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);

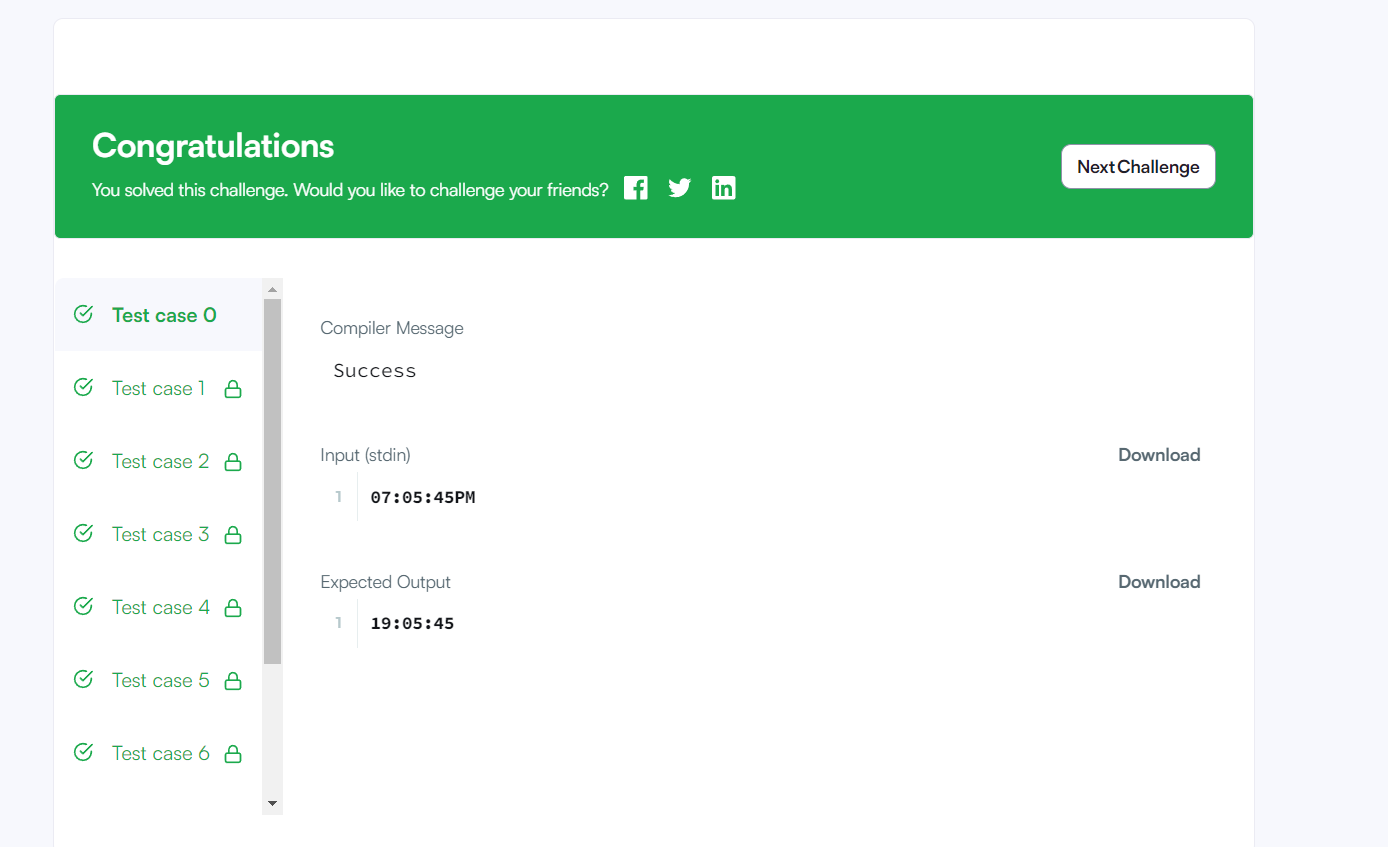
fout << result << "\n";

fout.close();

return 0;

}

Output :-



1. 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;

