

```
#include <stdio.h>


int main() {
    int num, reversed = 0, remainder;


    // Input the number from the user
    printf("Enter an integer: ");
    scanf("%d", &num);


    // Reverse the number
```

```
    while (num != 0) {
        remainder = num % 10;       // Get the last digit
        reversed = reversed * 10 + remainder; // Append the digit to reversed
        num /= 10;                  // Remove the last digit
    }

    // Output the reversed number
    printf("Reversed number: %d\n", reversed);

    return 0;
}
```

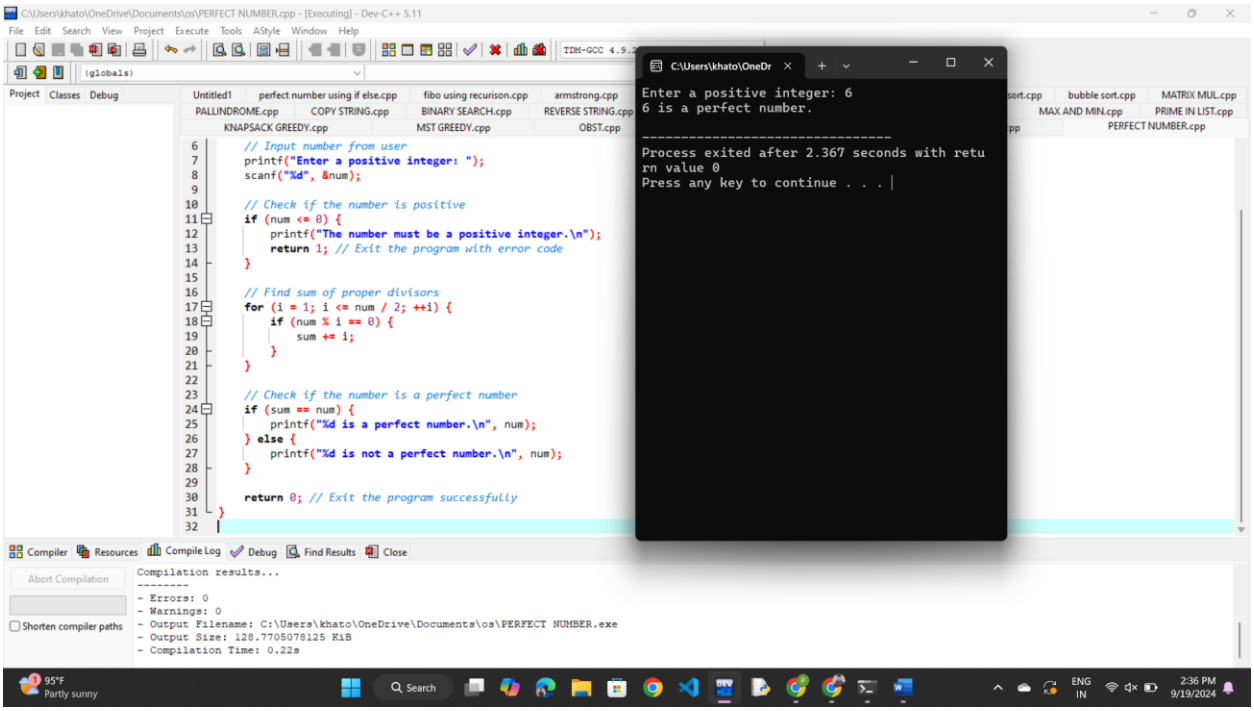


PERFECT NUMBER

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

```
int main() {
    int num, i, sum = 0;

    // Input number from user
    printf("Enter a positive integer: ");
    scanf("%d", &num);

    // Check if the number is positive
    if (num <= 0) {
        printf("The number must be a positive integer.\n");
        return 1; // Exit the program with error code
    }

    // Find sum of proper divisors
    for (i = 1; i <= num / 2; ++i) {
        if (num % i == 0) {
            sum += i;
        }
    }

    // Check if the number is a perfect number
    if (sum == num) {
        printf("%d is a perfect number.\n", num);
    } else {
```

```
        printf("%d is not a perfect number.\n", num);
    }

    return 0; // Exit the program successfully
}
```



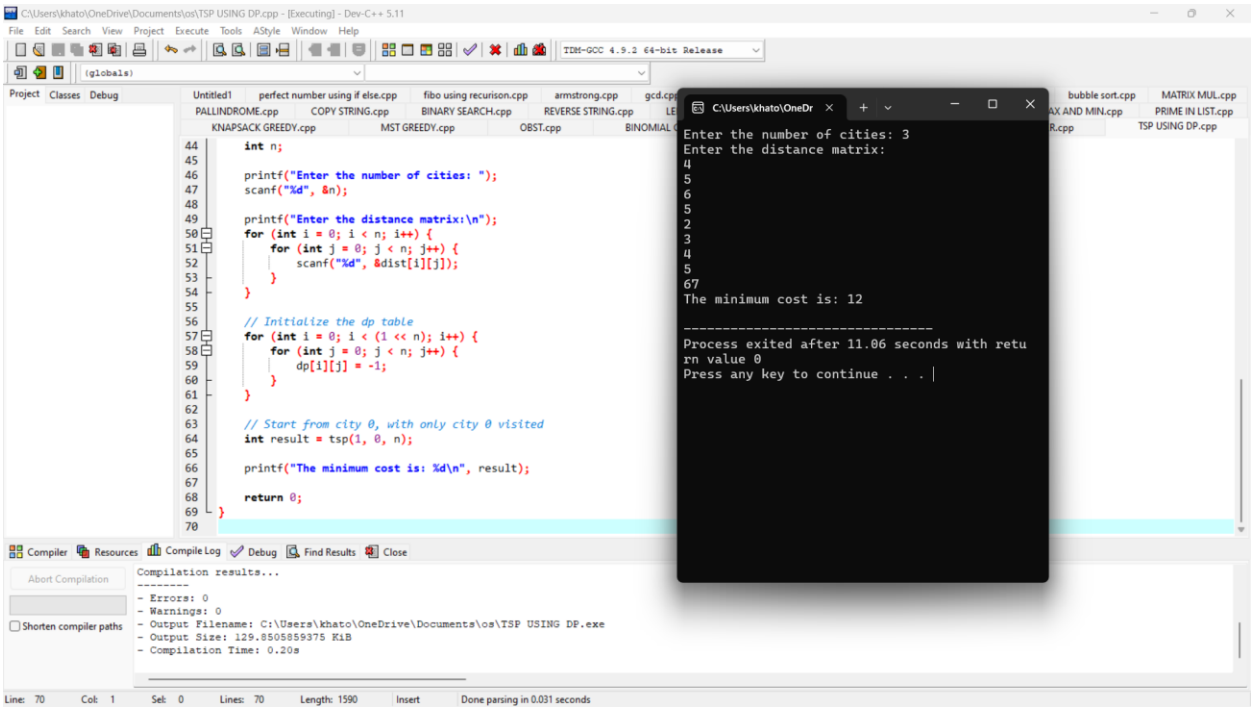
TSP USING DYNAMIC PROGRAMMING

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

#define MAX 16
#define INF 1000000

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

```
// Function to find the minimum of two numbers
int min(int a, int b) {
    return (a < b) ? a : b;
}

// Recursive function to solve TSP
int tsp(int mask, int pos, int n) {
    // If all cities have been visited, return to the starting point
    if (mask == (1 << n) - 1) {
        return dist[pos][0];
    }

    // If this state has already been computed
    if (dp[mask][pos] != -1) {
        return dp[mask][pos];
    }

    int answer = INF;

    // Try to visit all unvisited cities
    for (int city = 0; city < n; city++) {
        // If city is not visited
        if ((mask & (1 << city)) == 0) {
```

```
            int newAnswer = dist[pos][city] + tsp(mask | (1 << city), city, n);
            answer = min(answer, newAnswer);
        }
    }

    // Store and return the result
    dp[mask][pos] = answer;
    return answer;
}

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]);
        }
    }

    // Initialize the dp table
```

