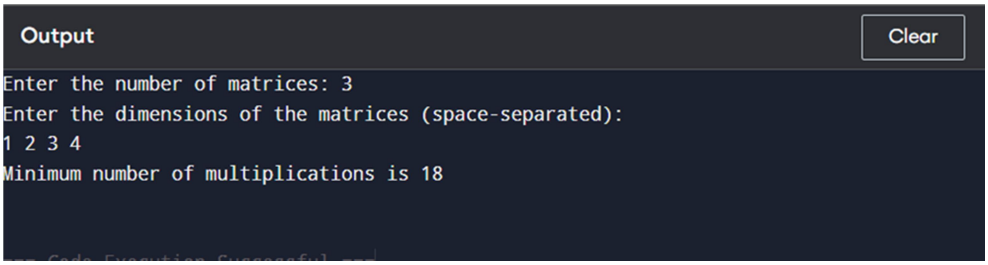
DAY - 8

Student Name: Abhay Bansal **UID: 22BCS15306**
Branch: BE-CSE **Section/Group: 620-A**
Semester: 5th **Date of Performance:** 28/12/24

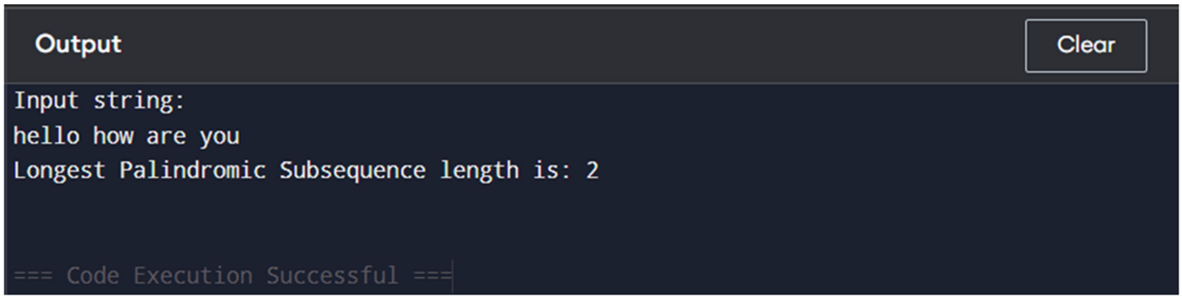
1. Write a program to find the most efficient method to multiply a matrix

```
#include <iostream>
#include <climits>
using namespace std;

int matrixChainOrder(int p[], int n) {
    int dp[n][n];

    for (int i = 1; i < n; ++i) {
        dp[i][i] = 0;
    }
    for (int L = 2; L < n; ++L) {
        for (int i = 1; i < n - L + 1; ++i) {
            int j = i + L - 1;
            dp[i][j] = INT_MAX;
            for (int k = i; k < j; ++k) {
                int cost = dp[i][k] + dp[k + 1][j] + p[i - 1] * p[k] * p[j];
                if (cost < dp[i][j]) {
                    dp[i][j] = cost;
                }
            }
        }
    }
    return dp[1][n - 1];
}

int main() {
    int n;
    cout << "Enter the number of matrices: ";
    cin >> n;
    int p[n + 1];
    cout << "Enter the dimensions of the matrices (space-separated):\n";
    for (int i = 0; i <= n; ++i) {
        cin >> p[i];
    }
    cout << "Minimum number of multiplications is " << matrixChainOrder(p, n + 1) << endl;
    return 0;
}
```



2. Write a program to find the longest palindromic subsequence in a string

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

int longestPalindromicSubsequence(string &str) {
    int n = str.length();
    int dp[n][n];