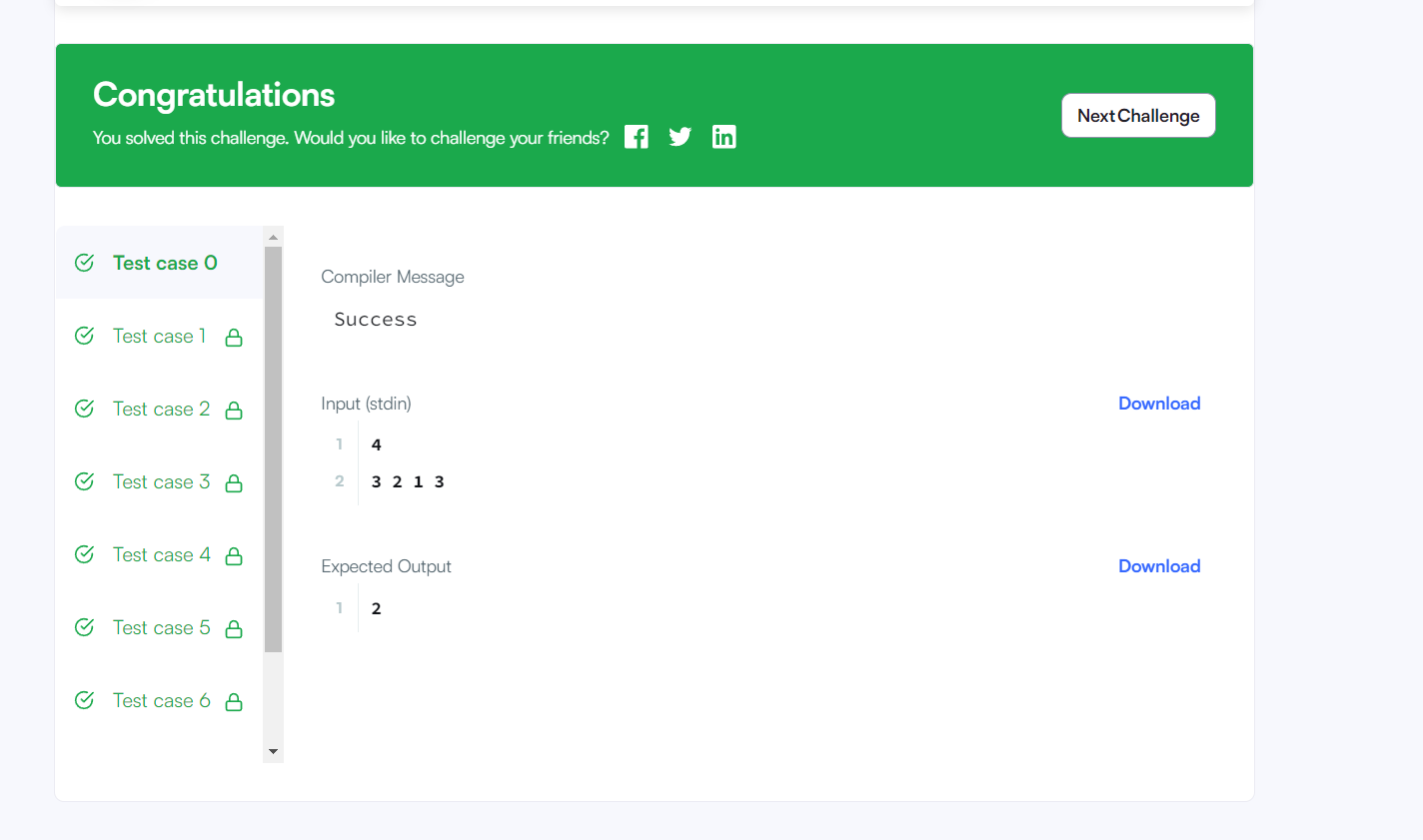
}

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;

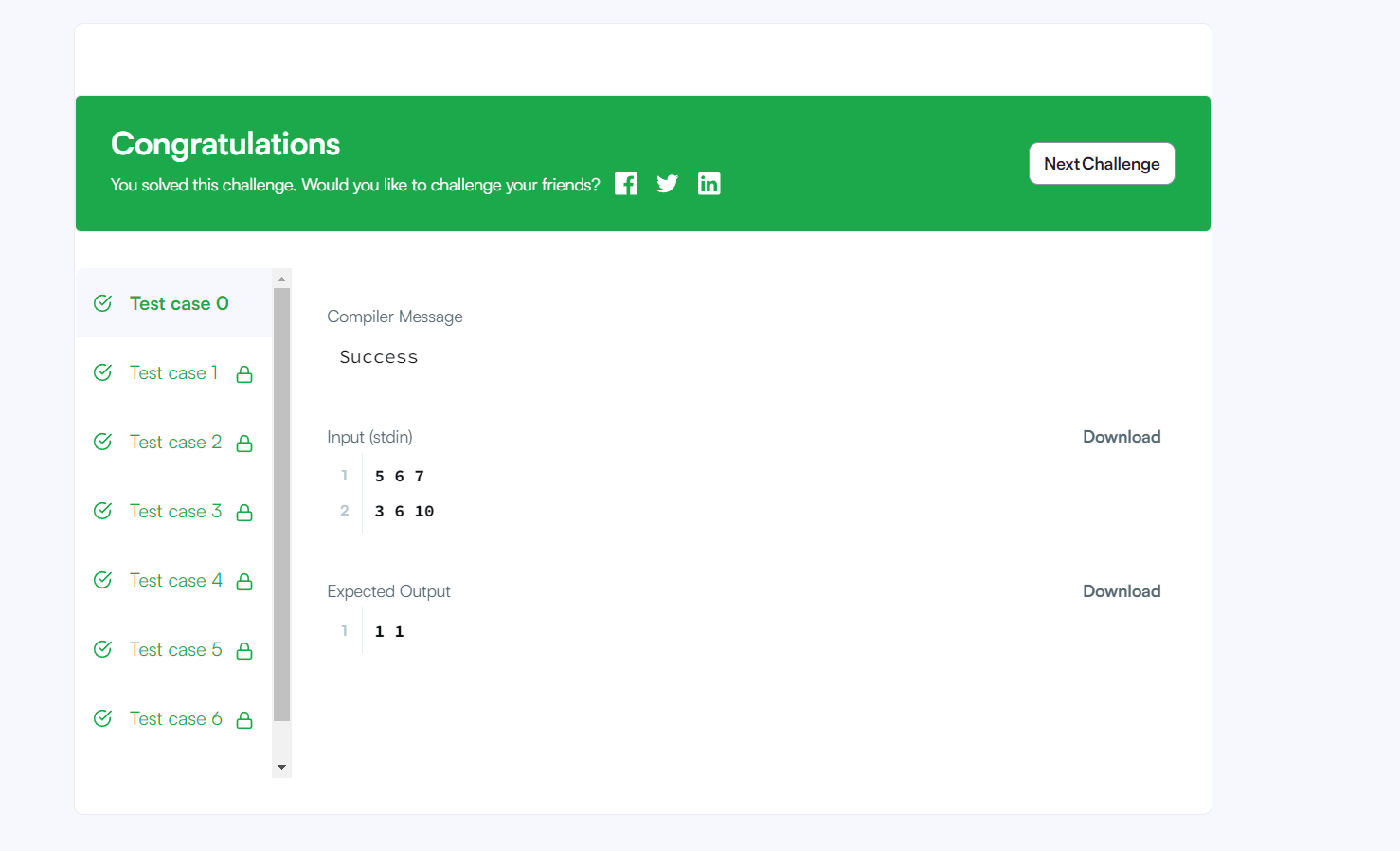
}

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;