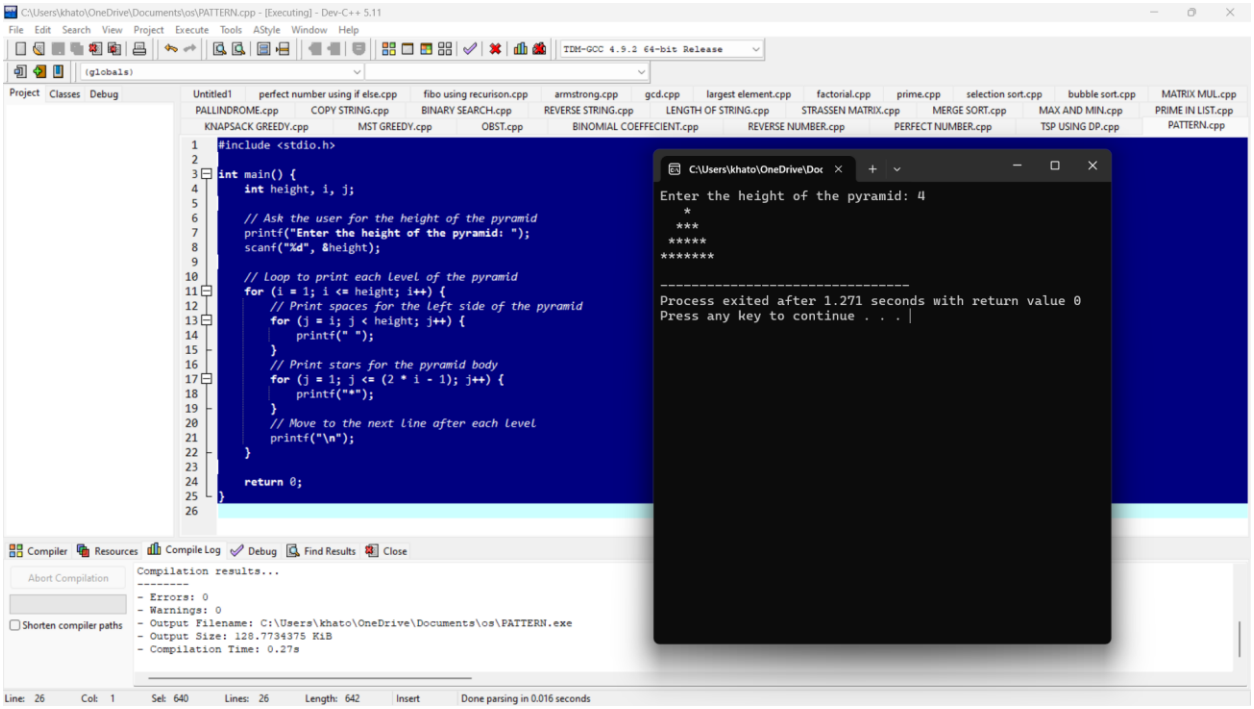
```
    for (int i = 0; i < (1 << n); i++) {
        for (int j = 0; j < n; j++) {
            dp[i][j] = -1;
        }
    }

    // Start from city 0, with only city 0 visited
    int result = tsp(1, 0, n);

    printf("The minimum cost is: %d\n", result);

    return 0;
}
```



PATTERN

```c
#include <stdio.h>

int main() {
    int height, i, j;

    // Ask the user for the height of the pyramid
    printf("Enter the height of the pyramid: ");
    scanf("%d", &height);

    // Loop to print each level of the pyramid
    for (i = 1; i <= height; i++) {
        // Print spaces for the left side of the pyramid
        for (j = i; j < height; j++) {
            printf(" ");
        }
        // Print stars for the pyramid body
        for (j = 1; j <= (2 * i - 1); j++) {
            printf("*");
        }
        // Move to the next line after each level
        printf("\n");
    }

    return 0;
```

}

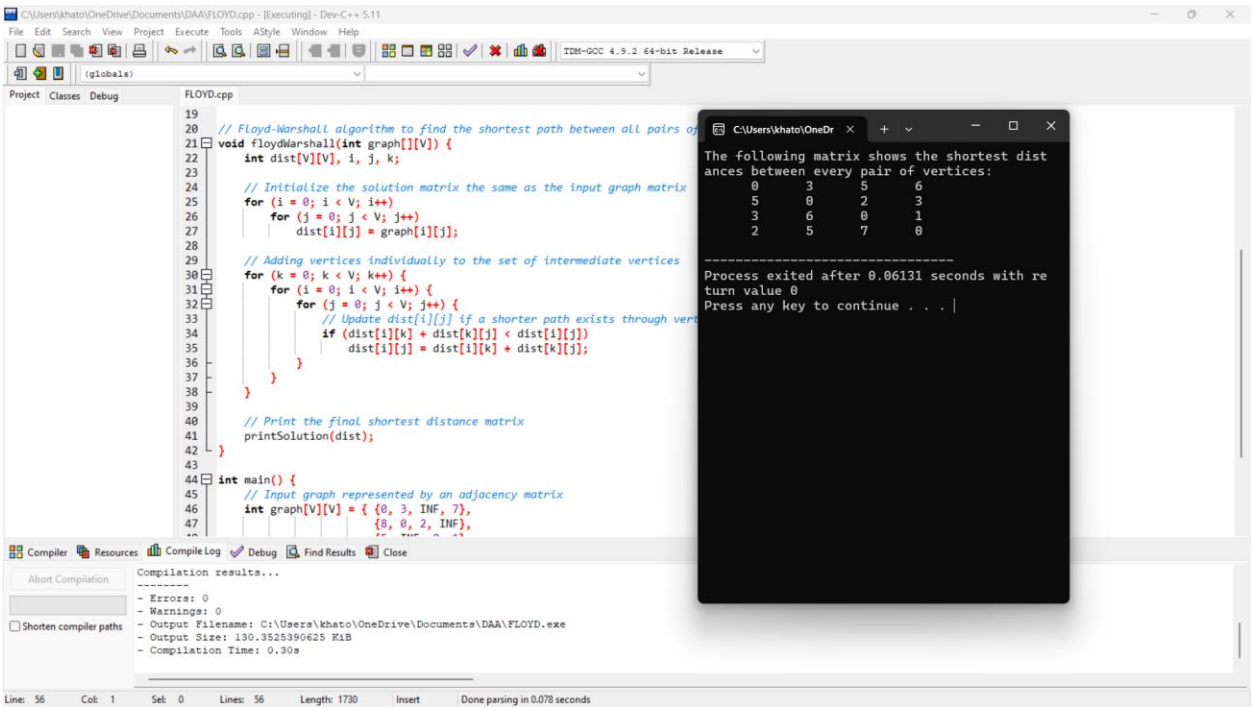


FLOYD ALGORITHM

```
#include <stdio.h>


#define INF 99999  // A large value to represent infinity
#define V 4        // Number of vertices in the graph


// Function to print the solution matrix
void printSolution(int dist[][V]) {
    printf("The following matrix shows the shortest distances between every pair of vertices:\n");
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (dist[i][j] == INF)
                printf("%7s", "INF");
```

```
            else
                printf("%7d", dist[i][j]);
        }
        printf("\n");
    }
}

// Floyd-Warshall algorithm to find the shortest path between all pairs of vertices
void floydWarshall(int graph[][V]) {
    int dist[V][V], i, j, k;

    // Initialize the solution matrix the same as the input graph matrix
    for (i = 0; i < V; i++)
        for (j = 0; j < V; j++)
            dist[i][j] = graph[i][j];

    // Adding vertices individually to the set of intermediate vertices
    for (k = 0; k < V; k++) {
        for (i = 0; i < V; i++) {
            for (j = 0; j < V; j++) {
                // Update dist[i][j] if a shorter path exists through vertex k
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
```

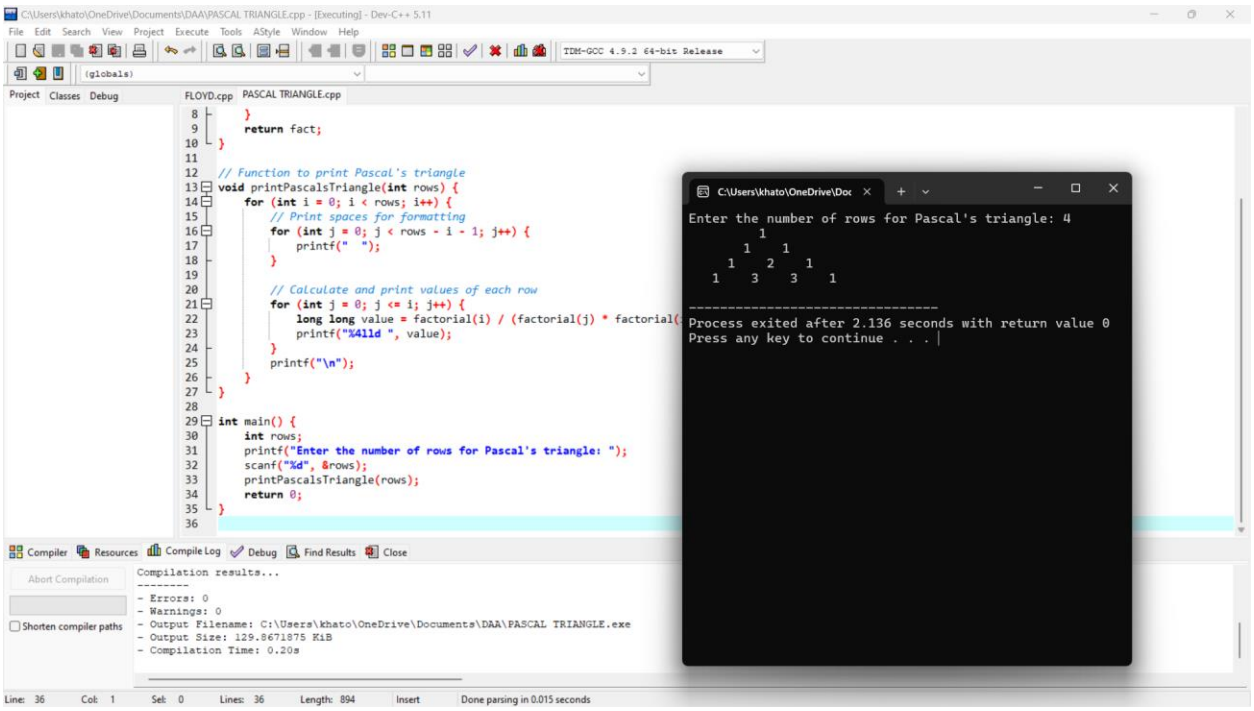
```
    }

    // Print the final shortest distance matrix
    printSolution(dist);
}

int main() {
    // Input graph represented by an adjacency matrix
    int graph[V][V] = { {0, 3, INF, 7},
                        {8, 0, 2, INF},
                        {5, INF, 0, 1},
                        {2, INF, INF, 0} };

    // Execute the Floyd-Warshall algorithm
    floydWarshall(graph);

    return 0;
}
```