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

    for (int cl = 2; cl <= n; ++cl) {
        for (int i = 0; i < n - cl + 1; ++i) {
            int j = i + cl - 1;
            if (str[i] == str[j] && cl == 2) {
                dp[i][j] = 2;
            } else if (str[i] == str[j]) {
                dp[i][j] = dp[i + 1][j - 1] + 2;
            } else {
                dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
            }
        }
    }
    return dp[0][n - 1];
}

int main() {
    string str;
    cout << "Input string: " << endl;
    cin >> str;
    cout << "Longest Palindromic Subsequence length is: " << longestPalindromicSubsequence(str) <<
endl;

    return 0;
}
```

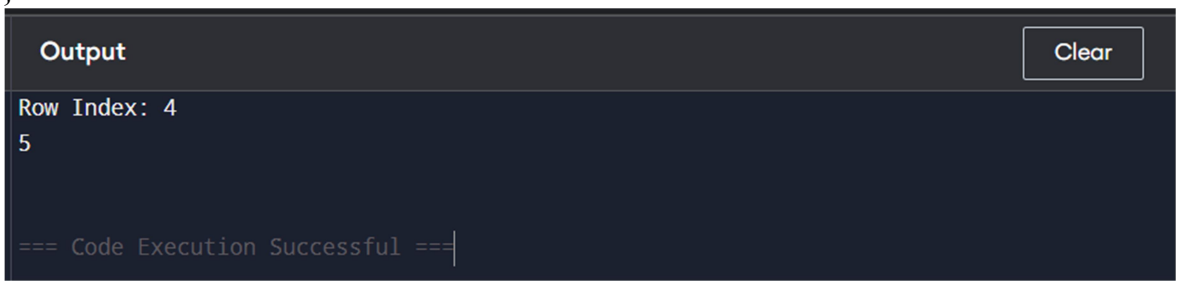


3. The Tribonacci sequence Tn is defined as follows:
T0 = 0, T1 = 1, T2 = 1, and Tn+3 = Tn + Tn+1 + Tn+2 for n >= 0.
Given n, return the value of Tn.

```
#include <iostream>
#include <cmath>
using namespace std;

int tribonacci(int n) {
    if (n == 0) return 0;
    if (n == 1 || n == 2) return 1;
    int t0 = 0, t1 = 1, t2 = 1, tn;

    for (int i = 3; i <= n; ++i) {
        tn = t0 + t1 + t2;
        t0 = t1;
        t1 = t2;
        t2 = tn;
    }
   return t2;
}

int main() {
    int n;
    cout<<"Input n: ";
    cin >> n;

    cout << "Tribonacci number T" << n << " is: " << tribonacci(n) << endl;

    return 0;
}
```

4. Given an integer rowIndex, return the rowIndexth (0-indexed) row of the Pascal's triangle.

```
#include <iostream>
using namespace std;

int main() {
    int rowIndex;
    cout << "Row Index: ";
    cin >> rowIndex;

    int prev1 = 1, prev2 = 1, current = 0;

    if (rowIndex == 0) {
        cout << "1" << endl;
    } else {
        for (int i = 2; i <= rowIndex; ++i) {
            current = prev1 + prev2;
            prev1 = prev2;
            prev2 = current;
        }
        cout << prev2 << endl;
    }

    return 0;
}
```



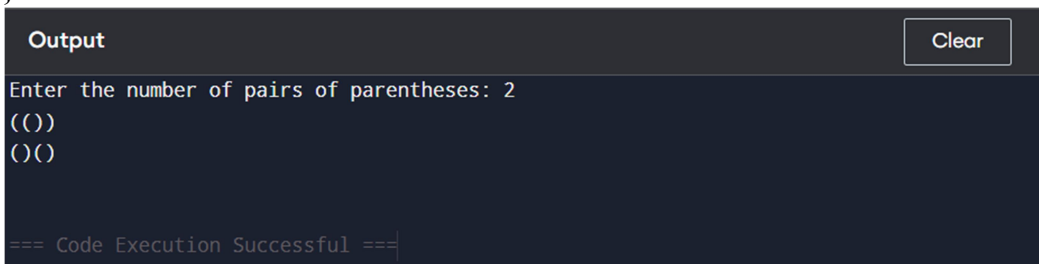
5. Given an integer n, return an array ans of length n + 1 such that for each i (0 <= i <= n), ans[i] is the number of 1's in the binary representation of i.