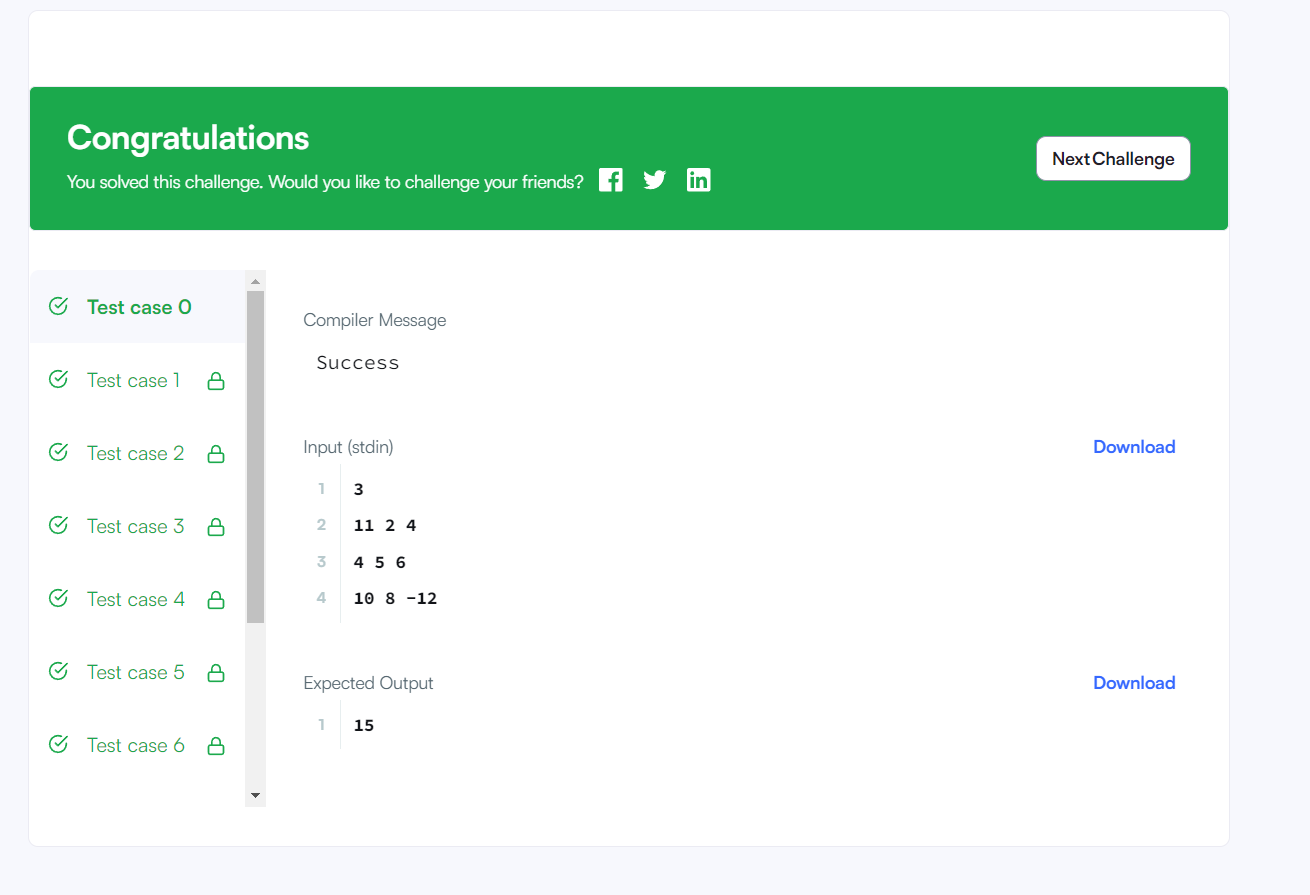
}

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;

