CU
CHANDIGARH UNIVERSITY

DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

DOMAIN WINTER CAMP WORKSHEET

DAY-8 (27/12/2024)

Student Name :- Pratham Kapoor **University ID :- 22BCS10732**

Branch :- B.E. (C.S.E.) **Section/Group :- FL_ 603-A**

Problem-1 :- 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. **(Very Easy)**

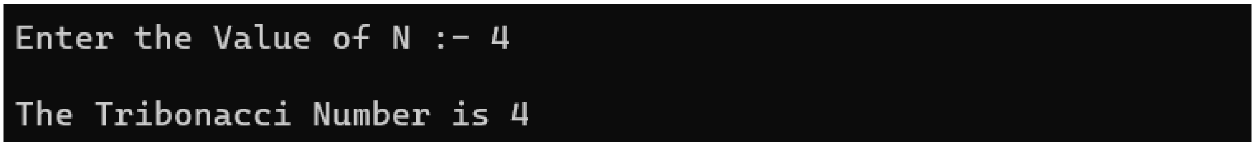
Source Code

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

int main() {
    int N;
    cout << "Enter the Value of N :- ";
    cin >> N;

    if(N == 0) {
        cout << "\nThe Tribonacci Number is 0";
    } else if(N == 1 || N == 2) {
        cout << "\nThe Tribonacci Number is 1";
    } else {
        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;
        }
        cout << "\nThe Tribonacci Number is " << Tn;
    }
    return 0;
}
```

Output



```
Enter the Value of N :- 4

The Tribonacci Number is 4
```

Problem-2 **:-** You are climbing a staircase. It takes n steps to reach the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top ? **(Easy)**

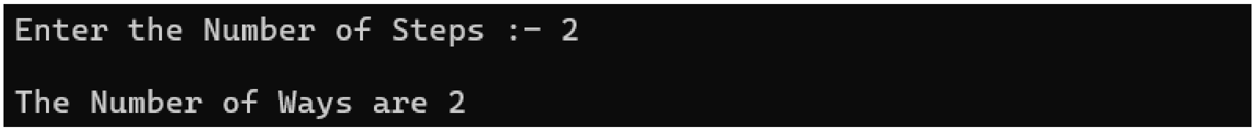
Source Code

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

int main() {
    int N;
    cout << "Enter the Number of Steps :- ";
    cin >> N;

    if(N == 1) {
        cout << "\nThe Number of Ways are 1";
    } else {
        int First = 1, Second = 2, Ways;
        for(int I = 3; I <= N; I++) {
            Ways = First + Second;
            First = Second;
            Second = Ways;
        }
        cout << "\nThe Number of Ways are " << (N == 2 ? 2 : Ways);
    }
    return 0;
}
```

Output



```
Enter the Number of Steps :- 2

The Number of Ways are 2
```

Problem-3 :- You are given string s. Return the longest palindromic substring in s. **(Medium)**

Source Code

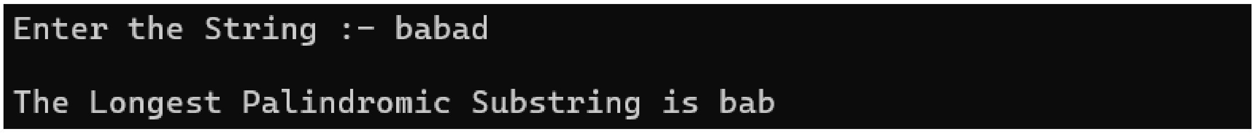
```
#include<iostream>
#include<string>
using namespace std;

string LongestPalindrome(string S) {
    int Start = 0, MaxLength = 1;
    int Length = S.length();

    for(int I = 0; I < Length; I++) {
        for(int J = I; J < Length; J++) {
            bool IsPalindrome = true;
            for(int K = 0; K < (J - I + 1) / 2; K++) {
                if(S[I + K] != S[J - K]) {
                    IsPalindrome = false;
                    break;
                }
            }
            if(IsPalindrome && (J - I + 1) > MaxLength) {
                Start = I;
                MaxLength = J - I + 1;
            }
        }
    }
    return S.substr(Start, MaxLength);
}

int main() {
    string S;
    cout << "Enter the String :- ";
    cin >> S;
    cout << "\nThe Longest Palindromic Substring is " <<
LongestPalindrome(S);
    return 0;
}
```