```
#include <iostream>
using namespace std;

int main() {
    cout << "Enter a number: ";
    int n;
    cin >> n;

    for (int i = 0; i <= n; ++i) {
        int count = 0, temp = i;
        while (temp) {
            count += temp & 1;
```

```
        temp >>= 1;
      }
      cout << count << " ";
   }
   return 0;
}
```

6. Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

```
#include <iostream>
#include <string>
using namespace std;

void generateParenthesesHelper(int n, int open, int close, string current) {
   if (current.size() == n * 2) {
      cout << current << endl;
      return;
   }
   if (open < n) {
      generateParenthesesHelper(n, open + 1, close, current + '(');
   }
   if (close < open) {
      generateParenthesesHelper(n, open, close + 1, current + ')');
   }
}

int main() {
   cout << "Enter the number of pairs of parentheses: ";
   int n;
   cin >> n;
   generateParenthesesHelper(n, 0, 0, "");
   return 0;
}
```



7. You are given an integer array nums. You are initially positioned at the array's first index, and each element in the array represents your maximum jump length at that position.
Return true if you can reach the last index, or false otherwise.

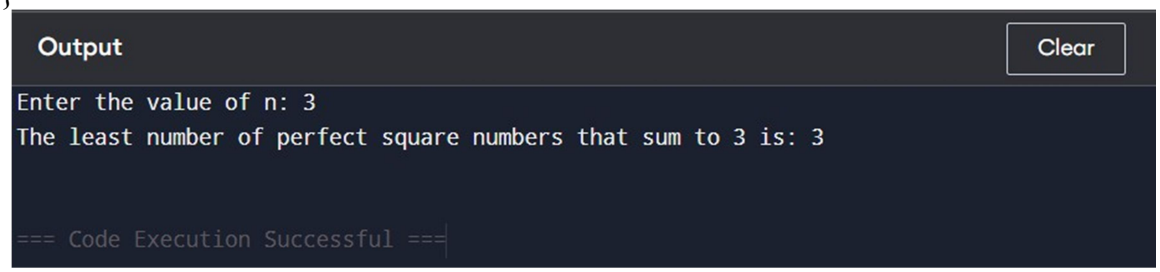
```
#include <iostream>
#include <climits>
using namespace std;

bool canJump(int arr[], int n) {
    int maxReach = 0;
    for (int i = 0; i < n; ++i) {
        if (i > maxReach) {
            return false;
        }
        maxReach = max(maxReach, i + arr[i]);
        if (maxReach >= n - 1) {
            return true;
        }
    }
    return false;
}

int main() {
    cout << "Enter the size of the array: ";
    int n;
    cin >> n;
    int arr[n];
    cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
    }

    cout << (canJump(arr, n) ? "True" : "False") << endl;

    return 0;
}
```

```
Output                                                    Clear
Enter the size of the array: 3
Enter the elements of the array: 1 2 3
True


=== Code Execution Successful ===
```

8. Given an integer n, return the least number of perfect square numbers that sum to n.

```
#include <iostream>
#include <climits>
using namespace std;
```

```
// Function to find the minimum number of perfect squares that sum to n
int minSquares(int n) {
    int dp[n + 1];
    fill(dp, dp + n + 1, INT_MAX);
    dp[0] = 0;

    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j * j <= i; ++j) {
            dp[i] = min(dp[i], dp[i - j * j] + 1);
        }
    }

    return dp[n];
}

int main() {
    cout << "Enter the value of n: ";
    int n;
    cin >> n;

    cout << "The least number of perfect square numbers that sum to " << n << " is: " << minSquares(n)
<< endl;