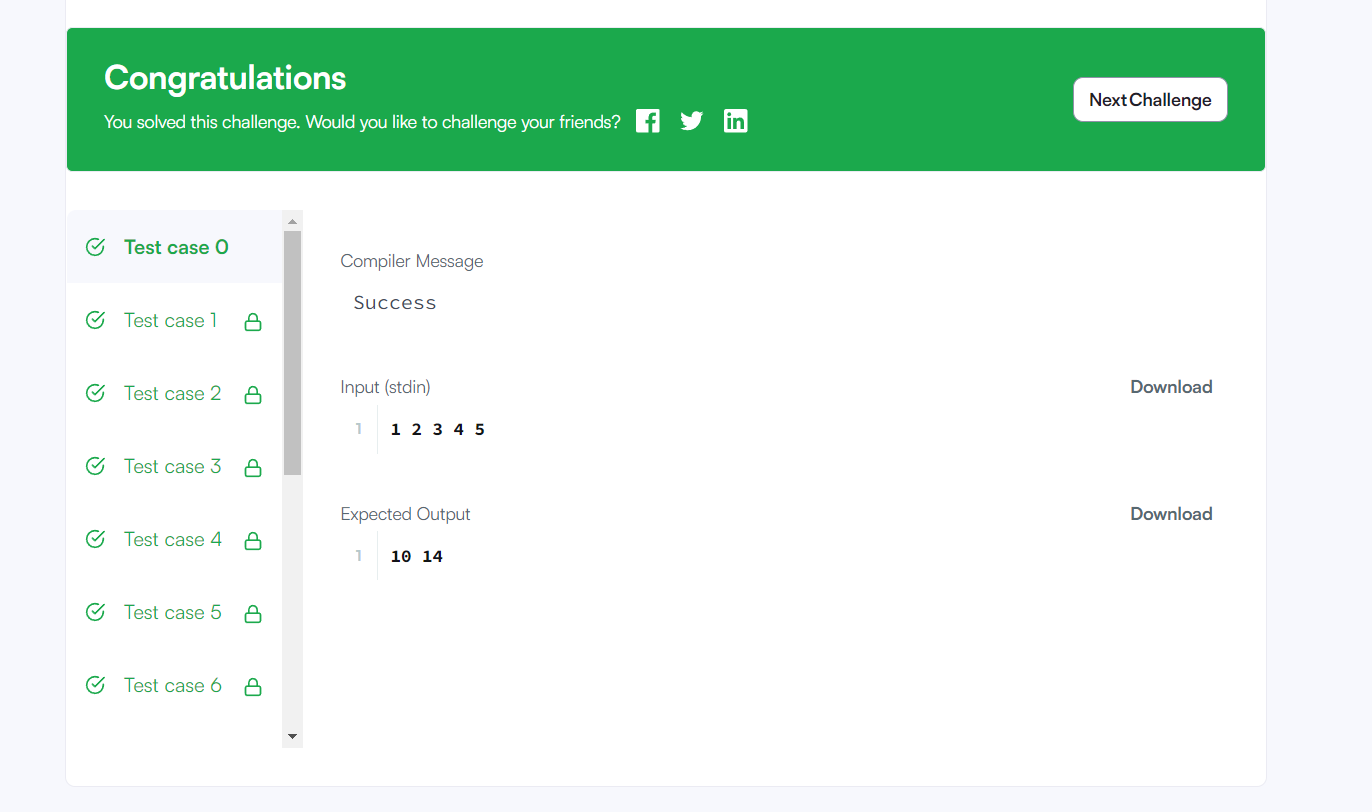
}

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;