Output



```
Enter the String :- babad

The Longest Palindromic Substring is bab
```

Problem-4 :- Given a rows x cols binary matrix filled with 0's and 1's, find the largest rectangle containing only 1's and return its area. **(Hard)**

Source Code

```
#include<iostream>
#include<vector>
```

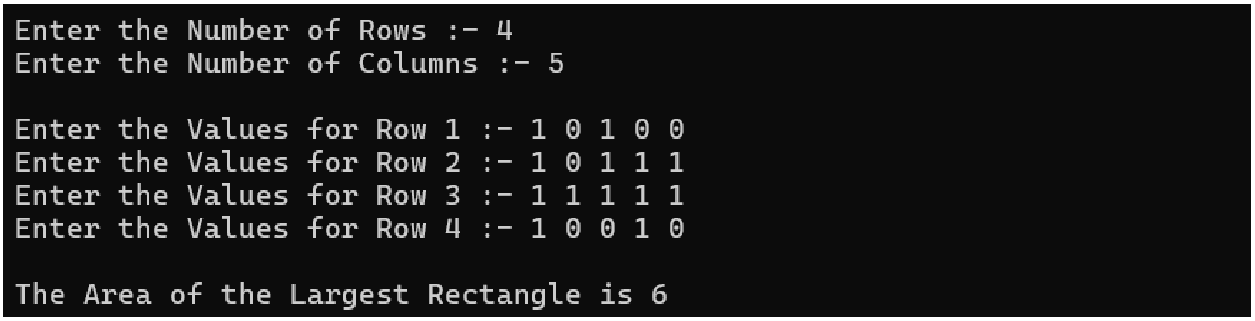
```
#include<stack>
using namespace std;

int LargestRectangleArea(vector<int>& Heights)
{
    stack<int> St;
    int MaxArea = 0;
    Heights.push_back(0);
    for(int I = 0; I < Heights.size(); I++)
   {
        while(!St.empty() && Heights[St.top()] > Heights[I]) {
            int Height = Heights[St.top()];
            St.pop();
            int Width = St.empty() ? I : I - St.top() - 1;
            MaxArea = max(MaxArea, Height * Width);
        }
        St.push(I);
    }
    return MaxArea;
}

int MaximalRectangle(vector<vector<char>>& Matrix)
{
    if(Matrix.empty()) return 0;
    int Rows = Matrix.size(), Cols = Matrix[0].size(), MaxArea = 0;
    vector<int> Heights(Cols, 0);

    for(int I = 0; I < Rows; I++)
   {
        for(int J = 0; J < Cols; J++)
       {
            Heights[J] = Matrix[I][J] == '1' ? Heights[J] + 1 : 0;
        }
        MaxArea = max(MaxArea, LargestRectangleArea(Heights));
    }
    return MaxArea;
}
int main() {
    int Rows;
    cout << "Enter the Number of Rows :- ";
    cin >> Rows;
    int Cols;
    cout << "Enter the Number of Columns :- ";
    cin >> Cols;
    cout << endl;
    vector<vector<char>> Matrix(Rows, vector<char>(Cols));
    for(int I = 0; I < Rows; I++) {
        cout << "Enter the Values for Row " << I + 1 << " :- ";
        for(int J = 0; J < Cols; J++) {
            cin >> Matrix[I][J];
        }
    }
    cout << "\nThe Area of the Largest Rectangle is " << MaximalRectangle(Matrix);
    return 0;
}
```

Output



```
Enter the Number of Rows :- 4
Enter the Number of Columns :- 5

Enter the Values for Row 1 :- 1 0 1 0 0
Enter the Values for Row 2 :- 1 0 1 1 1
Enter the Values for Row 3 :- 1 1 1 1 1
Enter the Values for Row 4 :- 1 0 0 1 0

The Area of the Largest Rectangle is 6
```