PASCAL TRIANGLE

```
#include <stdio.h>

// Function to calculate factorial of a number
long long factorial(int n) {
    long long fact = 1;
    for (int i = 1; i <= n; i++) {
        fact *= i;
    }
    return fact;
}

// Function to print Pascal's triangle
void printPascalsTriangle(int rows) {
    for (int i = 0; i < rows; i++) {
```

```
        // Print spaces for formatting
        for (int j = 0; j < rows - i - 1; j++) {
            printf("  ");
        }

        // Calculate and print values of each row
        for (int j = 0; j <= i; j++) {
            long long value = factorial(i) / (factorial(j) * factorial(i - j));
            printf("%4lld ", value);
        }
        printf("\n");
    }
}

int main() {
    int rows;
    printf("Enter the number of rows for Pascal's triangle: ");
    scanf("%d", &rows);
    printPascalsTriangle(rows);
    return 0;
}
```

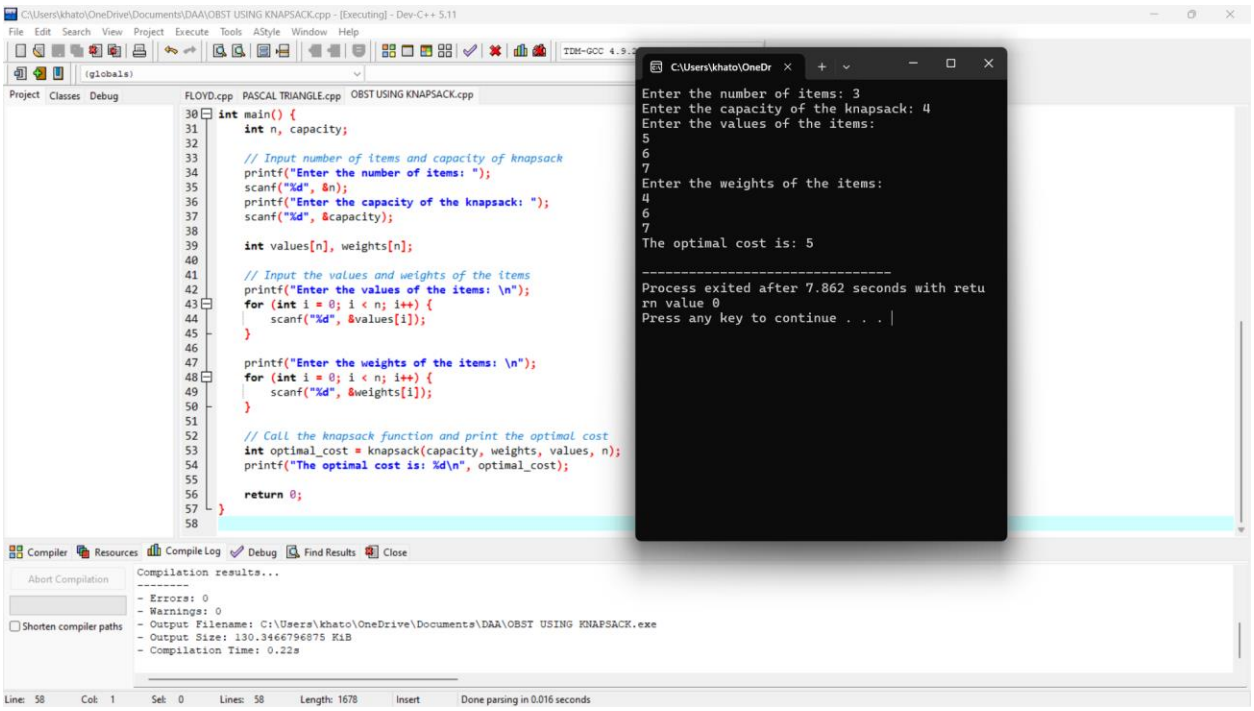


OPTIMAL COST USING KNAPSACK

```
#include <stdio.h>

// Function to find the maximum of two numbers
int max(int a, int b) {
    return (a > b) ? a : b;
}

// Function to solve the 0/1 Knapsack problem
int knapsack(int capacity, int weights[], int values[], int n) {
    int i, w;
    int dp[n + 1][capacity + 1]; // Create a 2D DP array

    // Build the DP table in a bottom-up manner
    for (i = 0; i <= n; i++) {
```

```
        for (w = 0; w <= capacity; w++) {
            if (i == 0 || w == 0) {
                dp[i][w] = 0;  // Base case: no items or zero capacity
            } else if (weights[i - 1] <= w) {
                dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w]);
            } else {
                dp[i][w] = dp[i - 1][w];
            }
        }
    }

    // Return the maximum value that can be put in the knapsack
    return dp[n][capacity];
}

int main() {
    int n, capacity;

    // Input number of items and capacity of knapsack
    printf("Enter the number of items: ");
    scanf("%d", &n);
    printf("Enter the capacity of the knapsack: ");
    scanf("%d", &capacity);

    int values[n], weights[n];
```

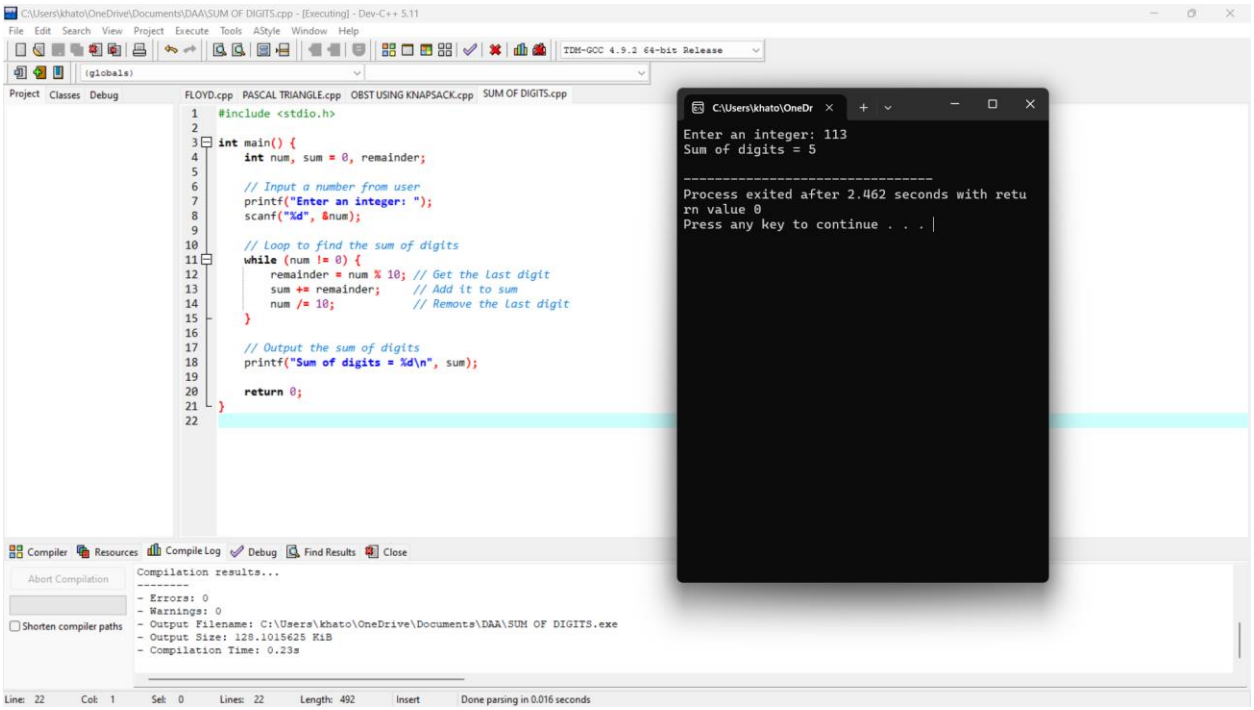
```
    // Input the values and weights of the items
    printf("Enter the values of the items: \n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &values[i]);
    }

    printf("Enter the weights of the items: \n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &weights[i]);
    }

    // Call the knapsack function and print the optimal cost
    int optimal_cost = knapsack(capacity, weights, values, n);
    printf("The optimal cost is: %d\n", optimal_cost);

    return 0;
}
```