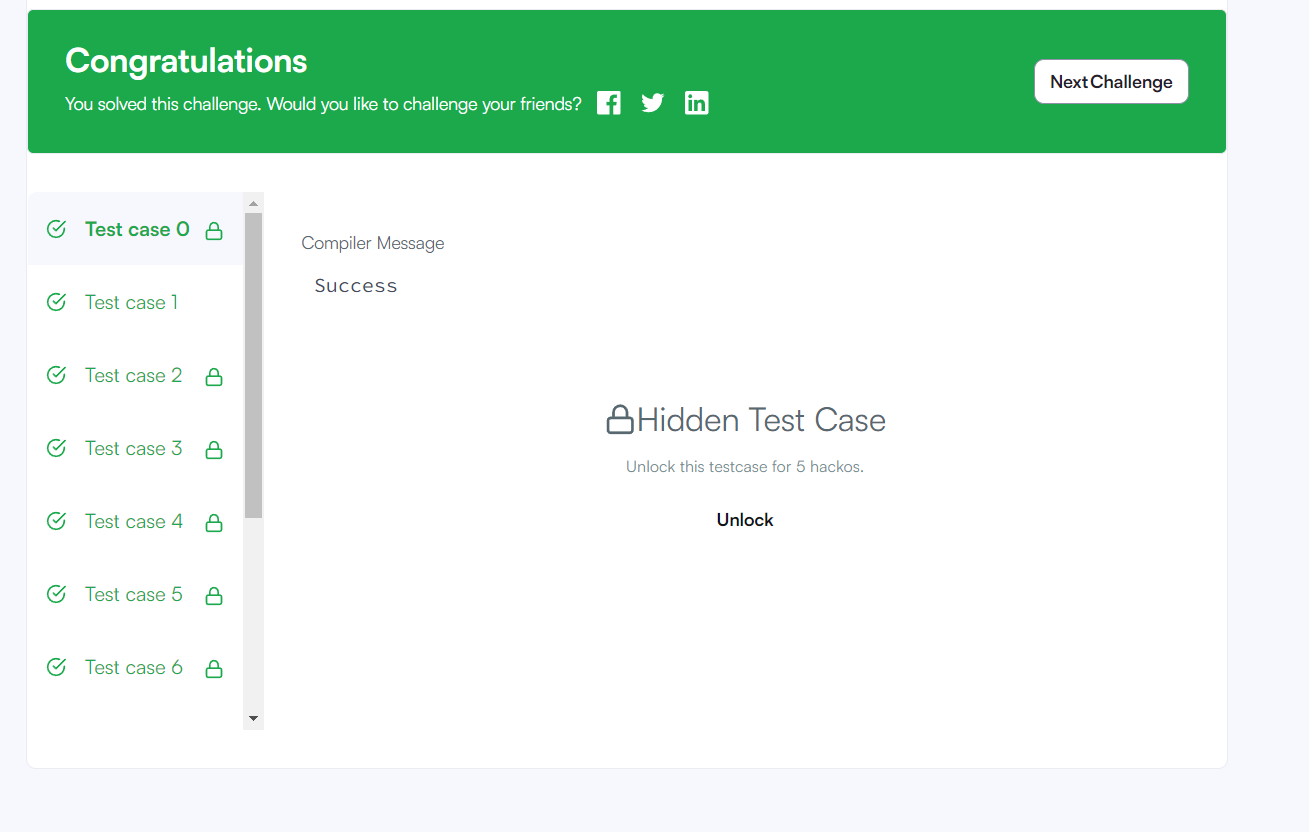
}

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(),

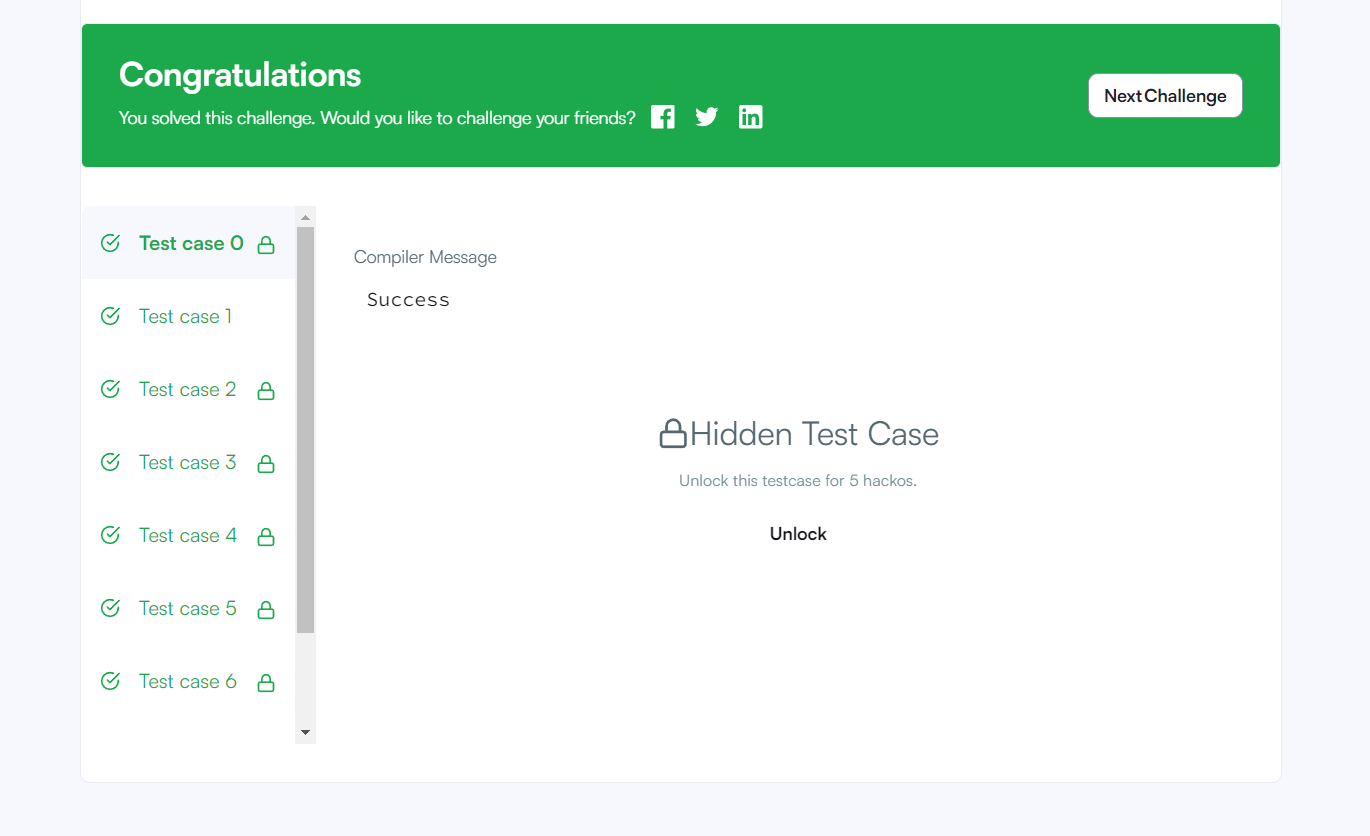
s.end()

);

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;

