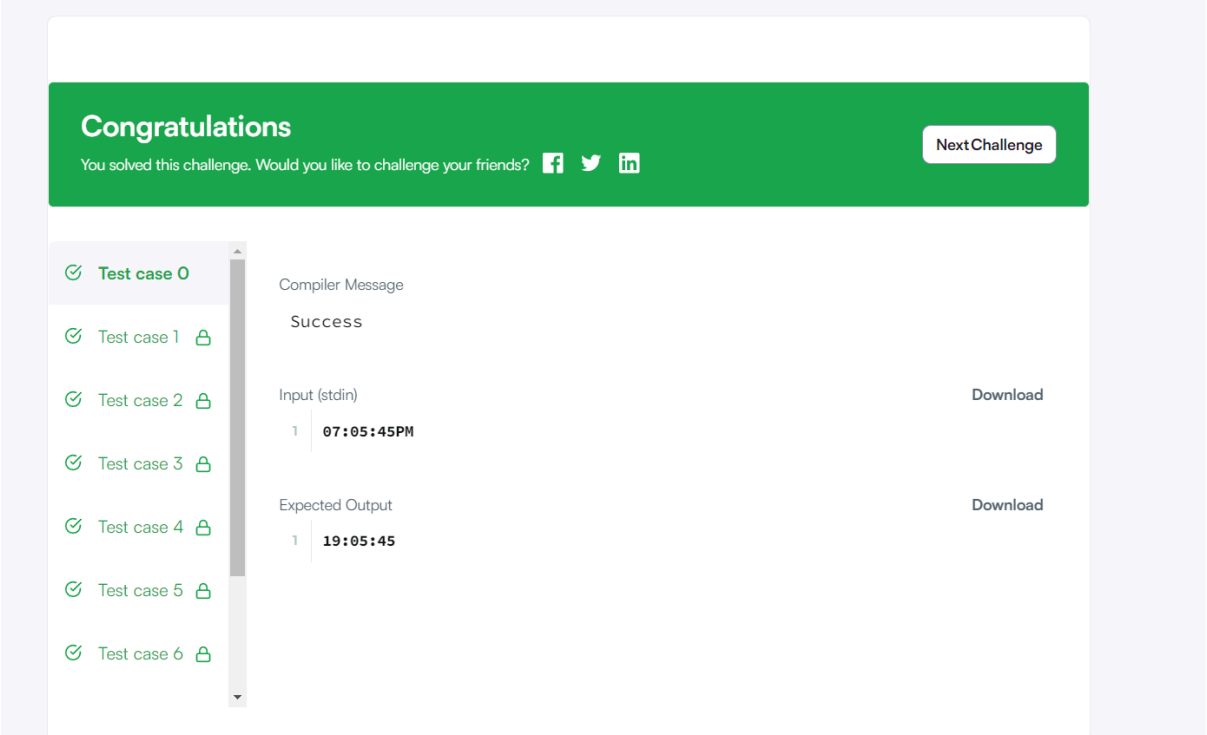
1. Time Conversion

Code:-

```
#include <bits/stdc++.h>

using namespace std;

/*
 * Complete the 'timeConversion' function below.
 *
 * The function is expected to return a STRING.
 * The function accepts STRING s as parameter.
 */

string timeConversion(string s) {

    int h = stoi(s.substr(0, 2)); // Extract hours
    string minute = s.substr(3, 2); // Extract minutes
    string second = s.substr(6, 2); // Extract seconds
    string am_pm = s.substr(8, 2); // Extract AM/PM

    if (am_pm == "PM" && h != 12) {
        h += 12;
    } else if (am_pm == "AM" && h == 12) {
        h = 0;
    }

    string hour;

    int unit = h % 10;
    h = h / 10;
    int decimals = h % 10;

    if (decimals == 0) {
        hour = "0" + to_string(unit);
    }

    else {
        hour = to_string(decimals) + to_string(unit);
    }

    string Time_24 = hour + ":" + minute + ":" + second;

    return Time_24;
}

int main()
```

```
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string s;
    getline(cin, s);

    string result = timeConversion(s);

    fout << result << "\n";

    fout.close();

    return 0;
}
```

Output :-

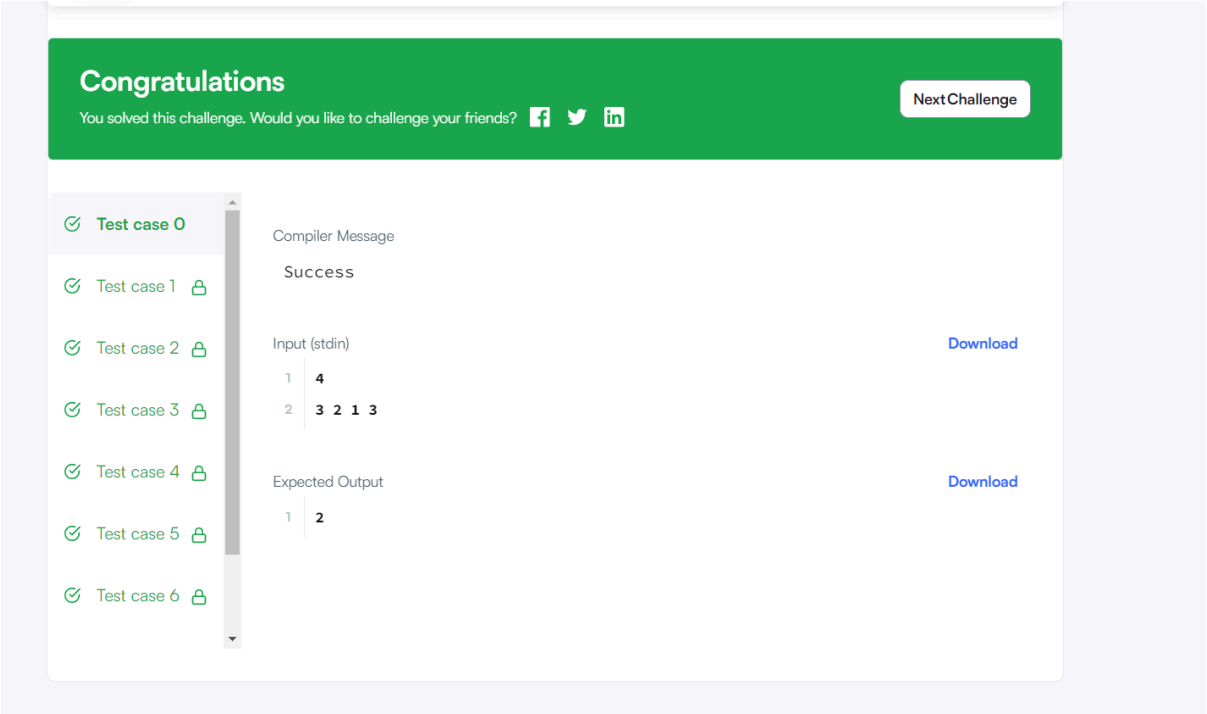


2. Birthday Cake Candles

```
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * Complete the 'birthdayCakeCandles' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts INTEGER_ARRAY candles as parameter.
 */

int birthdayCakeCandles(vector<int> candles) {
int max_number = candles[0];
int max_count = 0;
for(int i = 0; i < candles.size(); i++) {
if(candles[i] == max_number) {
max_count++;
} else if(candles[i] > max_number) {
max_number = candles[i];
max_count = 1;
}
}
return max_count;
}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string candles_count_temp;
    getline(cin, candles_count_temp);

    int candles_count = stoi(ltrim(rtrim(candles_count_temp)));

    string candles_temp_temp;
    getline(cin, candles_temp_temp);

    vector<string> candles_temp = split(rtrim(candles_temp_temp));

    vector<int> candles(candles_count);

    for (int i = 0; i < candles_count; i++) {
        int candles_item = stoi(candles_temp[i]);
```

```
        candles[i] = candles_item;
    }

    int result = birthdayCakeCandles(candles);

    fout << result << "\n";

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }

    tokens.push_back(str.substr(start));

    return tokens;
}
```

Output :[-



3. Compare the Triplets

Code:-

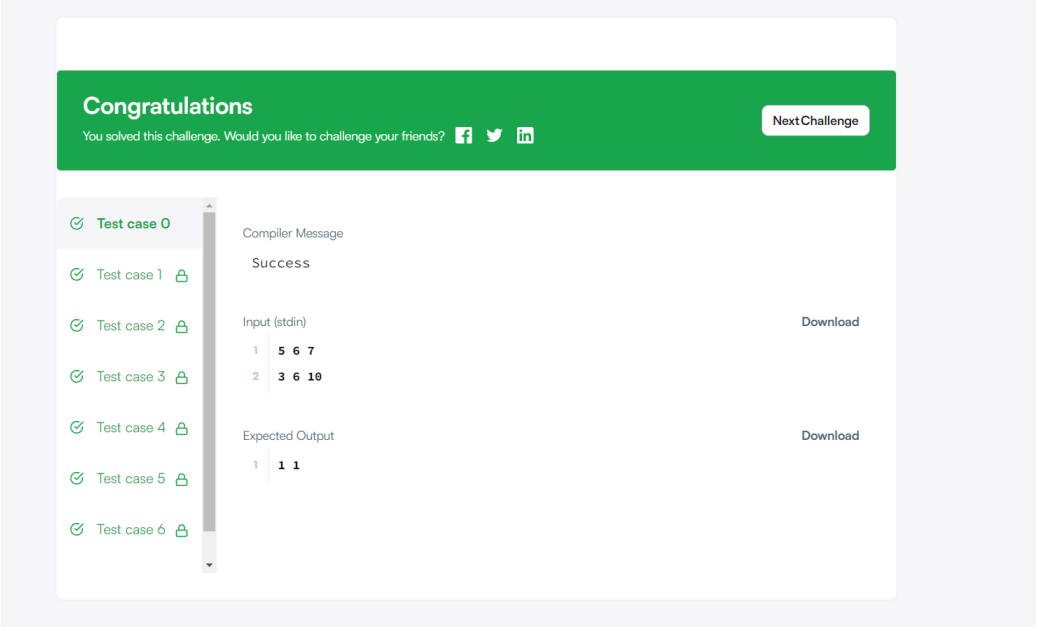
```
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * Complete the 'compareTriplets' function below.
 *
 * The function is expected to return an INTEGER_ARRAY.
 * The function accepts following parameters:
 *  1. INTEGER_ARRAY a
 *  2. INTEGER_ARRAY b
 */

vector<int> compareTriplets(vector<int> a, vector<int> b) {
```

```
    vector<int> Answer(2,0);

    for(int i = 0 ; i < a.size() ; i++) {
        if(a[i] > b[i]) {
            Answer[0]++;
        }

        else if(a[i] < b[i]) {
            Answer[1]++;
        }
    }

    return Answer;
}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string a_temp_temp;
    getline(cin, a_temp_temp);

    vector<string> a_temp = split(rtrim(a_temp_temp));

    vector<int> a(3);

    for (int i = 0; i < 3; i++) {
        int a_item = stoi(a_temp[i]);

        a[i] = a_item;
    }

    string b_temp_temp;
    getline(cin, b_temp_temp);

    vector<string> b_temp = split(rtrim(b_temp_temp));

    vector<int> b(3);

    for (int i = 0; i < 3; i++) {
        int b_item = stoi(b_temp[i]);

        b[i] = b_item;
    }

    vector<int> result = compareTriplets(a, b);

    for (size_t i = 0; i < result.size(); i++) {
```

```
        fout << result[i];

        if (i != result.size() - 1) {
            fout << " ";
        }
    }

    fout << "\n";

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }
```

```
    tokens.push_back(str.substr(start));

    return tokens;
}
```

