DOMAIN WINTER WINNING CAMP

Student Name: Unnati Srivastava **UID: 22BCS13475**

Branch: BE-CSE **Section/Group: TPP_FL_603-A**

DAY 8:

QUES 1: N-th Tribonacci Number

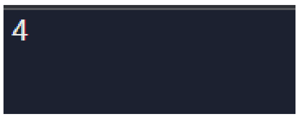
The Tribonacci sequence Tn is defined as follows:

T0 = 0, T1 = 1, T2 = 1, and Tn+3 = Tn + Tn+1 + Tn+2 for n >= 0.

Given n, return the value of Tn.

Solution:

```
#include <iostream>
#include <vector>
using namespace std;
class Solution {
public:
    int tribonacci(int n) {
        if (n == 0) return 0;
        if (n == 1 || n == 2) return 1;
        int a = 0, b = 1, c = 1, d;
        for (int i = 3; i <= n; ++i) {
            d = a + b + c;
            a = b;
            b = c;
            c = d;
        }
        return c;
    }
};
int main() {
    Solution solution;
    int n = 4;
    cout << solution.tribonacci(n) << endl;
    return 0;
}
```



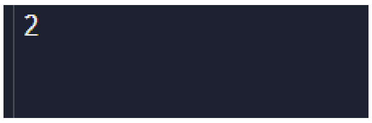
```
4
```

QUES 2: Climbing Stairs

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?

Solution:

```
#include <iostream>
using namespace std;
class Solution {
public:
    int climbStairs(int n) {
        if (n <= 2) return n;
        int a = 1, b = 2, c;
        for (int i = 3; i <= n; ++i) {
            c = a + b;
            a = b;
            b = c;
        }
        return b;
    }
};
int main() {
    Solution solution;
    int n = 2;
    cout << solution.climbStairs(n) << endl;
    return 0;
}
```



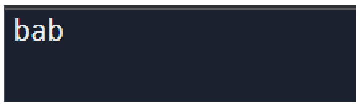
```
2
```

QUES 3: Longest Palindromic Substring

Given a string s, return the longest palindromic substring in s.

Solution:

```
#include <iostream>
#include <string>
using namespace std;
class Solution {
public:
    string longestPalindrome(string s) {
        int n = s.size(), start = 0, maxLen = 0;
        for (int i = 0; i < n; ++i) {
            auto expand = [&](int l, int r) {
                while (l >= 0 && r < n && s[l] == s[r]) --l, ++r;
                if (r - l - 1 > maxLen) {
                    start = l + 1;
                    maxLen = r - l - 1;
                }
            };
            expand(i, i);
            expand(i, i + 1);
        }
        return s.substr(start, maxLen);
    }
};
int main() {
    Solution solution;
    string s = "babad";
    cout << solution.longestPalindrome(s) << endl;
    return 0;
}
```



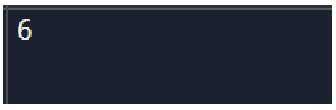
```
bab
```

QUES 4: Maximal Rectangle

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.

Solution:

```
#include <iostream>
#include <vector>
#include <stack>
using namespace std;
class Solution {
public:
    int maximalRectangle(vector<vector<char>>& matrix) {
        if (matrix.empty()) return 0;
        int m = matrix.size(), n = matrix[0].size(), maxArea = 0;
        vector<int> heights(n, 0);
        for (int i = 0; i < m; ++i) {
            for (int j = 0; j < n; ++j) {
                heights[j] = matrix[i][j] == '1' ? heights[j] + 1 : 0;
            }
            maxArea = max(maxArea, largestRectangleArea(heights));
        }
        return maxArea;
    }
private:
    int largestRectangleArea(vector<int>& heights) {
        stack<int> s;
        heights.push_back(0);
        int maxArea = 0;
        for (int i = 0; i < heights.size(); ++i) {
            while (!s.empty() && heights[s.top()] > heights[i]) {
                int h = heights[s.top()];
                s.pop();
                int w = s.empty() ? i : i - s.top() - 1;
                maxArea = max(maxArea, h * w);
            }
            s.push(i);
        }
        return maxArea;
    }
};
int main() {
    Solution solution;
    vector<vector<char>> matrix = {
        {'1', '0', '1', '0', '0'},
        {'1', '0', '1', '1', '1'},
        {'1', '1', '1', '1', '1'},
        {'1', '0', '0', '1', '0'}
    };
    cout << solution.maximalRectangle(matrix) << endl;
    return 0;
}
```



```
6
```

QUES 5: Cherry Pickup

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.

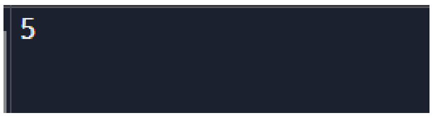
When passing through a path cell containing a cherry, you pick it up, and the cell becomes an empty cell 0.

If there is no valid path between (0, 0) and (n - 1, n - 1), then no cherries can be collected.

Solution:

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

```
public:
    int cherryPickup(vector<vector<int>>& grid) {
        int n = grid.size();
        vector<vector<vector<int>>> dp(n, vector<vector<int>>(n, vector<int>(n, -1)));
        return max(0, dfs(grid, dp, 0, 0, 0));
    }
private:
    int dfs(vector<vector<int>>& grid, vector<vector<vector<int>>>& dp, int x1, int y1, int
x2) {
        int y2 = x1 + y1 - x2;
        int n = grid.size();
        if (x1 >= n || y1 >= n || x2 >= n || y2 >= n || grid[x1][y1] == -1 || grid[x2][y2] == -1)
            return INT_MIN;
        if (x1 == n - 1 && y1 == n - 1)
            return grid[x1][y1];
        if (dp[x1][y1][x2] != -1)
            return dp[x1][y1][x2];
        int cherries = grid[x1][y1];
        if (x1 != x2)
            cherries += grid[x2][y2];
        int res = max({dfs(grid, dp, x1 + 1, y1, x2 + 1),
                       dfs(grid, dp, x1 + 1, y1, x2),
                       dfs(grid, dp, x1, y1 + 1, x2 + 1),
                       dfs(grid, dp, x1, y1 + 1, x2)});
        return dp[x1][y1][x2] = cherries + res;
    }
};
int main() {
    Solution solution;
    vector<vector<int>> grid = {{0, 1, -1}, {1, 0, -1}, {1, 1, 1}};
    cout << solution.cherryPickup(grid) << endl;
    return 0;
}
```



```
5
```