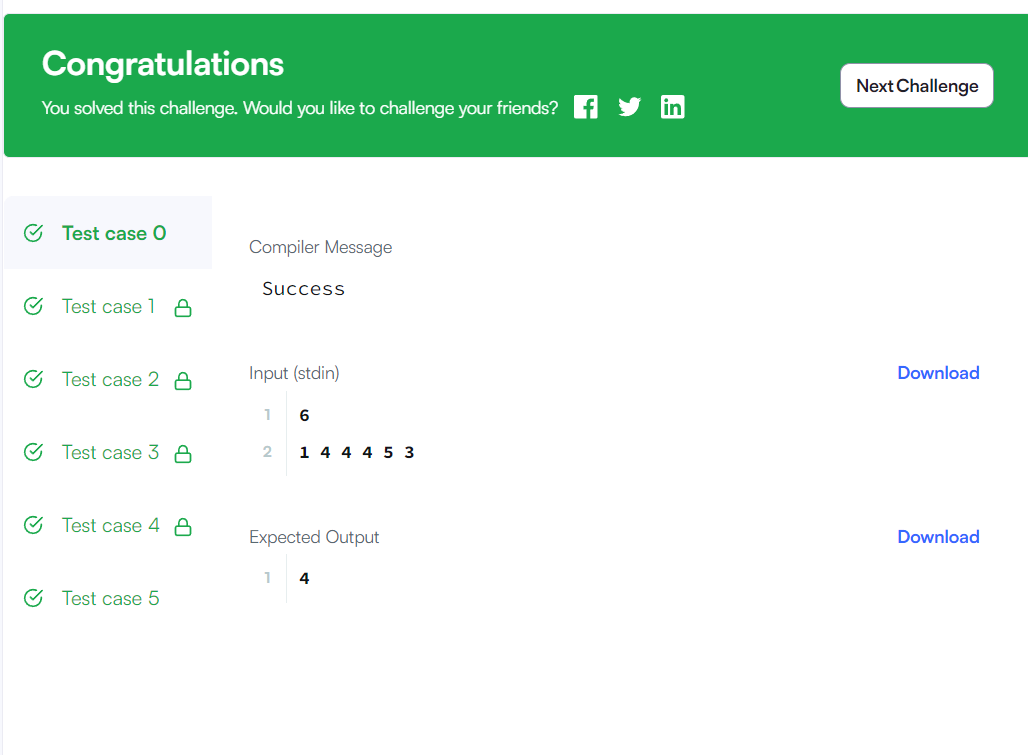
}

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;

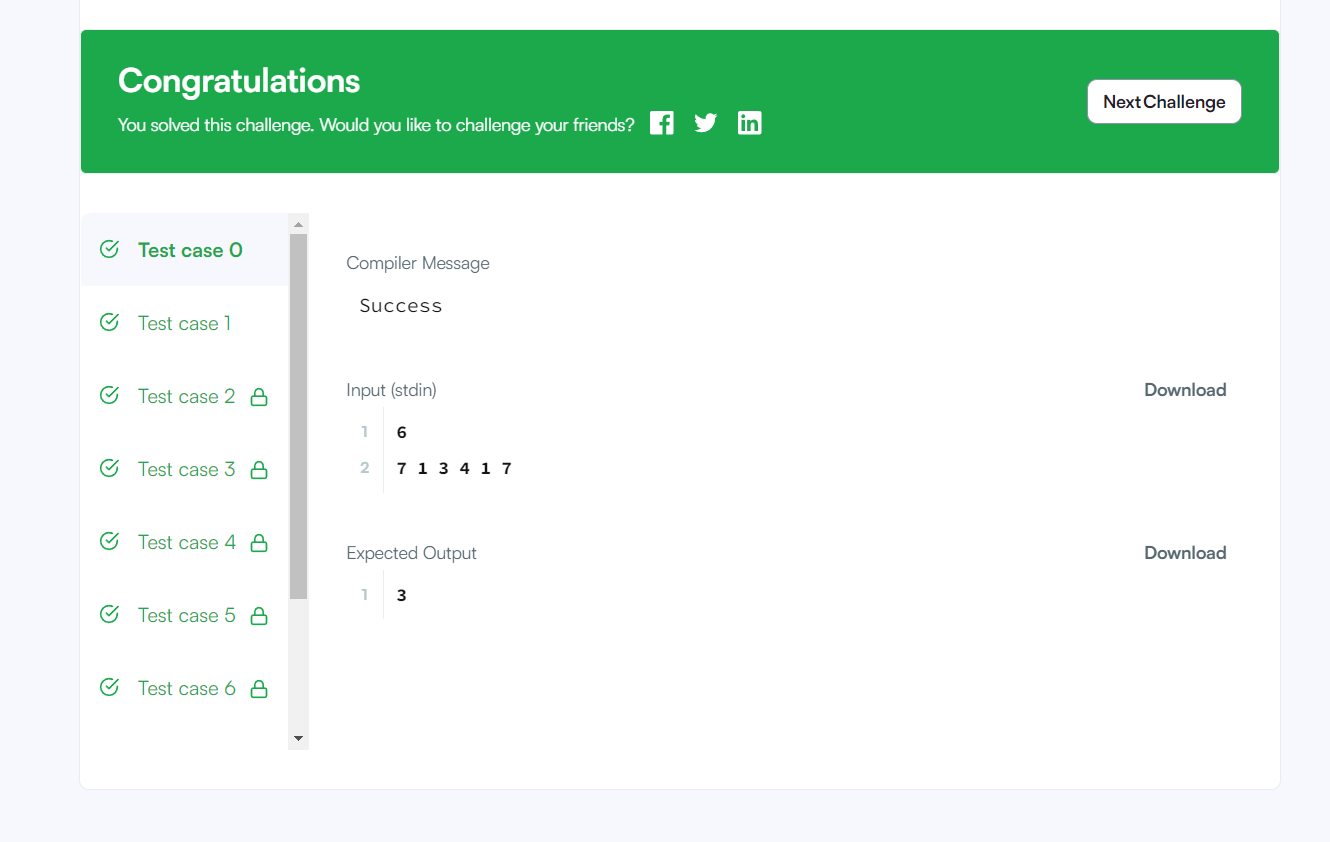
}

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;