Output



4. Diagonal Difference

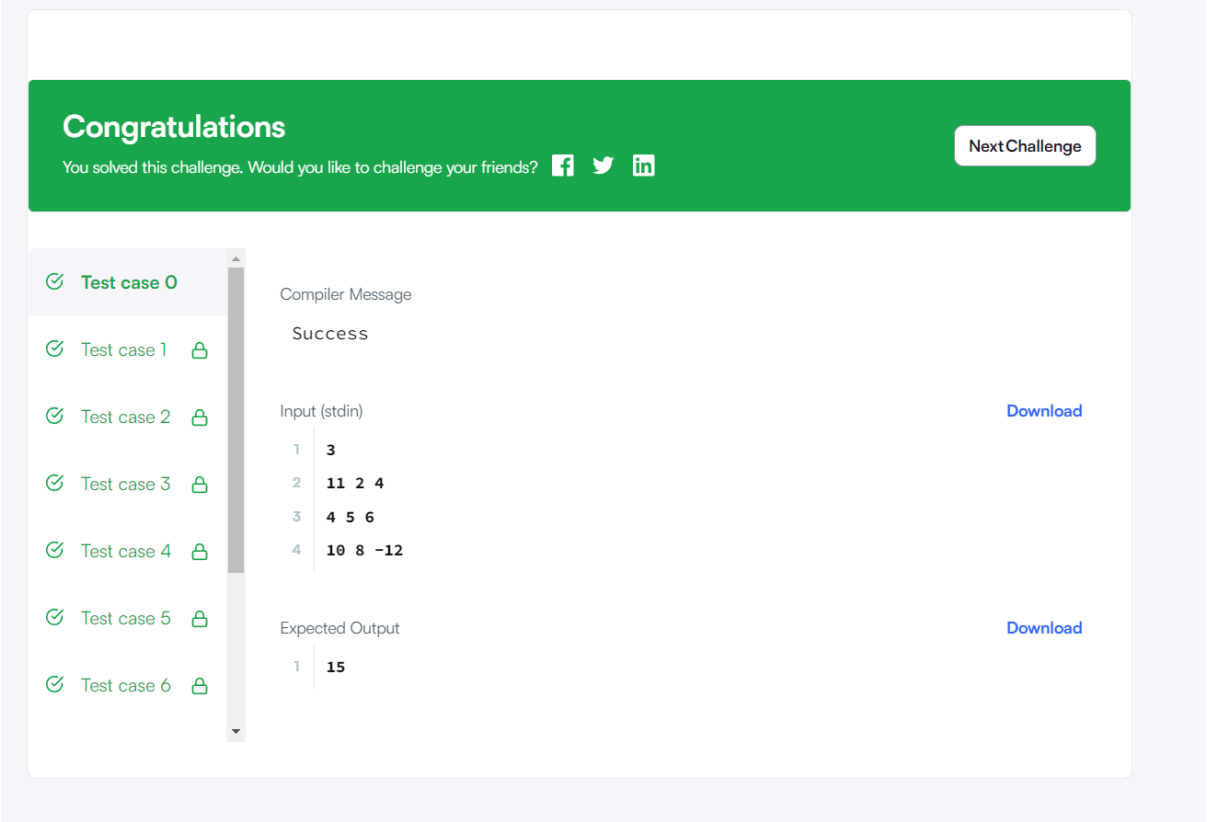
Code :-

```
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * Complete the 'diagonalDifference' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts 2D_INTEGER_ARRAY arr as parameter.
 */

int diagonalDifference(vector<vector<int>> arr) {
```

```
    int primary_diagonal = 0;
    int not_primary_diagonal = 0;
    int matrix_size = arr.size();
    for(int i = 0;i<matrix_size;i++){
        primary_diagonal+=arr[i][i];
        not_primary_diagonal+=arr[matrix_size-i-1][i];
    }
    return abs(primary_diagonal-not_primary_diagonal);

}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string n_temp;
    getline(cin, n_temp);

    int n = stoi(ltrim(rtrim(n_temp)));

    vector<vector<int>> arr(n);

    for (int i = 0; i < n; i++) {
        arr[i].resize(n);

        string arr_row_temp_temp;
        getline(cin, arr_row_temp_temp);

        vector<string> arr_row_temp = split(rtrim(arr_row_temp_temp));

        for (int j = 0; j < n; j++) {
            int arr_row_item = stoi(arr_row_temp[j]);

            arr[i][j] = arr_row_item;
        }
    }

    int result = diagonalDifference(arr);

    fout << result << "\n";

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);
```

```
    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }

    tokens.push_back(str.substr(start));

    return tokens;
}
```

Output :-



5. Mini-Max Sum

Code :-

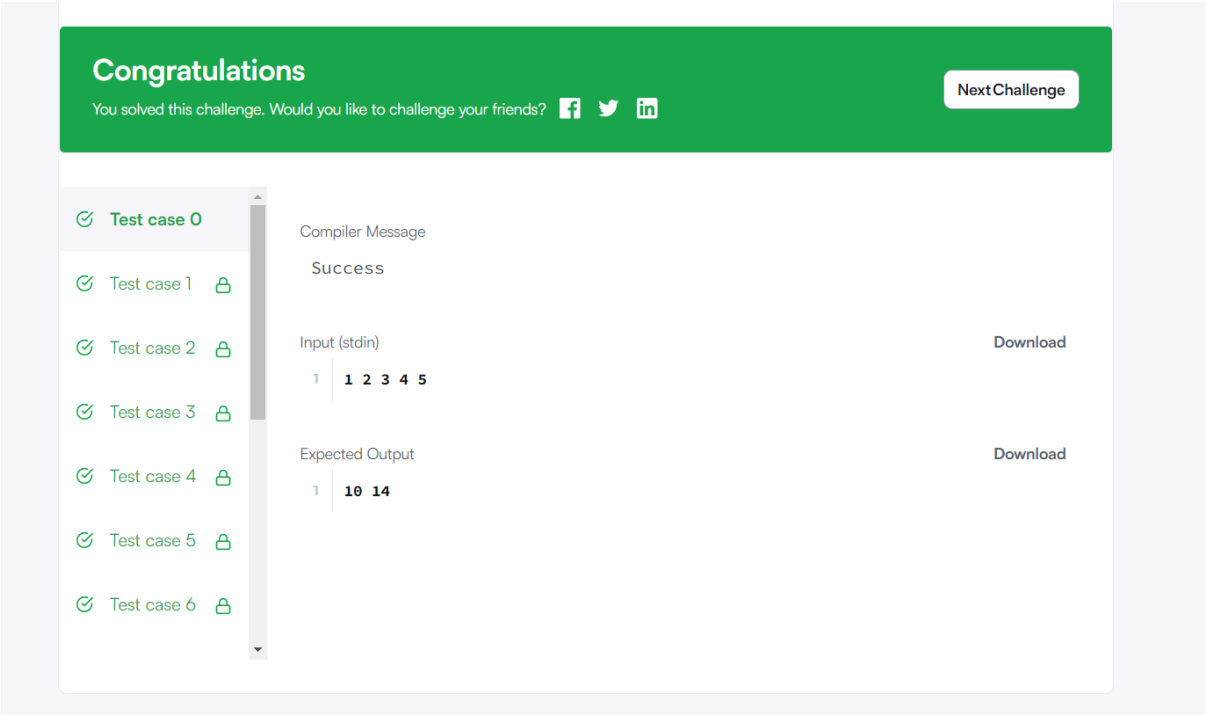
```
#include <bits/stdc++.h>


using namespace std;


string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);
```

```
/*
 * Complete the 'miniMaxSum' function below.
 *
 * The function accepts INTEGER_ARRAY arr as parameter.
 */


void miniMaxSum(vector<int> arr) {
    int_fast64_t min = 0 ;
    int_fast64_t max = 0 ;
    sort(arr.begin() , arr.end());

    for(int i = 0 ; i <= 3 ; i++) {
        min += arr[i] ;
        max += arr[i+1] ;
    }

    cout << min << " " << max << endl;
}


int main()
{


    string arr_temp_temp;
    getline(cin, arr_temp_temp);


    vector<string> arr_temp = split(rtrim(arr_temp_temp));


    vector<int> arr(5);


    for (int i = 0; i < 5; i++) {
        int arr_item = stoi(arr_temp[i]);


        arr[i] = arr_item;
    }


    miniMaxSum(arr);


    return 0;
}
```

```
string ltrim(const string &str) {
    string s(str);


    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );


    return s;
}


string rtrim(const string &str) {
    string s(str);


    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
        s.end()
    );


    return s;
}


vector<string> split(const string &str) {
    vector<string> tokens;


    string::size_type start = 0;
    string::size_type end = 0;


    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));


        start = end + 1;
    }


    tokens.push_back(str.substr(start));
```

```
    return tokens;
}
```