SUM OF DIGITS

```
#include <stdio.h>


int main() {
    int num, sum = 0, remainder;


    // Input a number from user
    printf("Enter an integer: ");
    scanf("%d", &num);


    // Loop to find the sum of digits
    while (num != 0) {
        remainder = num % 10; // Get the last digit
        sum += remainder;     // Add it to sum
        num /= 10;            // Remove the last digit
```

```
    }

    // Output the sum of digits
    printf("Sum of digits = %d\n", sum);

    return 0;
}
```

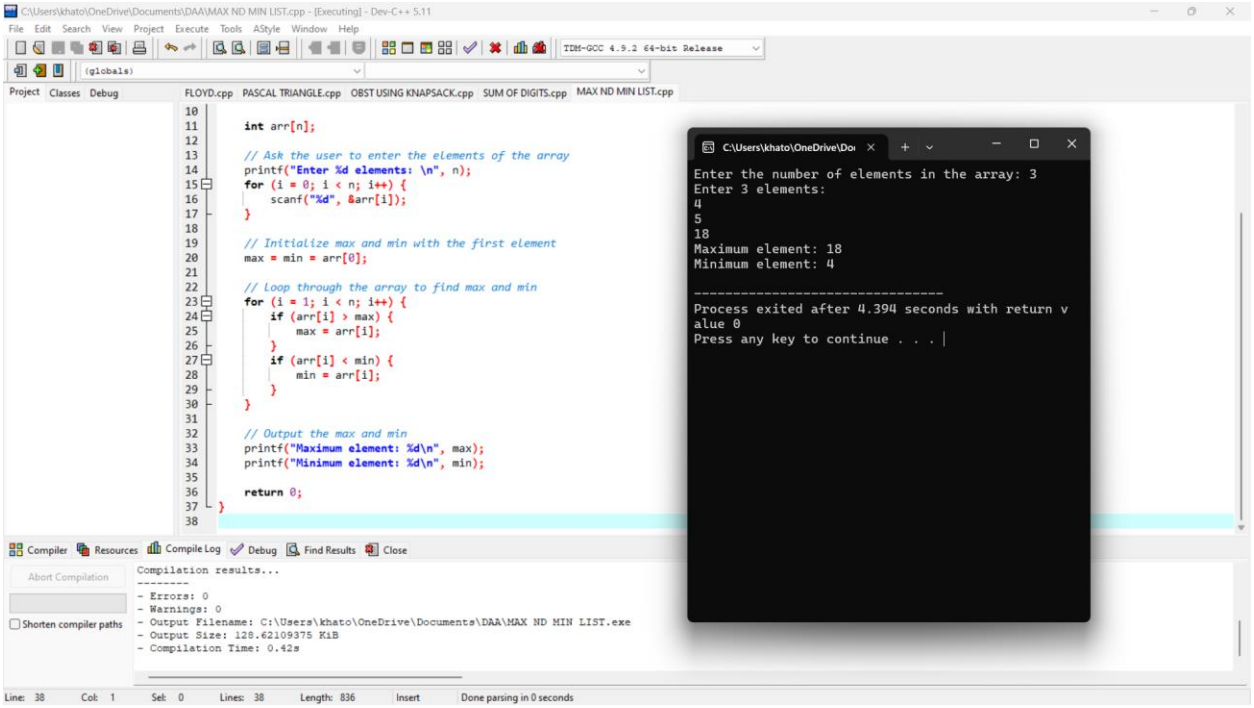


MAX ND MIN IN LIST

```
#include <stdio.h>

int main() {
    int n, i;
    int max, min;

    // Ask the user for the number of elements
```

```
printf("Enter the number of elements in the array: ");
scanf("%d", &n);

int arr[n];

// Ask the user to enter the elements of the array
printf("Enter %d elements: \n", n);
for (i = 0; i < n; i++) {
    scanf("%d", &arr[i]);
}

// Initialize max and min with the first element
max = min = arr[0];

// Loop through the array to find max and min
for (i = 1; i < n; i++) {
    if (arr[i] > max) {
        max = arr[i];
    }
    if (arr[i] < min) {
        min = arr[i];
    }
}

// Output the max and min
```

```
    printf("Maximum element: %d\n", max);
    printf("Minimum element: %d\n", min);

    return 0;
}
```



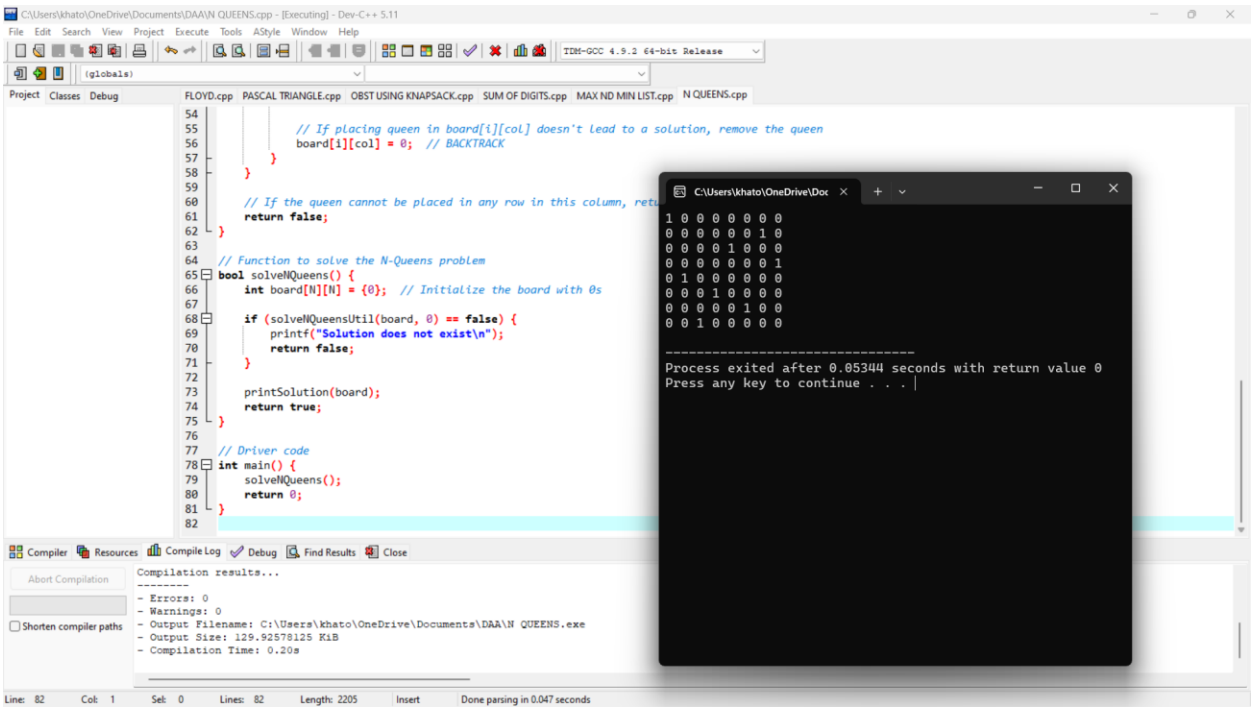
N QUEENS USING BACKTRACKING

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

#define N 8  // You can change this value for a different size board

// Function to print the solution
void printSolution(int board[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
```

```
            printf("%d ", board[i][j]);
        }
        printf("\n");
    }
}

// Function to check if a queen can be placed on board[row][col]
bool isSafe(int board[N][N], int row, int col) {
    int i, j;

    // Check this row on left side
    for (i = 0; i < col; i++)
        if (board[row][i])
            return false;

    // Check upper diagonal on left side
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
            return false;

    // Check lower diagonal on left side
    for (i = row, j = col; j >= 0 && i < N; i++, j--)
        if (board[i][j])
            return false;
```

```
    return true;
}

// Function to solve the N-Queens problem using backtracking
bool solveNQueensUtil(int board[N][N], int col) {
    // Base case: If all queens are placed, return true
    if (col >= N)
        return true;

    // Consider this column and try placing this queen in all rows one by one
    for (int i = 0; i < N; i++) {
        // Check if the queen can be placed on board[i][col]
        if (isSafe(board, i, col)) {
            // Place the queen
            board[i][col] = 1;

            // Recur to place the rest of the queens
            if (solveNQueensUtil(board, col + 1))
                return true;