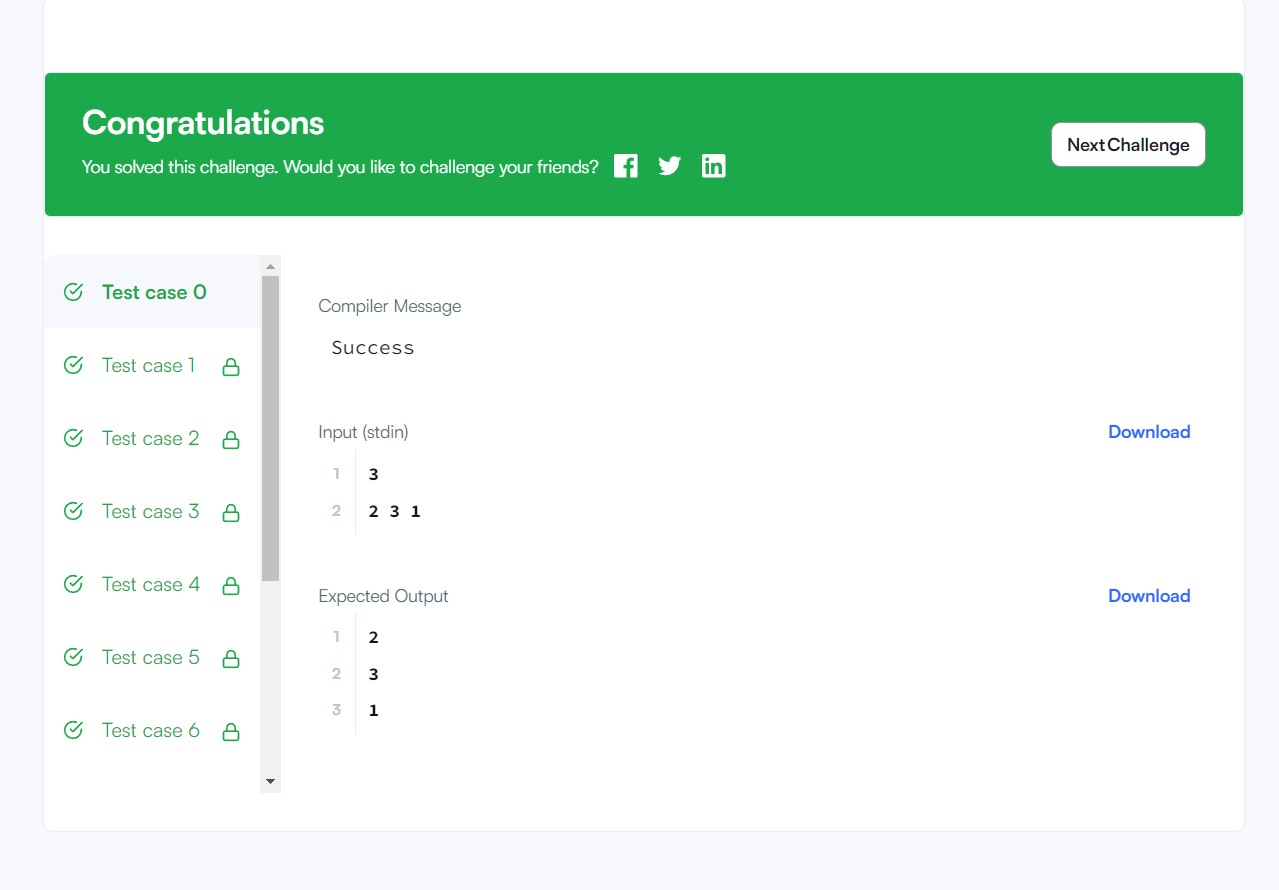
}

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;**

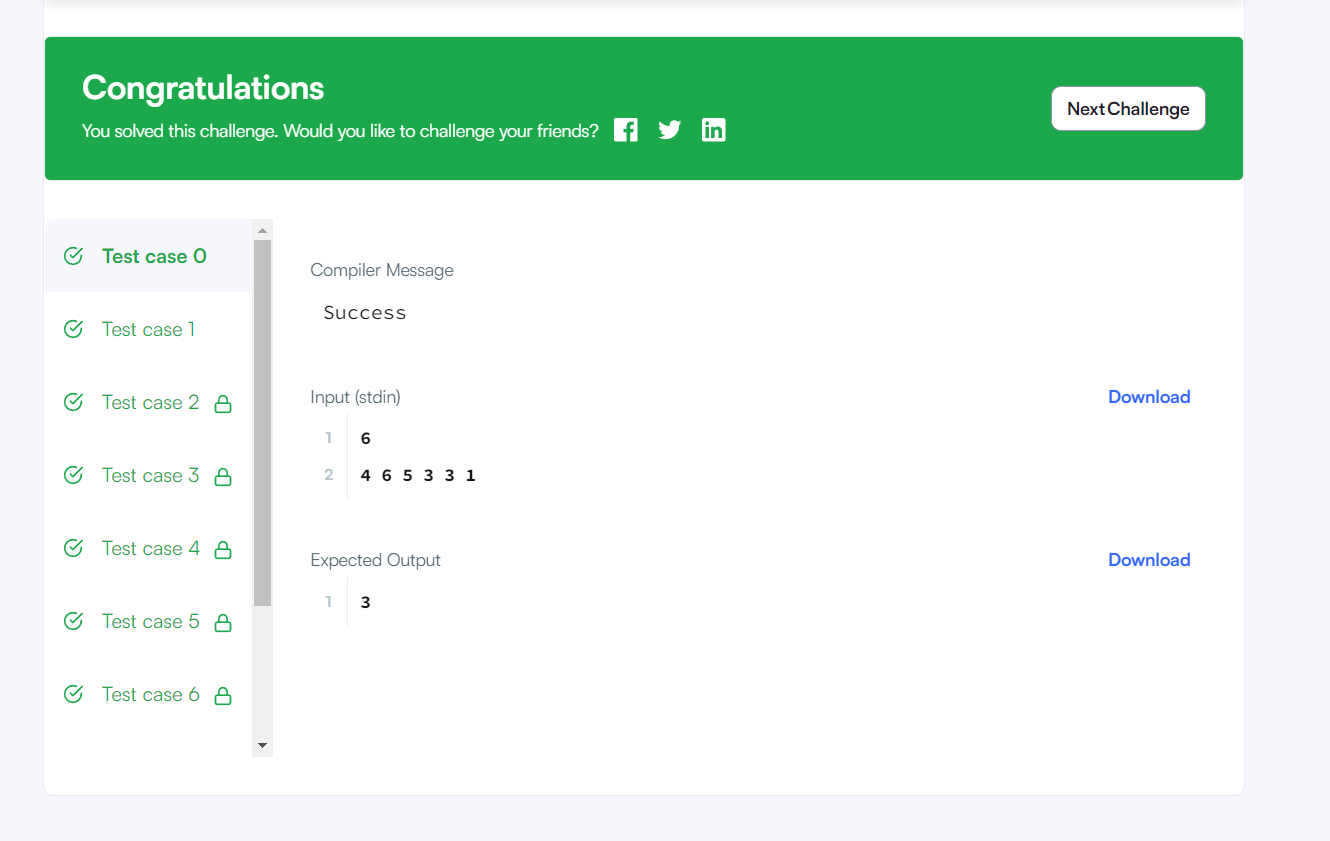