Output :-



6. Plus Minus

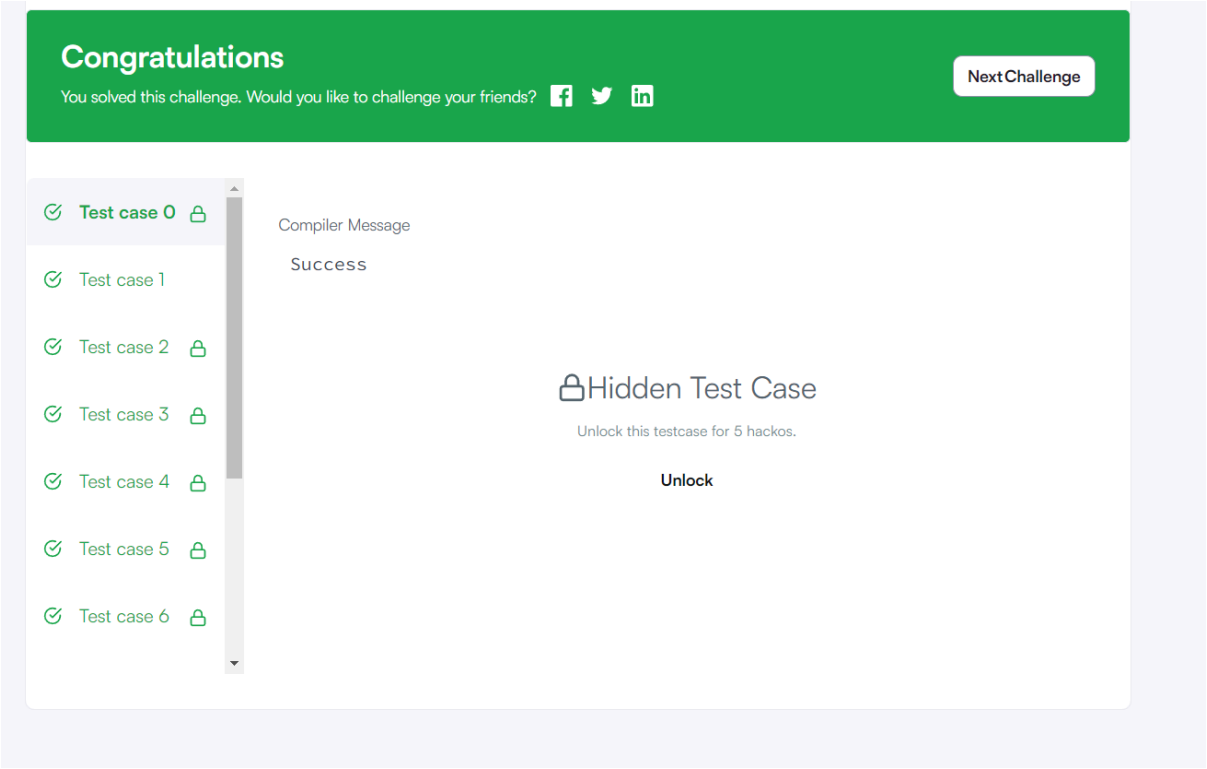
Code :-

```
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * Complete the 'plusMinus' function below.
```

```
 *
 * The function accepts INTEGER_ARRAY arr as parameter.
 */

void plusMinus(vector<int> arr) {
    double sp, sn, sz, pa = 1.0 / arr.size();
    sp = sn = sz = 0;
    for(int i = 0; i < arr.size(); i++){
        if(arr[i] > 0) sp+= pa;
        else if(arr[i]<0) sn+= pa;
        else sz+= pa;
    }
    cout << setprecision(6) << fixed;
    cout << sp <<endl;
    cout << sn <<endl;
    cout << sz <<endl;
}

int main()
{
    string n_temp;
    getline(cin, n_temp);

    int n = stoi(ltrim(rtrim(n_temp)));

    string arr_temp_temp;
    getline(cin, arr_temp_temp);

    vector<string> arr_temp = split(rtrim(arr_temp_temp));

    vector<int> arr(n);

    for (int i = 0; i < n; i++) {
        int arr_item = stoi(arr_temp[i]);

        arr[i] = arr_item;
    }

    plusMinus(arr);

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
```

```
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }

    tokens.push_back(str.substr(start));

    return tokens;
}
```

Output :-



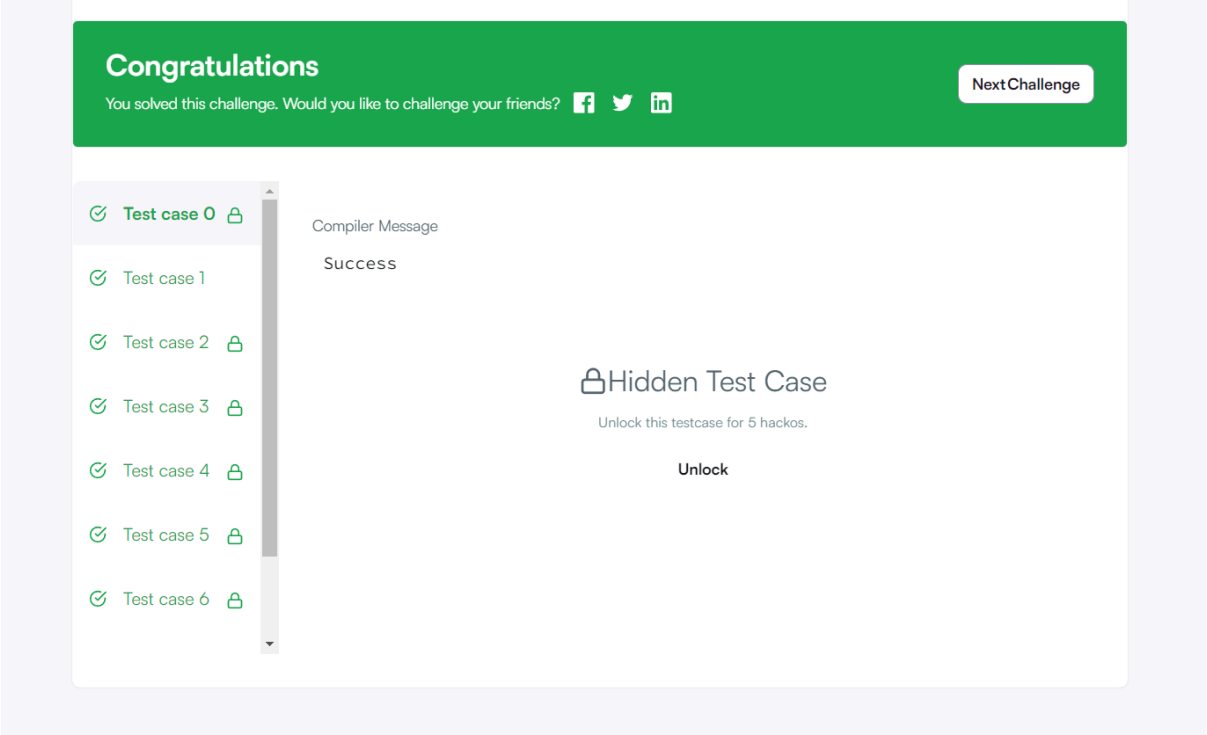
7. Staircase

```
#include <bits/stdc++.h>
```

```
using namespace std;

string ltrim(const string &);
string rtrim(const string &);

/*
 * Complete the 'staircase' function below.
 *
 * The function accepts INTEGER n as parameter.
 */

void staircase(int n) {

    for(int i = 1 ; i <= n; i++) {
        for(int j = 1 ; j <= n - i ; j++) {
            cout << " " ;
        }

        for(int j  =1 ; j <= i ; j++) {
            cout << "#" ;
        }

        cout << endl;
    }

}

int main()
{
    string n_temp;
    getline(cin, n_temp);

    int n = stoi(ltrim(rtrim(n_temp)));

    staircase(n);

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );
```

```
    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
        s.end()
    );

    return s;
}
```

Output :-



8. Migratory Bird

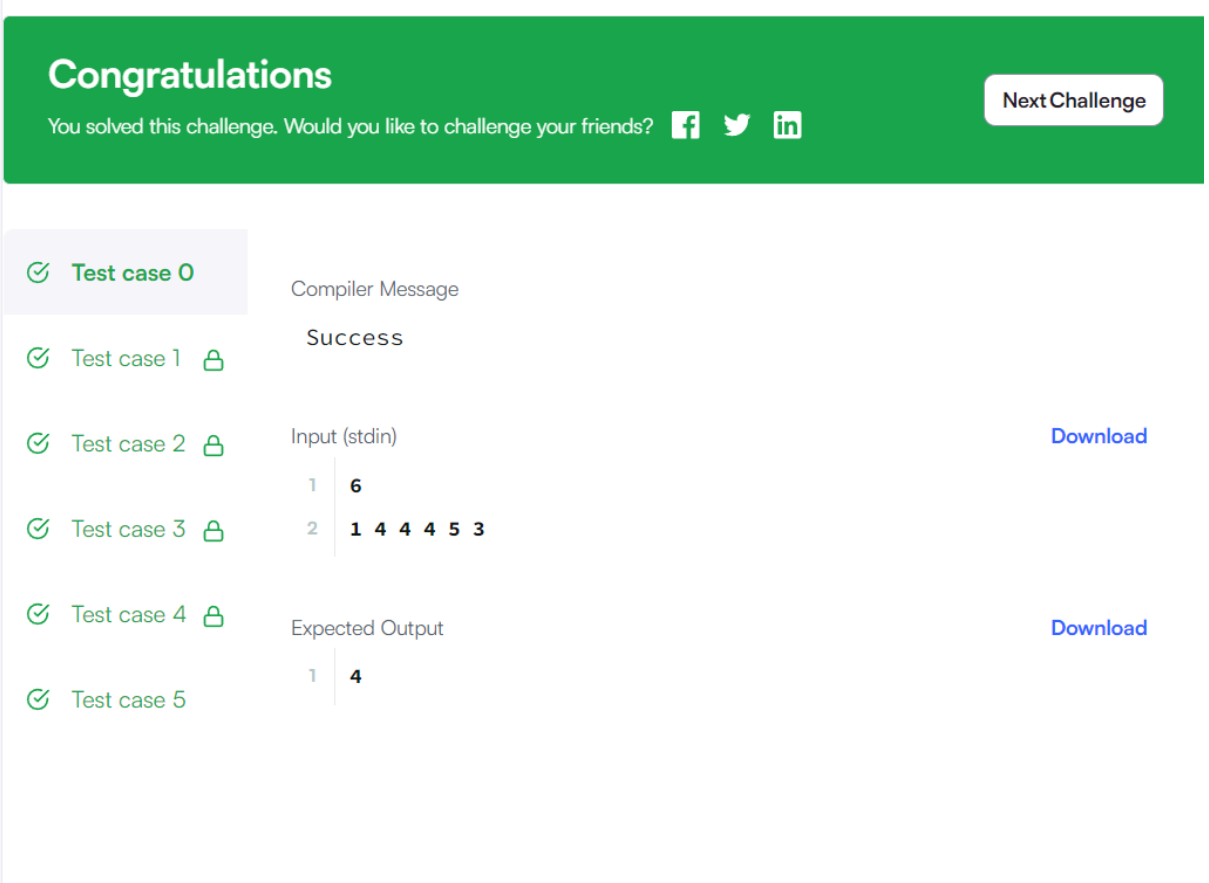
```
#include <bits/stdc++.h>
```

```
using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * Complete the 'migratoryBirds' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts INTEGER_ARRAY arr as parameter.
 */

int migratoryBirds(vector<int> arr) {
map<int, int> hello;
int max_element = 0;
for(int i:arr){
hello[i]++;
max_element = max(max_element,hello[i]);
}
int min_element = INT_MAX;
for(auto &current:hello){
if(current.second == max_element){
min_element =
min(min_element,current.first);
}
}
return min_element;
}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string arr_count_temp;
    getline(cin, arr_count_temp);

    int arr_count = stoi(ltrim(rtrim(arr_count_temp)));

    string arr_temp_temp;
    getline(cin, arr_temp_temp);

    vector<string> arr_temp = split(rtrim(arr_temp_temp));

    vector<int> arr(arr_count);

    for (int i = 0; i < arr_count; i++) {
```

```
        int arr_item = stoi(arr_temp[i]);

        arr[i] = arr_item;
    }

    int result = migratoryBirds(arr);

    fout << result << "\n";

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }
```

```
    tokens.push_back(str.substr(start));

    return tokens;
}
```