    return 0;
}
```

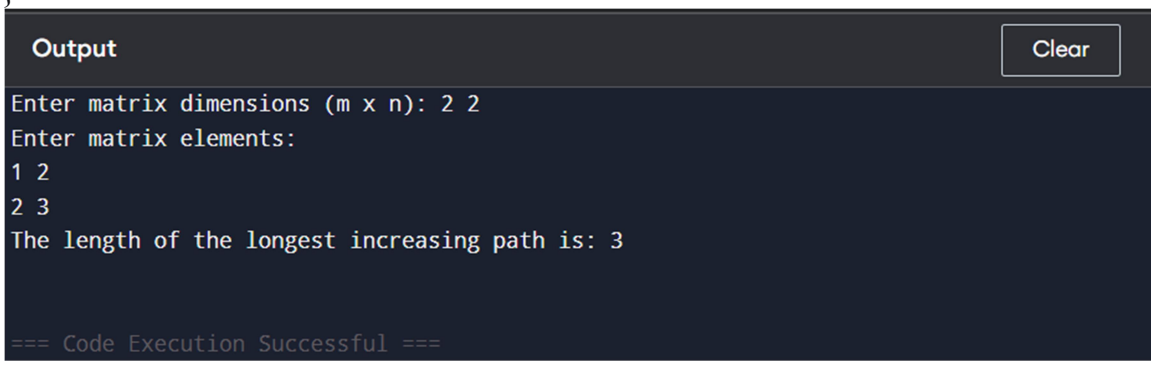


9. Given an m x n integers matrix, return the length of the longest increasing path in matrix. From each cell, you can either move in four directions: left, right, up, or down. You may not move diagonally or move outside the boundary (i.e., wrap-around is not allowed).

```
#include <iostream>
#include <algorithm>

using namespace std;

int dfs(int** matrix, int** memo, int i, int j, int m, int n) {
    if (memo[i][j] != -1)
        return memo[i][j];

    int maxLength = 1;
    int directions[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

    for (auto dir : directions) {
        int ni = i + dir[0], nj = j + dir[1];
```

```
        if (ni >= 0 && ni < m && nj >= 0 && nj < n && matrix[ni][nj] > matrix[i][j]) {
            maxLength = max(maxLength, 1 + dfs(matrix, memo, ni, nj, m, n));
        }
    }

    memo[i][j] = maxLength;
    return maxLength;
}

int longestIncreasingPath(int** matrix, int m, int n) {
    if (!m || !n) return 0;

    int** memo = new int*[m];
    for (int i = 0; i < m; ++i)
        memo[i] = new int[n];

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

    int maxLength = 0;
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            maxLength = max(maxLength, dfs(matrix, memo, i, j, m, n));
        }
    }

    for (int i = 0; i < m; ++i)
        delete[] memo[i];
    delete[] memo;

    return maxLength;
}

int main() {
    cout << "Enter matrix dimensions (m x n): ";
    int m, n;
    cin >> m >> n;

    int** matrix = new int*[m];
    cout << "Enter matrix elements:" << endl;
    for (int i = 0; i < m; ++i) {
        matrix[i] = new int[n];
        for (int j = 0; j < n; ++j) {
            cin >> matrix[i][j];
        }
    }

    cout << "The length of the longest increasing path is: " << longestIncreasingPath(matrix, m, n) <<
endl;
```

```
    for (int i = 0; i < m; ++i)
        delete[] matrix[i];
    delete[] matrix;

    return 0;
}
```



10. **For a string sequence, a string word is k-repeating if word concatenated k times is a substring of sequence. The word's maximum k-repeating value is the highest value k where word is k-repeating in sequence. If word is not a substring of sequence, word's maximum k-repeating value is 0. Given strings sequence and word, return the maximum k-repeating value of word in sequence.**

```
#include <iostream>
#include <string>
using namespace std;

bool isKRepeating(const string& sequence, const string& word, int k) {
    string concatenatedWord = "";
    for (int i = 0; i < k; ++i) {
        concatenatedWord += word;
    }
    return sequence.find(concatenatedWord) != string::npos;
}

int maxKRepeating(const string& sequence, const string& word) {
    int low = 0, high = sequence.size() / word.size() + 1, result = 0;

    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (isKRepeating(sequence, word, mid)) {
            result = mid;
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }

    return result;
}
```

```
int main() {
    cout << "Enter sequence: ";
    string sequence;
    cin >> sequence;

    cout << "Enter word: ";
    string word;
    cin >> word;

    cout << "Maximum k-repeating value of word in sequence: " << maxKRepeating(sequence, word) <<
endl;

    return 0;
}
```