            // If placing queen in board[i][col] doesn't lead to a solution, remove the queen
            board[i][col] = 0;  // BACKTRACK
        }
    }
```

```
    // If the queen cannot be placed in any row in this column, return false
    return false;
}

// Function to solve the N-Queens problem
bool solveNQueens() {
    int board[N][N] = {0};  // Initialize the board with 0s

    if (solveNQueensUtil(board, 0) == false) {
        printf("Solution does not exist\n");
        return false;
    }

    printSolution(board);
    return true;
}

// Driver code
int main() {
    solveNQueens();
    return 0;
}
```

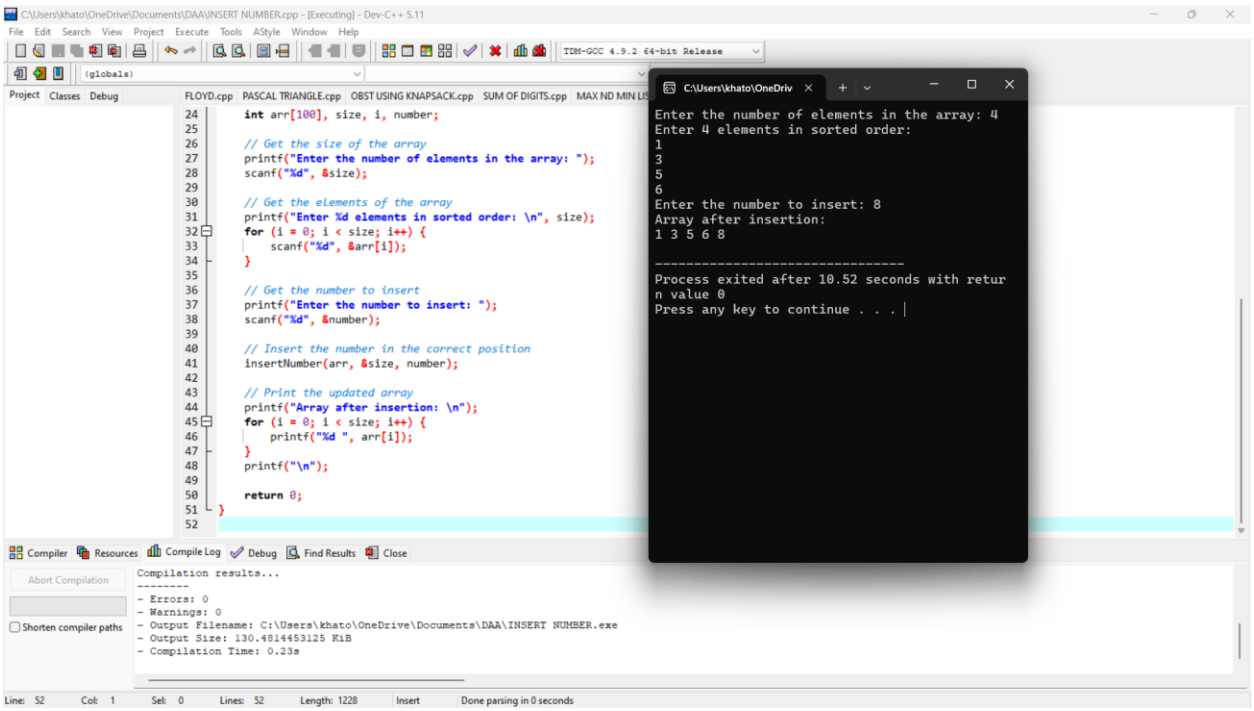


33. INSERT A NUMBER IN LIST

```
#include <stdio.h>


void insertNumber(int arr[], int *size, int number) {
    int i, j;

    // Find the correct position for the number
    for (i = 0; i < *size; i++) {
        if (arr[i] > number) {
            break;
        }
    }

    // Shift elements to the right to make space for the new number
    for (j = *size; j > i; j--) {
```

```
        arr[j] = arr[j - 1];
    }

    // Insert the number
    arr[i] = number;
    (*size)++; // Increment the size of the array
}

int main() {
    int arr[100], size, i, number;

    // Get the size of the array
    printf("Enter the number of elements in the array: ");
    scanf("%d", &size);

    // Get the elements of the array
    printf("Enter %d elements in sorted order: \n", size);
    for (i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }

    // Get the number to insert
    printf("Enter the number to insert: ");
    scanf("%d", &number);
```

```
    // Insert the number in the correct position
    insertNumber(arr, &size, number);

    // Print the updated array
    printf("Array after insertion: \n");
    for (i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

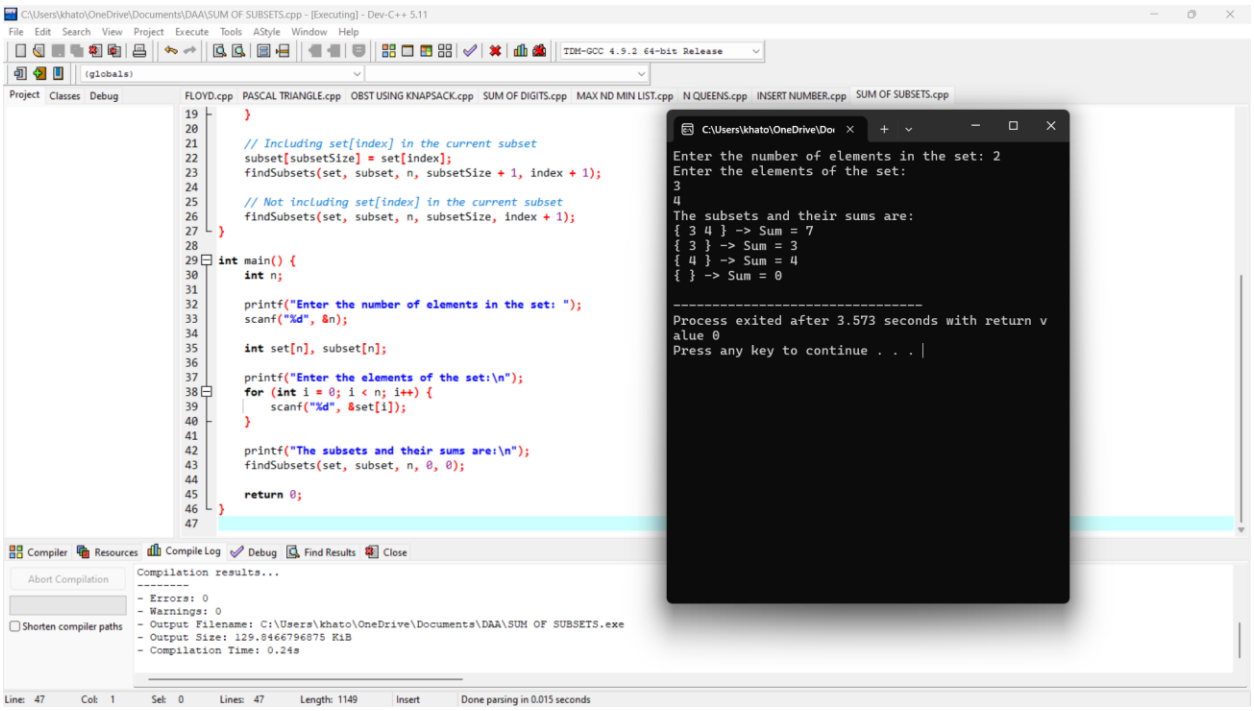


34. SUM OF SUBSETS

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

```
// Function to print a subset
void printSubset(int subset[], int size) {
    int sum = 0;
    printf("{ ");
    for (int i = 0; i < size; i++) {
        printf("%d ", subset[i]);
        sum += subset[i];
    }
    printf("} -> Sum = %d\n", sum);
}

// Recursive function to find the subsets
void findSubsets(int set[], int subset[], int n, int subsetSize, int index) {
    if (index == n) {
        printSubset(subset, subsetSize);
        return;
    }

    // Including set[index] in the current subset
    subset[subsetSize] = set[index];
    findSubsets(set, subset, n, subsetSize + 1, index + 1);

    // Not including set[index] in the current subset
    findSubsets(set, subset, n, subsetSize, index + 1);
}
```

```
int main() {
    int n;

    printf("Enter the number of elements in the set: ");
    scanf("%d", &n);

    int set[n], subset[n];

    printf("Enter the elements of the set:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &set[i]);
    }

    printf("The subsets and their sums are:\n");
    findSubsets(set, subset, n, 0, 0);

    return 0;
}
```