Output :-



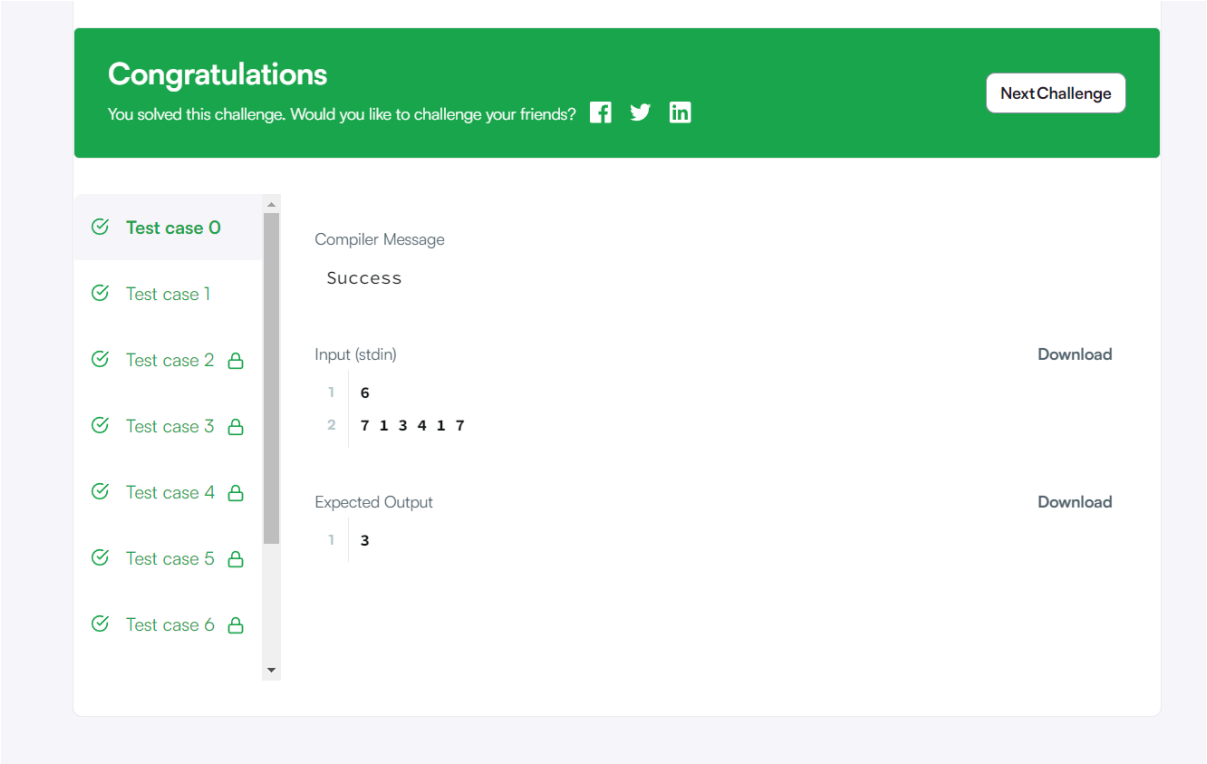
9. Minimum Distances

```
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * Complete the 'minimumDistances' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts INTEGER_ARRAY a as parameter.
 */

int minimumDistances(vector<int> a) {
    unordered_map<int,int> counter;
    int min_distance = 10000 ;

    for(int i = 0 ; i < a.size() ; i++) {
        if(counter[a[i]]) {
            min_distance = min(min_distance , abs(i + 1 - counter[a[i]]));
        }

        else {
            counter[a[i]] = i + 1;
        }
    }

    return min_distance == 10000 ? -1 : min_distance ;
}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string n_temp;
    getline(cin, n_temp);

    int n = stoi(ltrim(rtrim(n_temp)));

    string a_temp_temp;
    getline(cin, a_temp_temp);

    vector<string> a_temp = split(rtrim(a_temp_temp));

    vector<int> a(n);
```

```
    for (int i = 0; i < n; i++) {
        int a_item = stoi(a_temp[i]);

        a[i] = a_item;
    }

    int result = minimumDistances(a);

    fout << result << "\n";

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }
```

```
    tokens.push_back(str.substr(start));

    return tokens;
}
```

Output :-



10 Sequence Equation

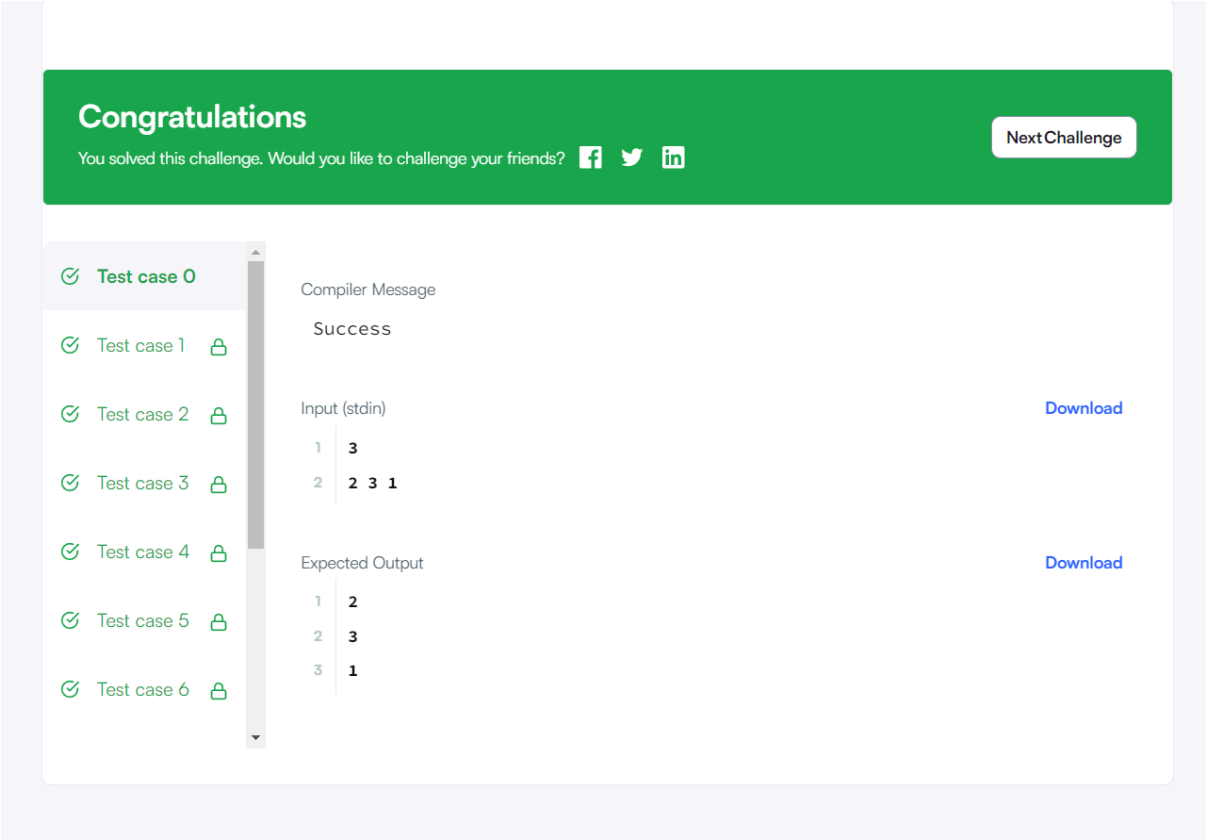
Code :-

```
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * Complete the 'permutationEquation' function below.
 *
 * The function is expected to return an INTEGER_ARRAY.
 * The function accepts INTEGER_ARRAY p as parameter.
 */

vector<int> permutationEquation(vector<int> p) {
    vector<int> indexes(p.size() + 1), result;
    for(int i = 1; i <= p.size(); i++) indexes[p[i-1]] = i;
    for(int i = 1; i <= p.size(); i++) result.push_back(indexes[indexes[i]]);
    return result;
}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string n_temp;
    getline(cin, n_temp);

    int n = stoi(ltrim(rtrim(n_temp)));

    string p_temp_temp;
    getline(cin, p_temp_temp);

    vector<string> p_temp = split(rtrim(p_temp_temp));

    vector<int> p(n);

    for (int i = 0; i < n; i++) {
        int p_item = stoi(p_temp[i]);

        p[i] = p_item;
    }

    vector<int> result = permutationEquation(p);
```

```
    for (size_t i = 0; i < result.size(); i++) {
        fout << result[i];

        if (i != result.size() - 1) {
            fout << "\n";
        }
    }

    fout << "\n";

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }
```

```
    tokens.push_back(str.substr(start));

    return tokens;
}
```

Output :-



11. **Picking Numbers**

Code :-

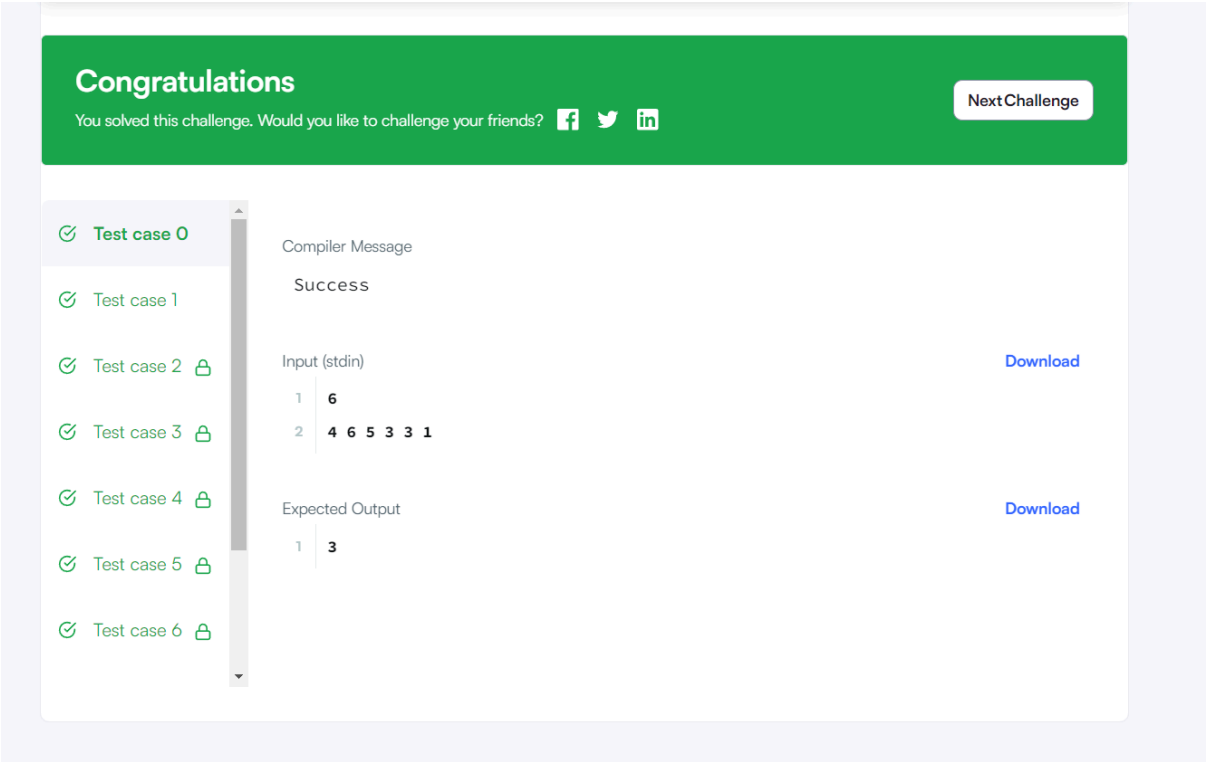
```
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * Complete the 'pickingNumbers' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts INTEGER_ARRAY a as parameter.
```

```
 */

int pickingNumbers(vector<int> a) {
    int count = 0 ;
    int max_count = INT_MIN;
    int negative_count = 0;
    int positive_count = 0;
    for(int j = 0;j<a.size()-1;j++){

    for(int i = j;i<a.size();i++){
        if(abs(a[j]-a[i])<2){

            if((a[j]-a[i])>0){
                positive_count++;
            }
            if((a[j]-a[i])<0){
                negative_count++;
            }
            if((a[j]-a[i])==0){
                positive_count++;
                negative_count++;
            }
        }
    }
    count = max(positive_count,negative_count);
    positive_count = 0;
    negative_count = 0;
    max_count = max(max_count,count);
    count = 0;
    }
    if(count == 0){
        count  = 1;
    }
    return max_count;
}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string n_temp;
    getline(cin, n_temp);

    int n = stoi(ltrim(rtrim(n_temp)));

    string a_temp_temp;
    getline(cin, a_temp_temp);

    vector<string> a_temp = split(rtrim(a_temp_temp));

    vector<int> a(n);
```

```
    for (int i = 0; i < n; i++) {
        int a_item = stoi(a_temp[i]);

        a[i] = a_item;
    }

    int result = pickingNumbers(a);

    fout << result << "\n";

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }

    tokens.push_back(str.substr(start));

    return tokens;
```