Problem-5 :- You are given an n x n grid representing a field of cherries, each cell is one of three possible integers. 0 means the cell is empty, so you can pass through, 1 means the cell contains a cherry that you can pick up and pass through, or -1 means the cell contains a thorn that blocks your way. Return the maximum number of cherries you can collect by following the rules below :- Starting at the position (0, 0) and reaching (n - 1, n - 1) by moving right or down through valid path cells (cells with value 0 or 1). After reaching (n - 1, n - 1), returning to (0, 0) by moving left or up through valid path cells. When passing through a path cell containing a cherry, you pick it up, and the cell becomes an empty cell 0. If there is no valid path between (0, 0) and (n - 1, n - 1), then no cherries can be collected. **(Very Hard)**

Source Code

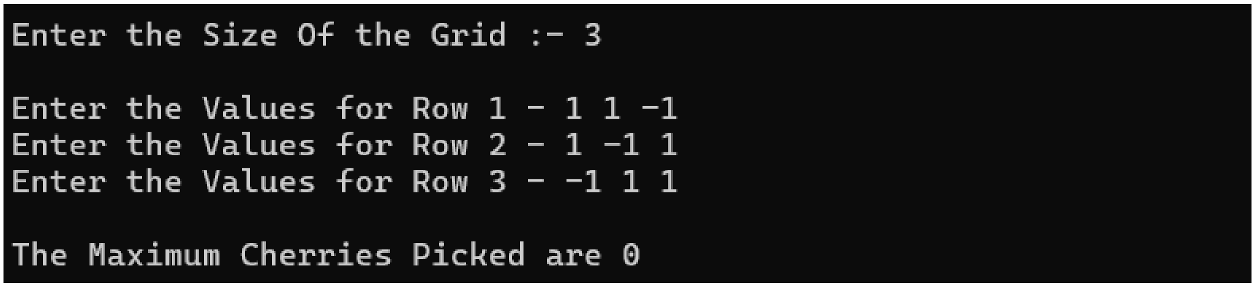
```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;

int CherryPickup(vector<vector<int>>&Grid){
    int N=Grid.size();

vector<vector<vector<int>>>Dp(N,vector<vector<int>>(N,vector<int>(N,-
1)));
    Dp[0][0][0]=Grid[0][0];
    for(int A=1;A<2*N-1;A++){
        for(int X1=min(N-1,A);X1>=0;X1--){
            for(int X2=min(N-1,A);X2>=0;X2--){
                int Y1=A-X1,Y2=A-X2;
                if(Y1>=N||Y2>=N||Grid[X1][Y1]==-1||Grid[X2][Y2]==-1){
                    Dp[X1][X2][A%N]=-1;
                    continue;
                }
                int Cherries=Grid[X1][Y1];
                if(X1!=X2)Cherries+=Grid[X2][Y2];
                int MaxPrev=-1;
                if(X1>0&&X2>0)MaxPrev=max(MaxPrev,Dp[X1-1][X2-1][(A-1)%N]);
                if(X1>0)MaxPrev=max(MaxPrev,Dp[X1-1][X2][(A-1)%N]);
                if(X2>0)MaxPrev=max(MaxPrev,Dp[X1][X2-1][(A-1)%N]);
                MaxPrev=max(MaxPrev,Dp[X1][X2][(A-1)%N]);
                if(MaxPrev>=0)Dp[X1][X2][A%N]=MaxPrev+Cherries;
                else Dp[X1][X2][A%N]=-1;
            }
        }
    }
    return max(0,Dp[N-1][N-1][(2*N-2)%N]);
}

int main(){
    int N;
    cout<<"Enter the Size Of the Grid :- ";
    cin>>N;
    cout << endl;
    vector<vector<int>>Grid(N,vector<int>(N));
    for(int I=0;I<N;I++){
         cout << "Enter the Values for Row " << I + 1 << " - ";
        for(int J=0;J<N;J++){
            cin>>Grid[I][J];
        }
    }
    int Result=CherryPickup(Grid);
    cout<<"\nThe Maximum Cherries Picked are "<<Result;
    return 0; }
```

Output



```
Enter the Size Of the Grid :- 3

Enter the Values for Row 1 - 1 1 -1
Enter the Values for Row 2 - 1 -1 1
Enter the Values for Row 3 - -1 1 1

The Maximum Cherries Picked are 0
```