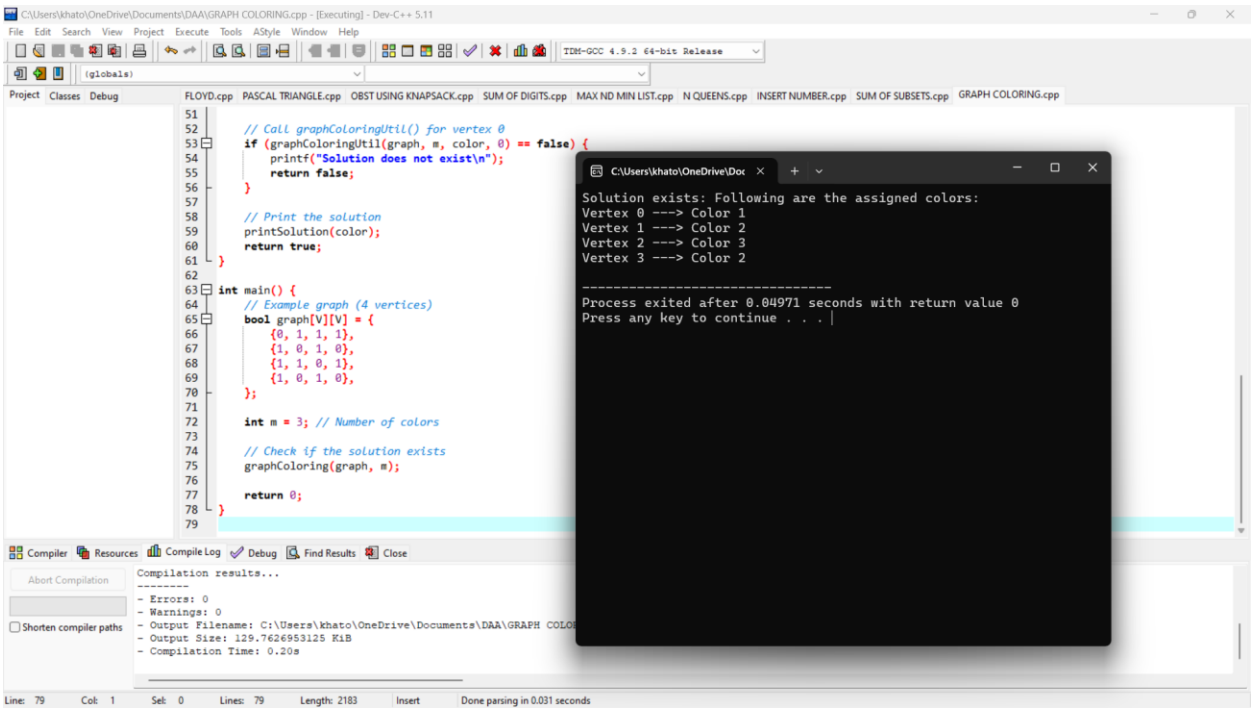
35. GRAPH COLORING

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

#define V 4  // Number of vertices

// Function to print the solution
void printSolution(int color[]) {
    printf("Solution exists: Following are the assigned colors:\n");
    for (int i = 0; i < V; i++)
        printf("Vertex %d ---> Color %d\n", i, color[i]);
}

// Function to check if the current color assignment is safe
bool isSafe(int v, bool graph[V][V], int color[], int c) {
```

```
    for (int i = 0; i < V; i++) {
        if (graph[v][i] && c == color[i]) // If adjacent and color is the same
            return false;
    }
    return true;
}

// A recursive utility function to solve the graph coloring problem
bool graphColoringUtil(bool graph[V][V], int m, int color[], int v) {
    // Base case: If all vertices are assigned a color then return true
    if (v == V)
        return true;

    // Consider this vertex v and try different colors
    for (int c = 1; c <= m; c++) {
        // Check if assignment of color c to v is valid
        if (isSafe(v, graph, color, c)) {
            color[v] = c;

            // Recursively assign colors to the rest of the vertices
            if (graphColoringUtil(graph, m, color, v + 1))
                return true;

            // If assigning color c doesn't lead to a solution, remove it
            color[v] = 0;
```

```
        }
    }

    // If no color can be assigned, return false
    return false;
}

// Function to solve the m Coloring problem
bool graphColoring(bool graph[V][V], int m) {
    // Initialize all color values as 0 (unassigned)
    int color[V] = {0};

    // Call graphColoringUtil() for vertex 0
    if (graphColoringUtil(graph, m, color, 0) == false) {
        printf("Solution does not exist\n");
        return false;
    }

    // Print the solution
    printSolution(color);
    return true;
}

int main() {
    // Example graph (4 vertices)
```

```
    bool graph[V][V] = {
        {0, 1, 1, 1},
        {1, 0, 1, 0},
        {1, 1, 0, 1},
        {1, 0, 1, 0},
    };

    int m = 3; // Number of colors

    // Check if the solution exists
    graphColoring(graph, m);

    return 0;
}
```

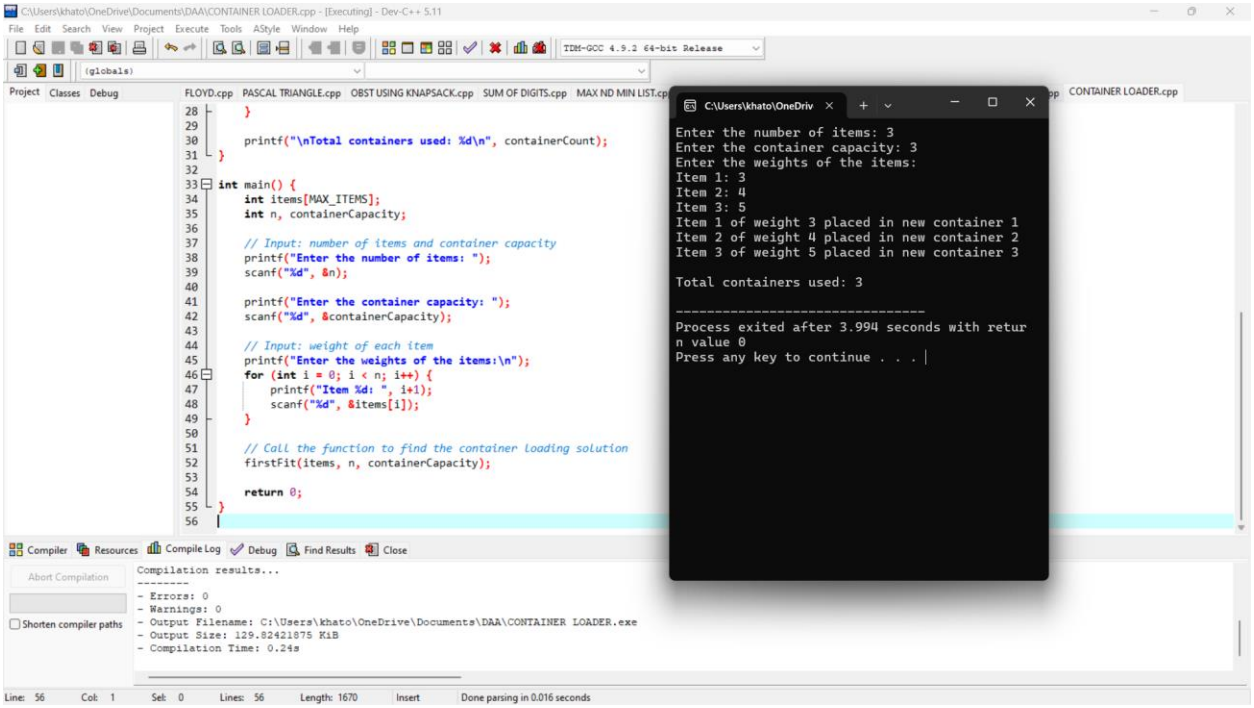


36. CONTAINER LOADER

```
#include <stdio.h>

#define MAX_ITEMS 100
#define MAX_CONTAINERS 100

void firstFit(int items[], int n, int containerCapacity) {
    int containers[MAX_CONTAINERS] = {0};  // Array to store remaining capacity of containers
    int containerCount = 0;

    for (int i = 0; i < n; i++) {
        int j;

        // Try to fit the item in an existing container
        for (j = 0; j < containerCount; j++) {
            if (containers[j] >= items[i]) {
                containers[j] -= items[i];
                printf("Item %d of weight %d placed in container %d\n", i+1, items[i], j+1);
                break;
            }
        }

        // If it doesn't fit in any existing container, use a new one
        if (j == containerCount) {
            containers[containerCount] = containerCapacity - items[i];
```

```
            printf("Item %d of weight %d placed in new container %d\n", i+1, items[i],
containerCount+1);
            containerCount++;
        }
    }

    printf("\nTotal containers used: %d\n", containerCount);
}

int main() {
    int items[MAX_ITEMS];
    int n, containerCapacity;

    // Input: number of items and container capacity
    printf("Enter the number of items: ");
    scanf("%d", &n);

    printf("Enter the container capacity: ");
    scanf("%d", &containerCapacity);