}

Output :-



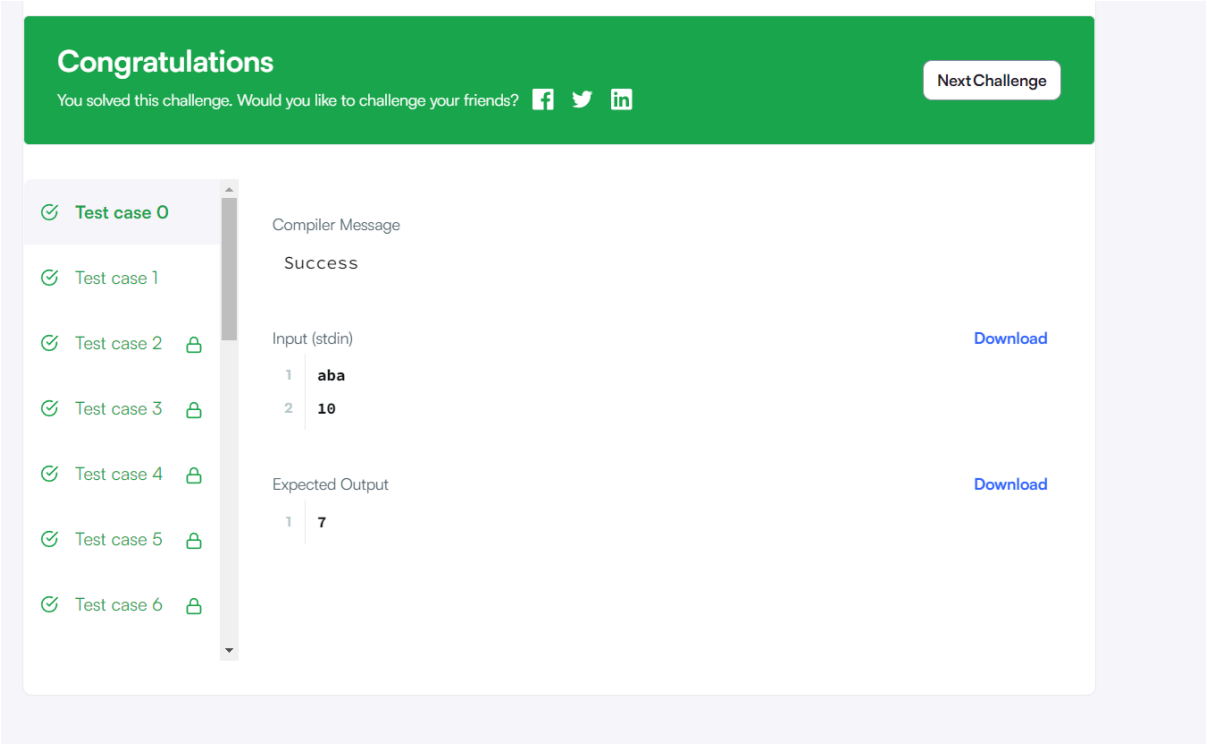
13 .Repeated String

```
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);

/*
 * Complete the 'repeatedString' function below.
 *
 * The function is expected to return a LONG_INTEGER.
 * The function accepts following parameters:
 *  1. STRING s
 *  2. LONG_INTEGER n
 */
```

```
long repeatedString(string s, long n) {
    long ans  = 0 ;
    int noofA = 0;

    for(char& chr : s) {
        if(chr == 'a') {
            noofA++;
        }
    }

    long full = n / s.size() ;
    long partial_r = n % s.size() ;
    ans = noofA * full ;

    for(int i = 0 ; i < partial_r ; i++) {
        if(s[i] == 'a') {
            ans++;
        }
    }

    return ans;

}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string s;
    getline(cin, s);

    string n_temp;
    getline(cin, n_temp);

    long n = stol(ltrim(rtrim(n_temp)));

    long result = repeatedString(s, n);

    fout << result << "\n";

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);
```

```
    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
        s.end()
    );

    return s;
}
```

Output :-



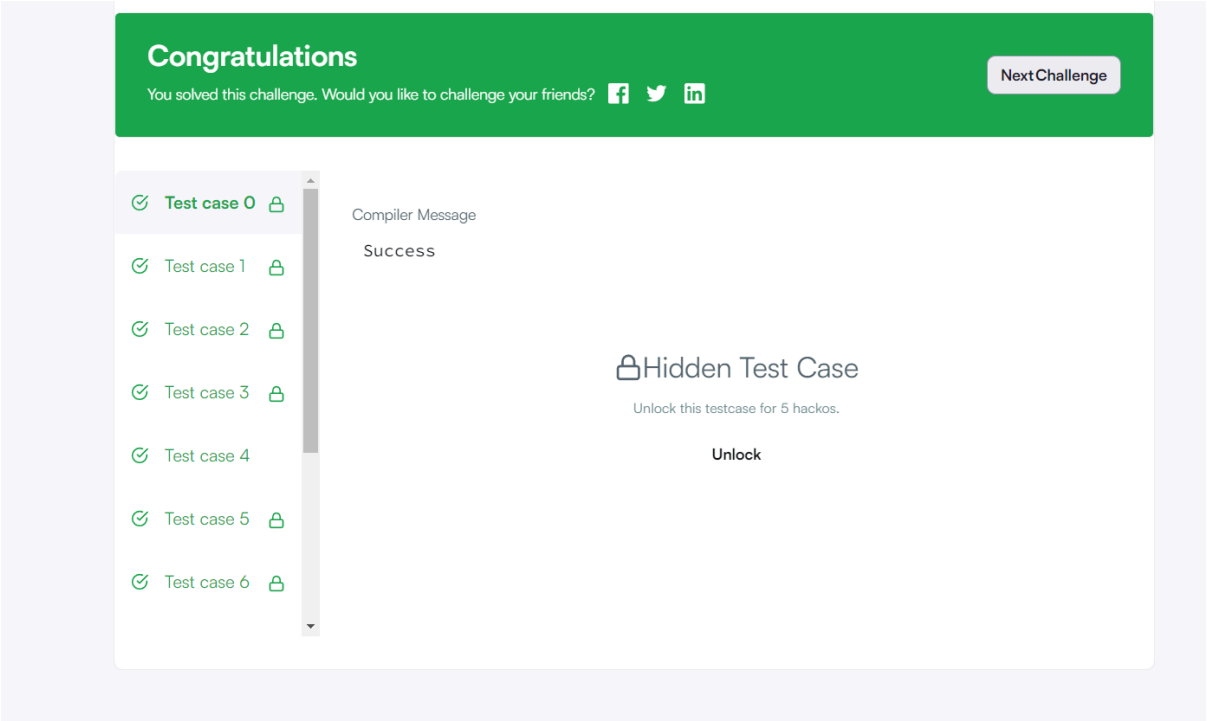
14. Save the Prisoner

Code :-

```
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * Complete the 'saveThePrisoner' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts following parameters:
 *  1. INTEGER n
 *  2. INTEGER m
 *  3. INTEGER s
 */

int saveThePrisoner(int n, int m, int s) {
    return (s+m-1) %n == 0 ? n : (s+m-1)%n;
}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string t_temp;
    getline(cin, t_temp);

    int t = stoi(ltrim(rtrim(t_temp)));

    for (int t_itr = 0; t_itr < t; t_itr++) {
        string first_multiple_input_temp;
        getline(cin, first_multiple_input_temp);

        vector<string> first_multiple_input = split(rtrim(first_multiple_input_temp));

        int n = stoi(first_multiple_input[0]);

        int m = stoi(first_multiple_input[1]);

        int s = stoi(first_multiple_input[2]);

        int result = saveThePrisoner(n, m, s);

        fout << result << "\n";
    }
```

```
    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }

    tokens.push_back(str.substr(start));

    return tokens;
}
```

Output :-



14. Sock Merchant

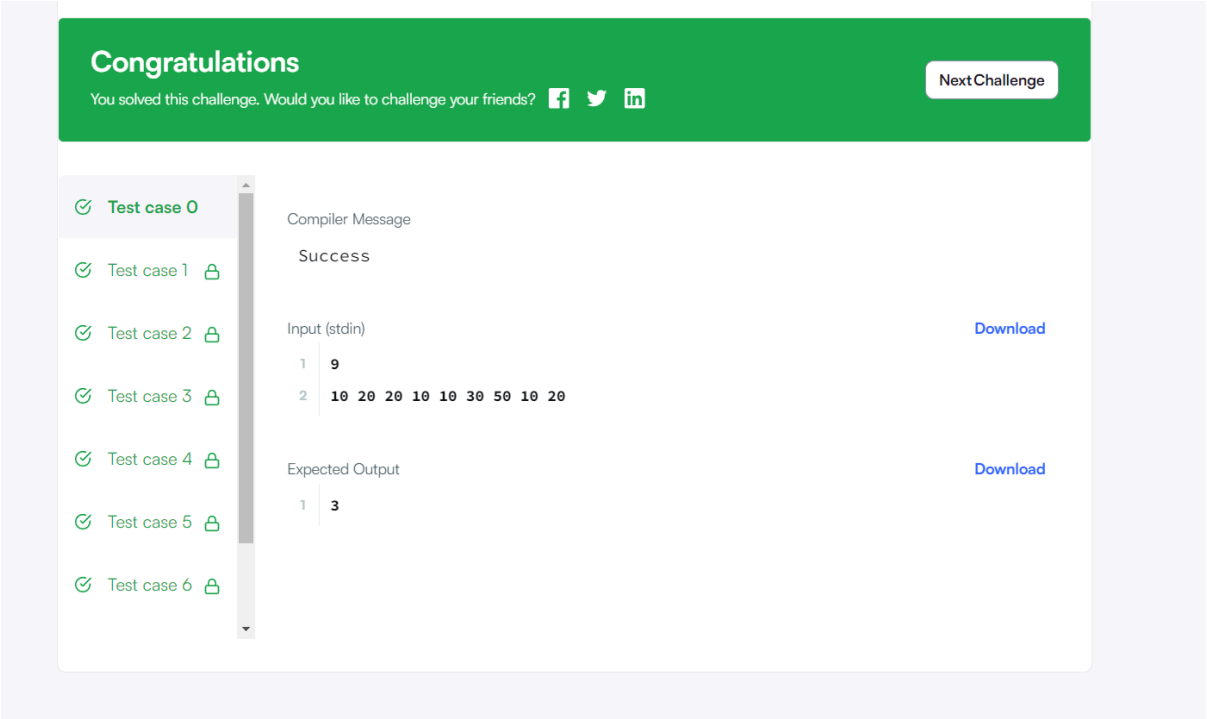
Code :-

```
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * Complete the 'sockMerchant' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts following parameters:
 *  1. INTEGER n
 *  2. INTEGER_ARRAY ar
 */

int sockMerchant(int n, vector<int> ar) {

    unordered_map<int, int> socks_count;
    for(int i:ar){
        socks_count[i]++;
    }
    int count = 0;
```

```
    for(auto &i:socks_count){
        count+= (i.second)/2;
    }
    return count;
}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string n_temp;
    getline(cin, n_temp);

    int n = stoi(ltrim(rtrim(n_temp)));

    string ar_temp_temp;
    getline(cin, ar_temp_temp);

    vector<string> ar_temp = split(rtrim(ar_temp_temp));

    vector<int> ar(n);

    for (int i = 0; i < n; i++) {
        int ar_item = stoi(ar_temp[i]);

        ar[i] = ar_item;
    }

    int result = sockMerchant(n, ar);

    fout << result << "\n";

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
}
```

```
string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }

    tokens.push_back(str.substr(start));

    return tokens;
}
```

Output :-



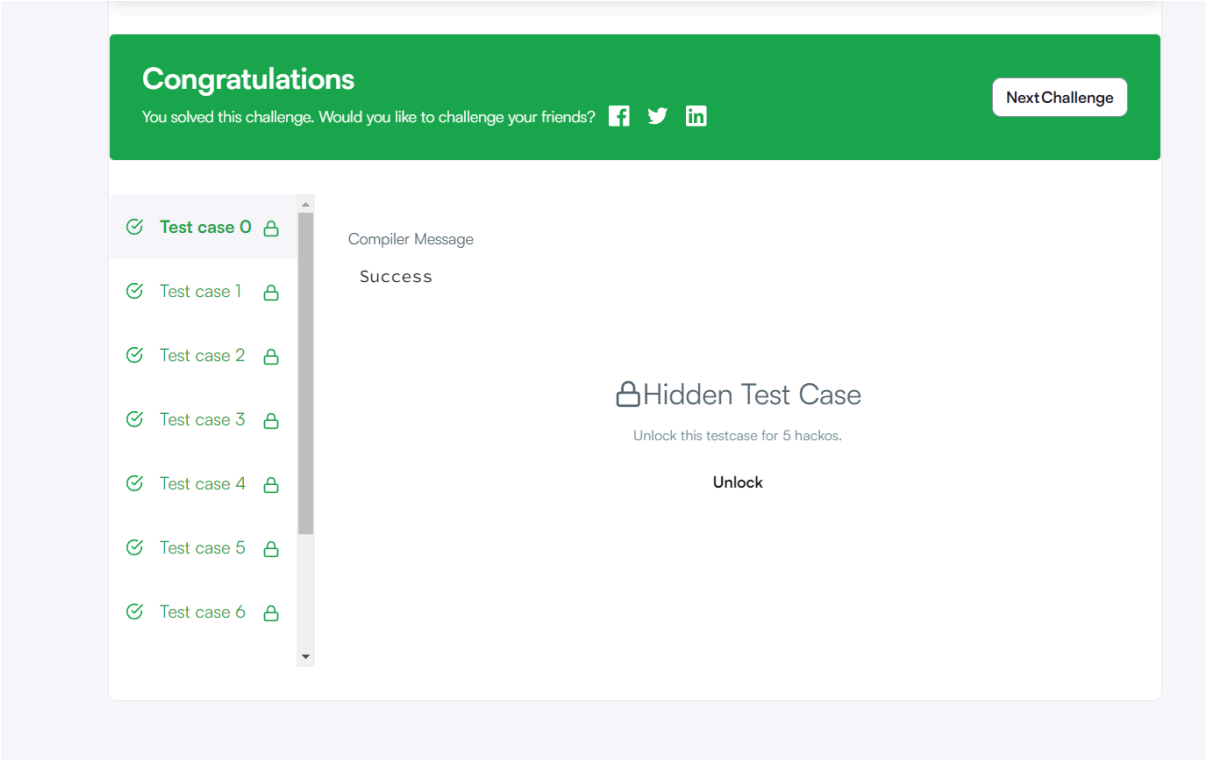
15. Sherlock and Squares

```
#include <bits/stdc++.h>

using namespace std;
```

```
string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * Complete the 'squares' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts following parameters:
 *  1. INTEGER a
 *  2. INTEGER b
 */

int squares(int a, int b) {

    int lower_bound = ceil(sqrt(double(a)));
    int upper_bound = floor(sqrt(double(b)));
    return upper_bound - lower_bound + 1 ;

}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string q_temp;
    getline(cin, q_temp);

    int q = stoi(ltrim(rtrim(q_temp)));

    for (int q_itr = 0; q_itr < q; q_itr++) {
        string first_multiple_input_temp;
        getline(cin, first_multiple_input_temp);

        vector<string> first_multiple_input = split(rtrim(first_multiple_input_temp));

        int a = stoi(first_multiple_input[0]);

        int b = stoi(first_multiple_input[1]);

        int result = squares(a, b);

        fout << result << "\n";
    }

    fout.close();
```

```
    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }

    tokens.push_back(str.substr(start));

    return tokens;
}
```

Output :-



17. Birthday Chocolate

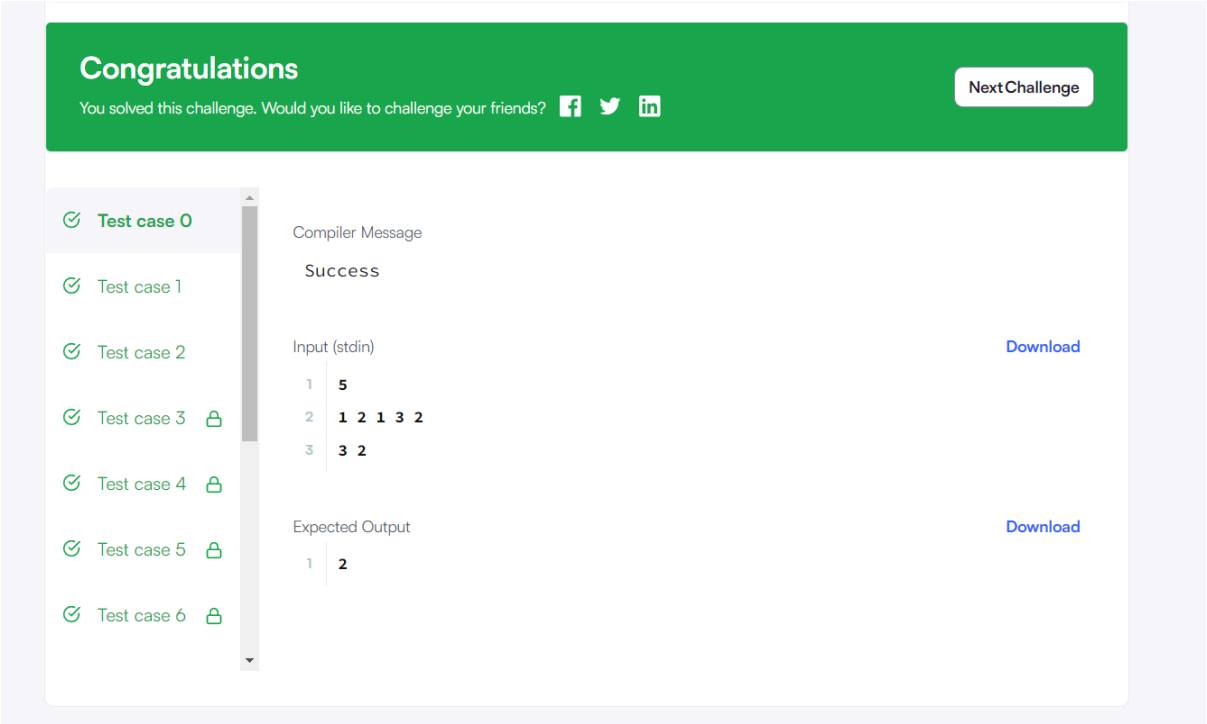
```
#include <bits/stdc++.h>
```

```
using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * Complete the 'birthday' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts following parameters:
 *  1. INTEGER_ARRAY s
 *  2. INTEGER d
 *  3. INTEGER m
 */

int sum(vector<int> a , int start , int end) {
    int sum = 0 ;
    for(int i = start ; i <= end ; i++) {
        sum += a[i] ;
    }

    return sum ;
}

int birthday(vector<int> s, int d, int m) {
    int start = 0;
    int end = m -1 ;
    int ans = 0 ;

    while(end < s.size()) {
        if(sum(s , start , end) ==  d) {
            ans++;
        }

        start++;
        end++;
    }

    return ans;
}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string n_temp;
```

```
    getline(cin, n_temp);

    int n = stoi(ltrim(rtrim(n_temp)));

    string s_temp_temp;
    getline(cin, s_temp_temp);

    vector<string> s_temp = split(rtrim(s_temp_temp));

    vector<int> s(n);

    for (int i = 0; i < n; i++) {
        int s_item = stoi(s_temp[i]);

        s[i] = s_item;
    }

    string first_multiple_input_temp;
    getline(cin, first_multiple_input_temp);

    vector<string> first_multiple_input = split(rtrim(first_multiple_input_temp));

    int d = stoi(first_multiple_input[0]);

    int m = stoi(first_multiple_input[1]);

    int result = birthday(s, d, m);

    fout << result << "\n";

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);
```

```
    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }

    tokens.push_back(str.substr(start));

    return tokens;
}
```