    // Input: weight of each item
    printf("Enter the weights of the items:\n");
    for (int i = 0; i < n; i++) {
        printf("Item %d: ", i+1);
        scanf("%d", &items[i]);
    }
```

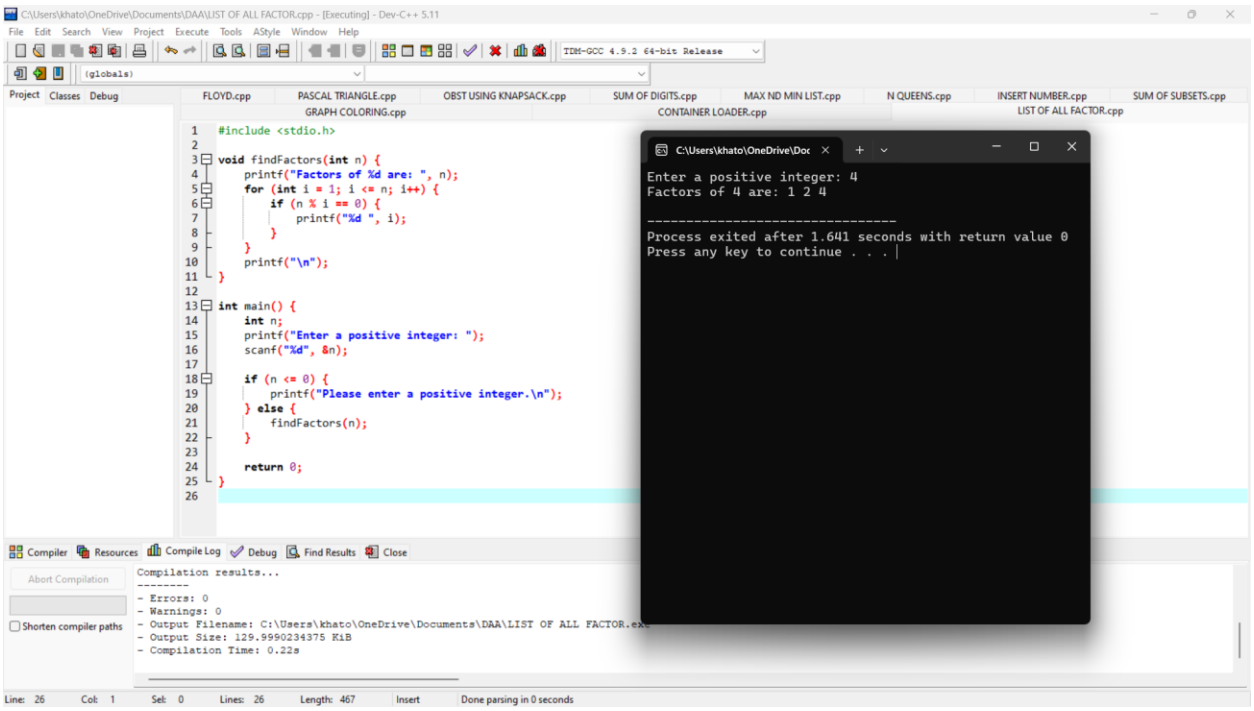
```
    // Call the function to find the container loading solution
    firstFit(items, n, containerCapacity);

    return 0;
}
```



37. LIST OF ALL FACTOR

```
#include <stdio.h>

void findFactors(int n) {
    printf("Factors of %d are: ", n);
    for (int i = 1; i <= n; i++) {
        if (n % i == 0) {
            printf("%d ", i);
        }
```

```
    }
    printf("\n");
}

int main() {
    int n;
    printf("Enter a positive integer: ");
    scanf("%d", &n);

    if (n <= 0) {
        printf("Please enter a positive integer.\n");
    } else {
        findFactors(n);
    }

    return 0;
}
```

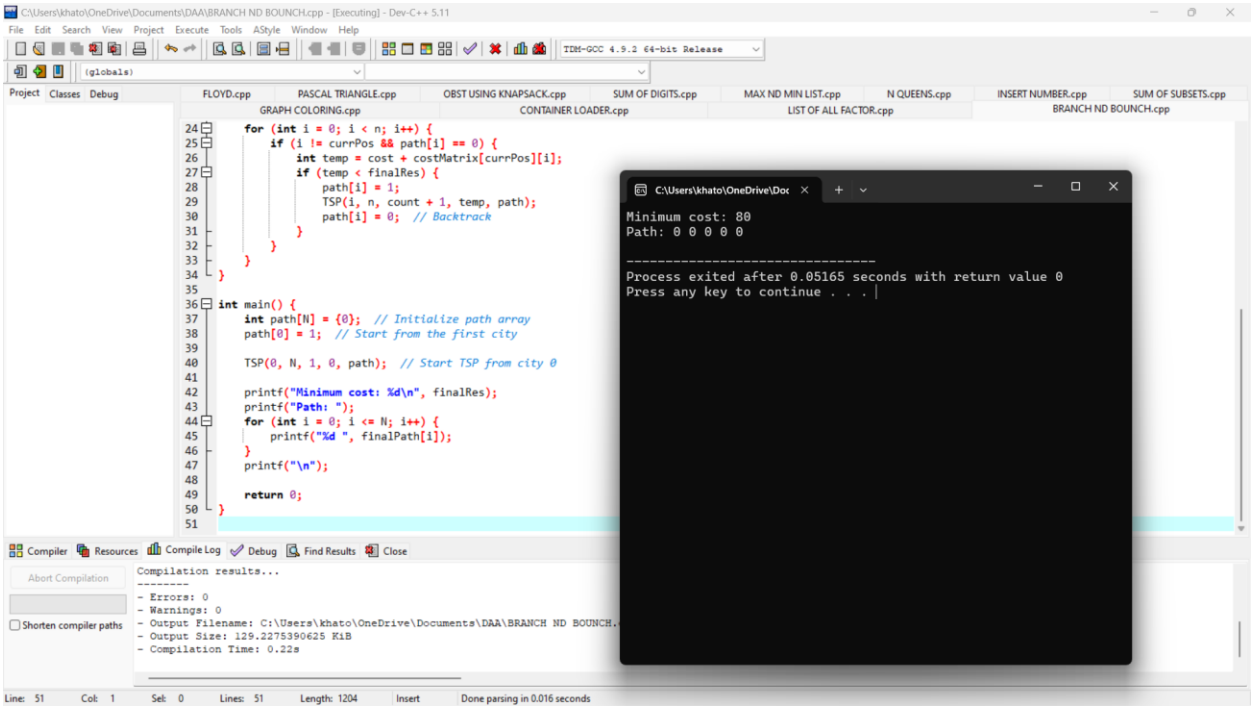


38. BRANCH AND BOUND

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

#define N 4  // Number of cities

int costMatrix[N][N] = {
    {0, 10, 15, 20},
    {10, 0, 35, 25},
    {15, 35, 0, 30},
    {20, 25, 30, 0}
};

int finalRes = INT_MAX;
```

```
int finalPath[N + 1];

void TSP(int currPos, int n, int count, int cost, int path[]) {
    if (count == n && cost + costMatrix[currPos][0] < finalRes) {
        finalPath[count] = 0;
        finalRes = cost + costMatrix[currPos][0];
        return;
    }

    for (int i = 0; i < n; i++) {
        if (i != currPos && path[i] == 0) {
            int temp = cost + costMatrix[currPos][i];
            if (temp < finalRes) {
                path[i] = 1;
                TSP(i, n, count + 1, temp, path);
                path[i] = 0;  // Backtrack
            }
        }
    }
}

int main() {
    int path[N] = {0};  // Initialize path array
    path[0] = 1;  // Start from the first city
```

```
    TSP(0, N, 1, 0, path);  // Start TSP from city 0

    printf("Minimum cost: %d\n", finalRes);
    printf("Path: ");
    for (int i = 0; i <= N; i++) {
        printf("%d ", finalPath[i]);
    }
    printf("\n");

    return 0;
}
```