Output :-



18. Utopian Tree

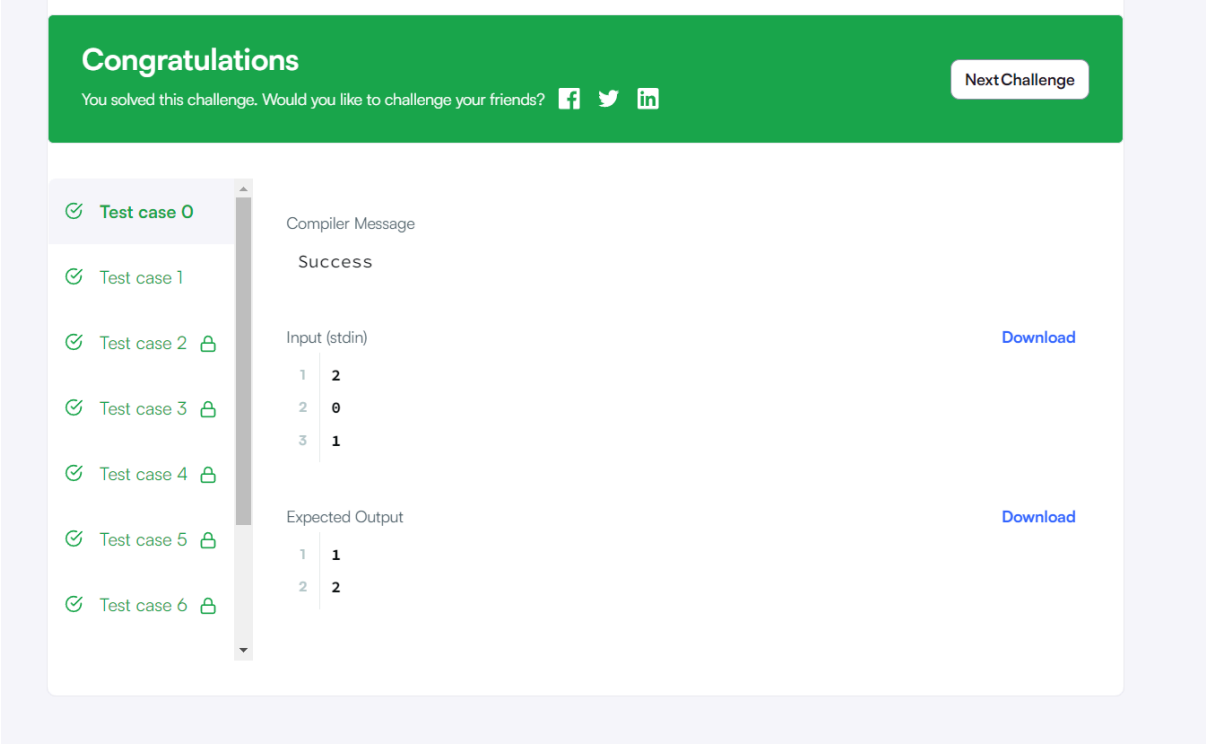
Code :-

```
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);

/*
 * Complete the 'utopianTree' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts INTEGER n as parameter.
 */

int utopianTree(int n) {

    int height = 1;
    if(n==0){
        return height;
    }
    for(int i = 1;i<=n;i++){
```

```
        if((i%2)==1){
            height*=2;
        }else{
            height+=1;
        }
    }
    return height;

}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string t_temp;
    getline(cin, t_temp);

    int t = stoi(ltrim(rtrim(t_temp)));

    for (int t_itr = 0; t_itr < t; t_itr++) {
        string n_temp;
        getline(cin, n_temp);

        int n = stoi(ltrim(rtrim(n_temp)));

        int result = utopianTree(n);

        fout << result << "\n";
    }

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);
```

```
    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
        s.end()
    );

    return s;
}
```

Output :-



19. The Hurdle Race

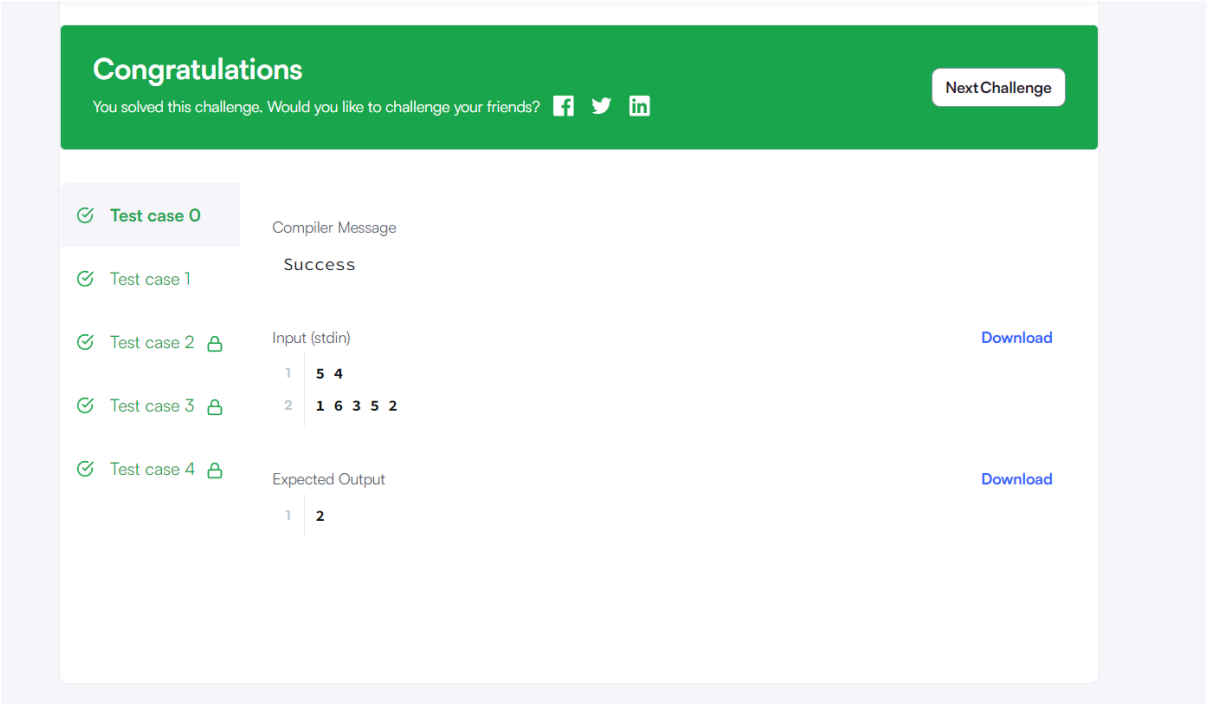
Code :-

```
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * Complete the 'hurdleRace' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts following parameters:
 *  1. INTEGER k
 *  2. INTEGER_ARRAY height
 */

int hurdleRace(int k, vector<int> a) {
    return *max_element(a.begin() , a.end()) > k ? *max_element(a.begin() , a.end()) - k : 0 ;
}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string first_multiple_input_temp;
    getline(cin, first_multiple_input_temp);

    vector<string> first_multiple_input = split(rtrim(first_multiple_input_temp));

    int n = stoi(first_multiple_input[0]);

    int k = stoi(first_multiple_input[1]);

    string height_temp_temp;
    getline(cin, height_temp_temp);

    vector<string> height_temp = split(rtrim(height_temp_temp));
```

```
    vector<int> height(n);

    for (int i = 0; i < n; i++) {
        int height_item = stoi(height_temp[i]);

        height[i] = height_item;
    }

    int result = hurdleRace(k, height);

    fout << result << "\n";

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));
```

```
        start = end + 1;
    }

    tokens.push_back(str.substr(start));

    return tokens;
}
```

Output :-



21. Angry Professor

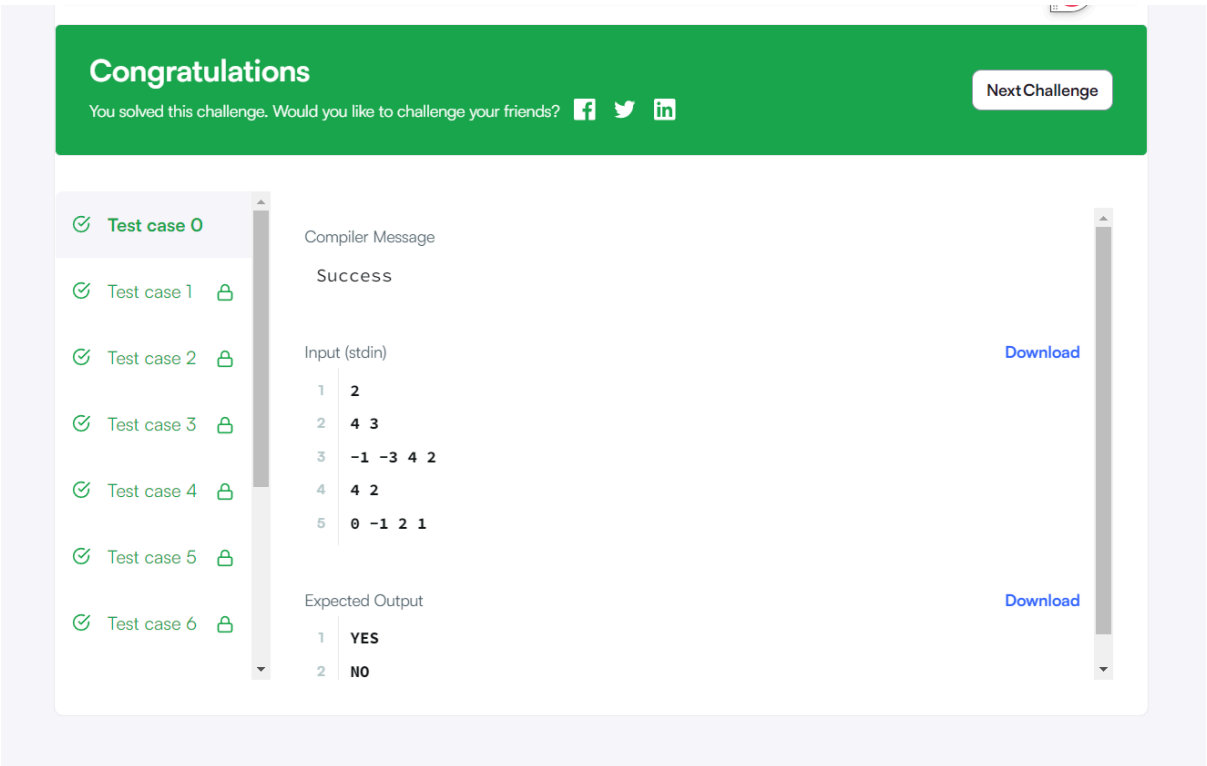
Code :-

```
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * Complete the 'angryProfessor' function below.
 *
 * The function is expected to return a STRING.
 * The function accepts following parameters:
 *  1. INTEGER k
 *  2. INTEGER_ARRAY a
 */

string angryProfessor(int k, vector<int> a) {
    int present = 0 ;

    for(int& stud : a) {
        if(stud <= 0) {
            present++;
        }
    }

    return present >= k ? "NO" : "YES" ;
}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string t_temp;
    getline(cin, t_temp);

    int t = stoi(ltrim(rtrim(t_temp)));

    for (int t_itr = 0; t_itr < t; t_itr++) {
```

```
        string first_multiple_input_temp;
        getline(cin, first_multiple_input_temp);

        vector<string> first_multiple_input = split(rtrim(first_multiple_input_temp));

        int n = stoi(first_multiple_input[0]);

        int k = stoi(first_multiple_input[1]);

        string a_temp_temp;
        getline(cin, a_temp_temp);

        vector<string> a_temp = split(rtrim(a_temp_temp));

        vector<int> a(n);

        for (int i = 0; i < n; i++) {
            int a_item = stoi(a_temp[i]);

            a[i] = a_item;
        }

        string result = angryProfessor(k, a);

        fout << result << "\n";
    }

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
```

```
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }

    tokens.push_back(str.substr(start));

    return tokens;
}
```