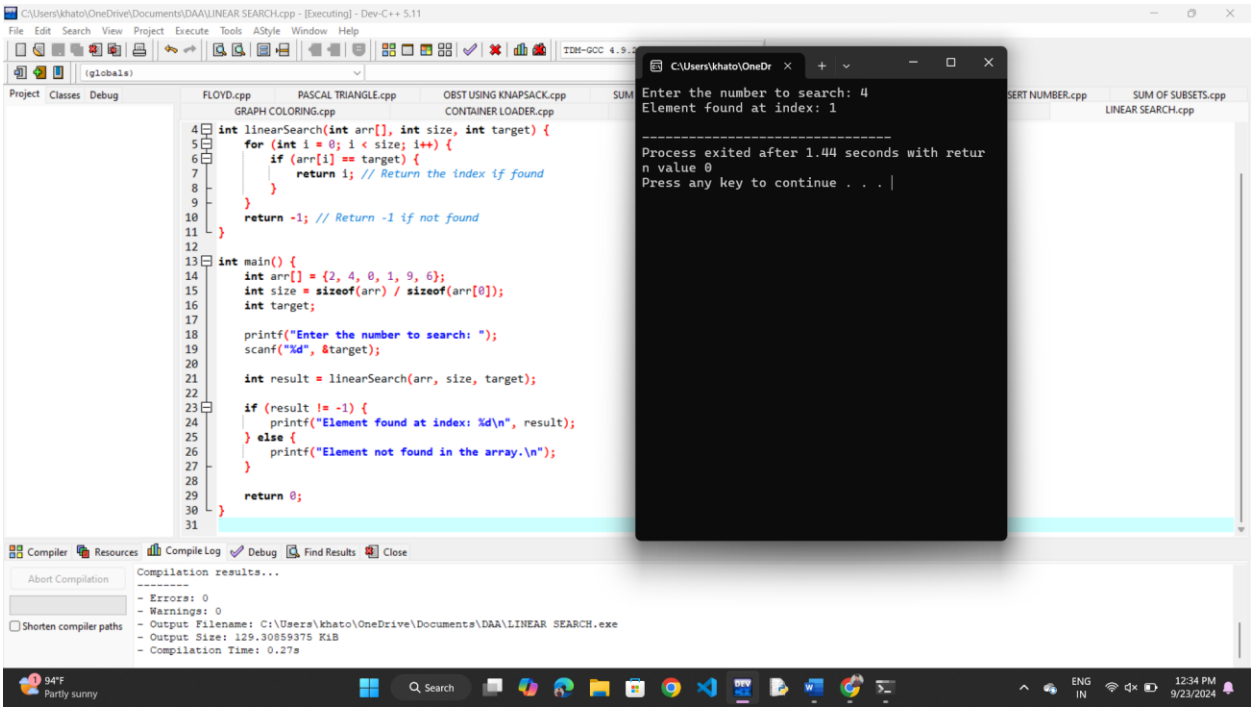


39. LINEAR SEARCH

```
#include <stdio.h>

// Function for linear search
```

```
int linearSearch(int arr[], int size, int target) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == target) {
            return i; // Return the index if found
        }
    }
    return -1; // Return -1 if not found
}

int main() {
    int arr[] = {2, 4, 0, 1, 9, 6};
    int size = sizeof(arr) / sizeof(arr[0]);
    int target;

    printf("Enter the number to search: ");
    scanf("%d", &target);

    int result = linearSearch(arr, size, target);

    if (result != -1) {
        printf("Element found at index: %d\n", result);
    } else {
        printf("Element not found in the array.\n");
    }
```

```
    return 0;
}
```



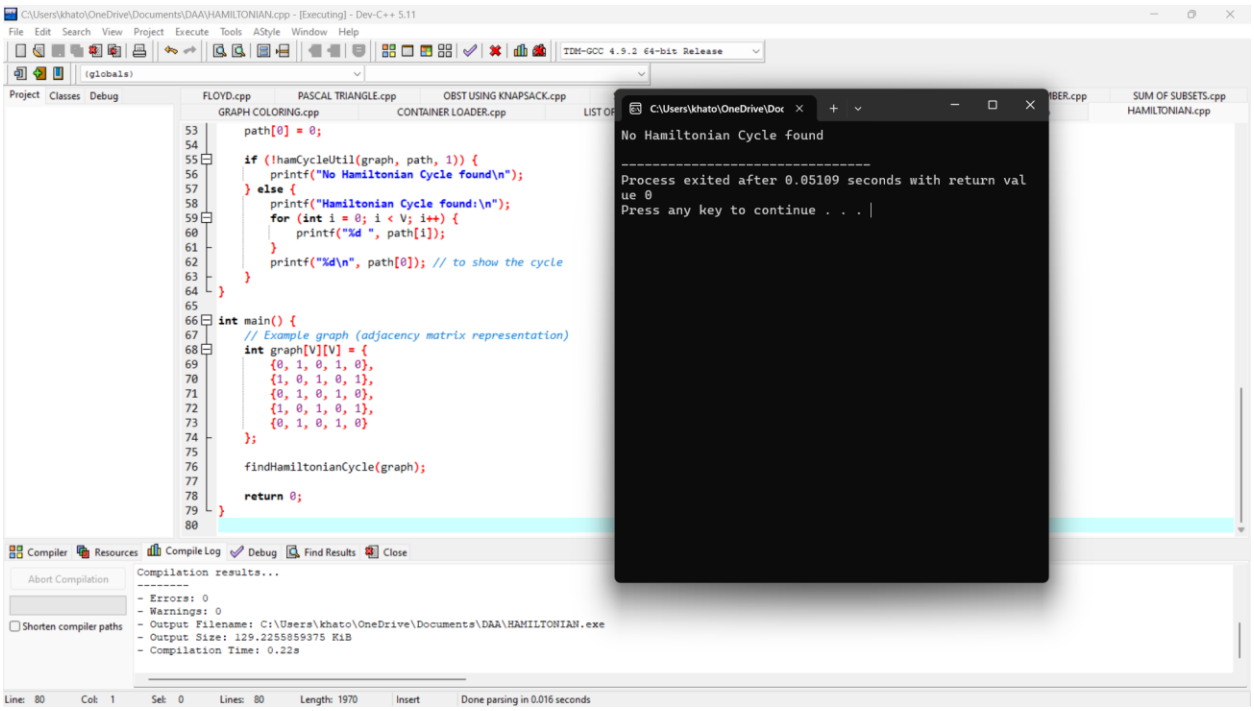
40. HAMILTONIAN

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

#define V 5 // Number of vertices in the graph

bool isSafe(int graph[V][V], int path[], int pos, int v) {
    // Check if this vertex is an adjacent vertex of the previously added vertex.
    if (graph[path[pos - 1]][v] == 0) {
        return false;
    }

    // Check if the vertex has already been included in the path.
```

```
    for (int i = 0; i < pos; i++) {
        if (path[i] == v) {
            return false;
        }
    }

    return true;
}

bool hamCycleUtil(int graph[V][V], int path[], int pos) {
    // Base case: If all vertices are included in the path
    if (pos == V) {
        // And if there is an edge from the last included vertex to the first vertex
        return graph[path[pos - 1]][path[0]] == 1;
    }

    // Try different vertices as the next candidate in the Hamiltonian Path
    for (int v = 1; v < V; v++) {
        if (isSafe(graph, path, pos, v)) {
            path[pos] = v;

            if (hamCycleUtil(graph, path, pos + 1)) {
                return true;
            }
```

```
            // Backtrack
            path[pos] = -1;
        }
    }

    return false;
}

void findHamiltonianCycle(int graph[V][V]) {
    int path[V];
    for (int i = 0; i < V; i++) {
        path[i] = -1;
    }

    // Start from the first vertex
    path[0] = 0;

    if (!hamCycleUtil(graph, path, 1)) {
        printf("No Hamiltonian Cycle found\n");
    } else {
        printf("Hamiltonian Cycle found:\n");
        for (int i = 0; i < V; i++) {
            printf("%d ", path[i]);
        }
        printf("%d\n", path[0]); // to show the cycle
```

```
    }
}

int main() {
    // Example graph (adjacency matrix representation)
    int graph[V][V] = {
        {0, 1, 0, 1, 0},
        {1, 0, 1, 0, 1},
        {0, 1, 0, 1, 0},
        {1, 0, 1, 0, 1},
        {0, 1, 0, 1, 0}
    };

    findHamiltonianCycle(graph);

    return 0;
}
```


C:\Users\khato\OneDrive\Documents\DAA\HAMILTONIAN.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
FLOYD.cpp
PASCAL TRIANGLE.cpp
OBST USING KNAPSACK.cpp
SUM OF SUBSETS.cpp
GRAPH COLORING.cpp
CONTAINER LOADER.cpp
HAMILTONIAN.cpp
path[0] = 0;
if (!hamCycleUtil(graph, path, 1)) {
printf("No Hamiltonian Cycle found\n");
} else {
printf("Hamiltonian Cycle found:\n");
for (int i = 0; i < V; i++) {
printf("%d ", path[i]);
}
printf("%d\n", path[0]); // to show the cycle
}
}
int main() {
// Example graph (adjacency matrix representation)
int graph[V][V] = {
{0, 1, 0, 1, 0},
{1, 0, 1, 0, 1},
{0, 1, 0, 1, 0},
{1, 0, 1, 0, 1},
{0, 1, 0, 1, 0}
};
findHamiltonianCycle(graph);
return 0;
}
C:\Users\khato\OneDrive\Doc
No Hamiltonian Cycle found
Process exited after 0.05109 seconds with return value 0
Press any key to continue . . .
Compiler Resources Compile Log Debug Find Results Close
Abort Compilation
Shorten compiler paths
Compilation results...
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\khato\OneDrive\Documents\DAA\HAMILTONIAN.exe
- Output Size: 129.2255859375 KiB
- Compilation Time: 0.22s
Line: 80 Col: 1 Sel: 0 Lines: 80 Length: 1970 Insert Done parsing in 0.016 seconds