Output :-



23. Beautiful Days at the Movies

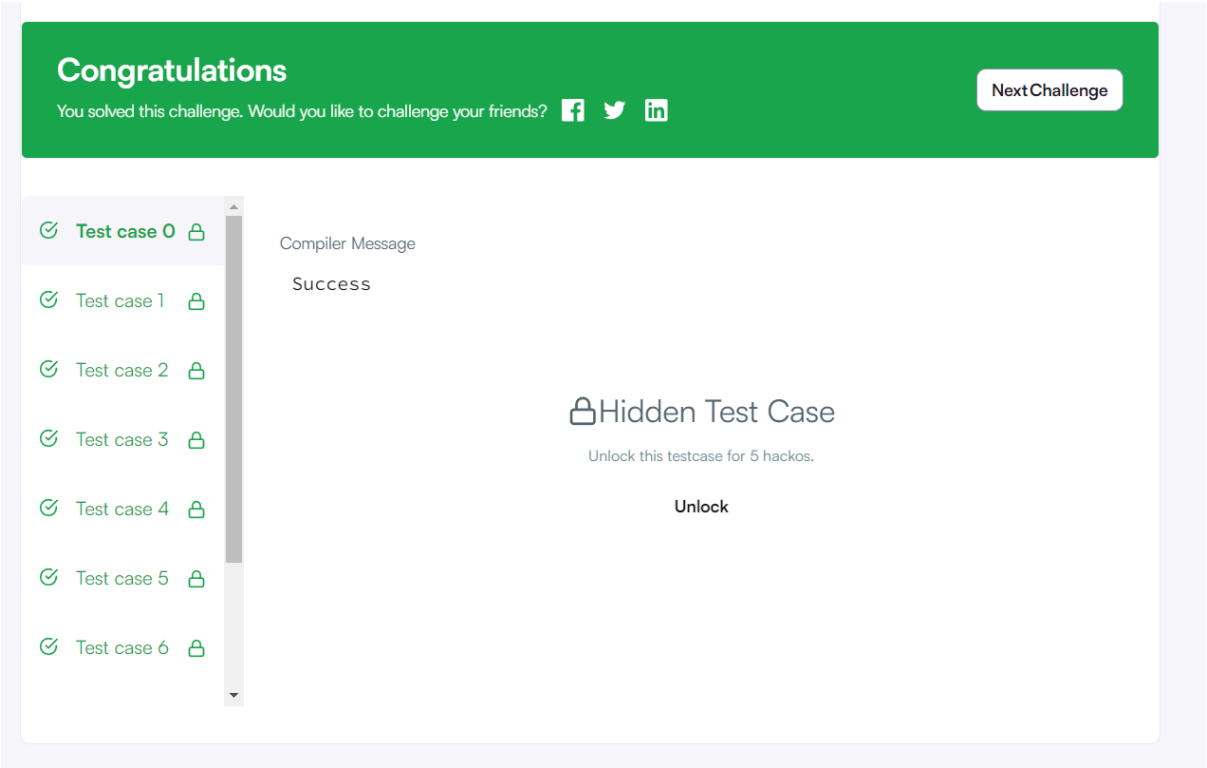
Code :-

```
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * Complete the 'beautifulDays' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts following parameters:
 *  1. INTEGER i
 *  2. INTEGER j
 *  3. INTEGER k
 */

int reverse(int num) {
    int reverse_int = 0 ;

    while(num > 0) {
        reverse_int = reverse_int*10 + num % 10 ;
        num /= 10 ;
    }

    return reverse_int ;

}

int beautifulDays(int i, int j, int k) {
    int beautifulDays = 0 ;
    for(int days = i ; days <= j ; days++) {
        int reverse_day = reverse(days) ;
        if(abs(reverse_day - days) % k == 0) {
            beautifulDays++;
        }
    }
```

```
    return beautifulDays;
}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string first_multiple_input_temp;
    getline(cin, first_multiple_input_temp);

    vector<string> first_multiple_input = split(rtrim(first_multiple_input_temp));

    int i = stoi(first_multiple_input[0]);

    int j = stoi(first_multiple_input[1]);

    int k = stoi(first_multiple_input[2]);

    int result = beautifulDays(i, j, k);

    fout << result << "\n";

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
        s.end()
    );

    return s;
}
```

```
vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }

    tokens.push_back(str.substr(start));

    return tokens;
}
```

Output :-



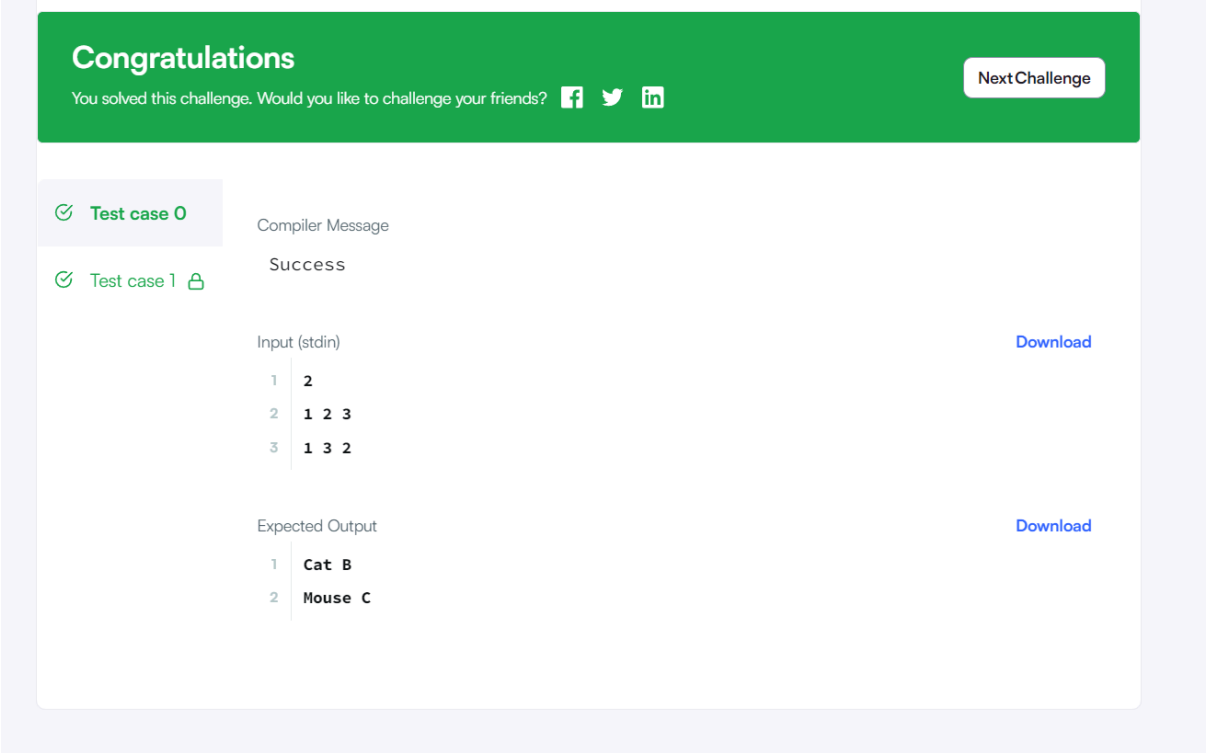
25. Bon Appétit

27. Cats and Mouse

```
#include <bits/stdc++.h>

using namespace std;

vector<string> split_string(string);

// Complete the catAndMouse function below.
string catAndMouse(int x, int y, int z) {
    if(abs(x-z) == abs(y-z)) {
        return "Mouse C" ;
    }

    else if(abs(x-z) > abs(y-z)) {
        return "Cat B";
    }

    else {
        return "Cat A" ;
    }

}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    int q;
    cin >> q;
    cin.ignore(numeric_limits<streamsize>::max(), '\n');

    for (int q_itr = 0; q_itr < q; q_itr++) {
        string xyz_temp;
        getline(cin, xyz_temp);

        vector<string> xyz = split_string(xyz_temp);

        int x = stoi(xyz[0]);

        int y = stoi(xyz[1]);

        int z = stoi(xyz[2]);
```

```
        string result = catAndMouse(x, y, z);

        fout << result << "\n";
    }

    fout.close();

    return 0;
}

vector<string> split_string(string input_string) {
    string::iterator new_end = unique(input_string.begin(), input_string.end(), [] (const char &x,
const char &y) {
        return x == y and x == ' ';
    });

    input_string.erase(new_end, input_string.end());

    while (input_string[input_string.length() - 1] == ' ') {
        input_string.pop_back();
    }

    vector<string> splits;
    char delimiter = ' ';

    size_t i = 0;
    size_t pos = input_string.find(delimiter);

    while (pos != string::npos) {
        splits.push_back(input_string.substr(i, pos - i));

        i = pos + 1;
        pos = input_string.find(delimiter, i);
    }

    splits.push_back(input_string.substr(i, min(pos, input_string.length()) - i + 1));

    return splits;
}
```

Output :-


Congratulations
You solved this challenge. Would you like to challenge your friends?
Next Challenge
Test case 0
Test case 1
Compiler Message
Success
Input (stdin)
Download
2
1 2 3
1 3 2
Expected Output
Download
Cat B
Mouse C

29. Circular Array Rotation

```
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * Complete the 'circularArrayRotation' function below.
 *
 * The function is expected to return an INTEGER_ARRAY.
 * The function accepts following parameters:
 *  1. INTEGER_ARRAY a
 *  2. INTEGER k
 *  3. INTEGER_ARRAY queries
 */

vector<int> circularArrayRotation(vector<int> a, int k, vector<int> queries) {
    vector<int> newArray(a.size()), result;
    for(int i = 0; i < a.size(); i++){
        newArray[(i + k) % a.size()] = a[i];
    }
    for(int i = 0; i < queries.size(); i++) result.push_back(newArray[queries[i]]);
    return result;
}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string first_multiple_input_temp;
    getline(cin, first_multiple_input_temp);

    vector<string> first_multiple_input = split(rtrim(first_multiple_input_temp));

    int n = stoi(first_multiple_input[0]);

    int k = stoi(first_multiple_input[1]);

    int q = stoi(first_multiple_input[2]);
```

```
    string a_temp_temp;
    getline(cin, a_temp_temp);

    vector<string> a_temp = split(rtrim(a_temp_temp));

    vector<int> a(n);

    for (int i = 0; i < n; i++) {
        int a_item = stoi(a_temp[i]);

        a[i] = a_item;
    }

    vector<int> queries(q);

    for (int i = 0; i < q; i++) {
        string queries_item_temp;
        getline(cin, queries_item_temp);

        int queries_item = stoi(ltrim(rtrim(queries_item_temp)));

        queries[i] = queries_item;
    }

    vector<int> result = circularArrayRotation(a, k, queries);

    for (size_t i = 0; i < result.size(); i++) {
        fout << result[i];

        if (i != result.size() - 1) {
            fout << "\n";
        }
    }

    fout << "\n";

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
```

```
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }

    tokens.push_back(str.substr(start));

    return tokens;
}
```

Output :-

Congratulations

You solved this challenge. Would you like to challenge your friends?

Next Challenge

- Test case 0
- Test case 1
- Test case 2
- Test case 3
- Test case 4
- Test case 5
- Test case 6

Compiler Message

Success

Hidden Test Case

Unlock this testcase for 5 hackos.

Unlock