**}**

**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(),

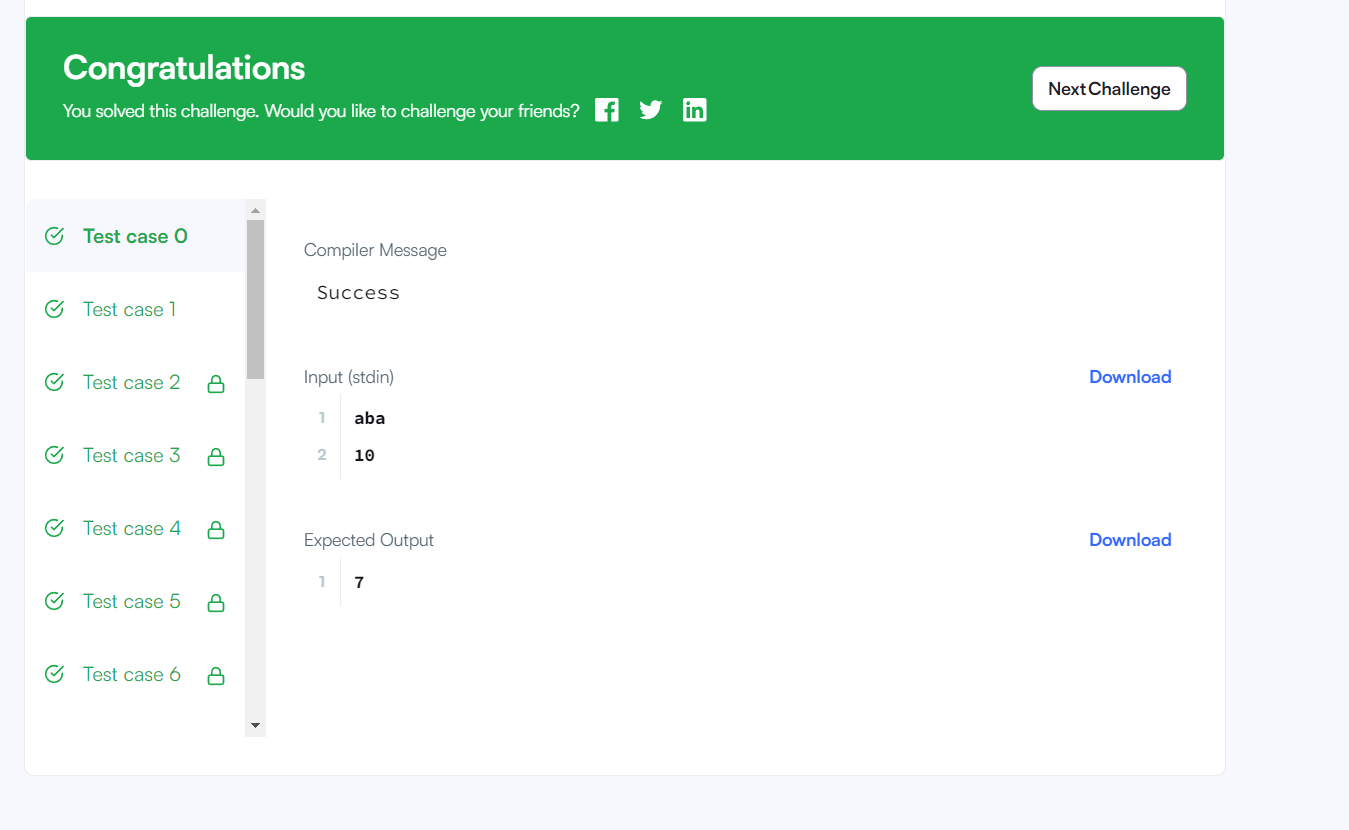
s.end()

);

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;

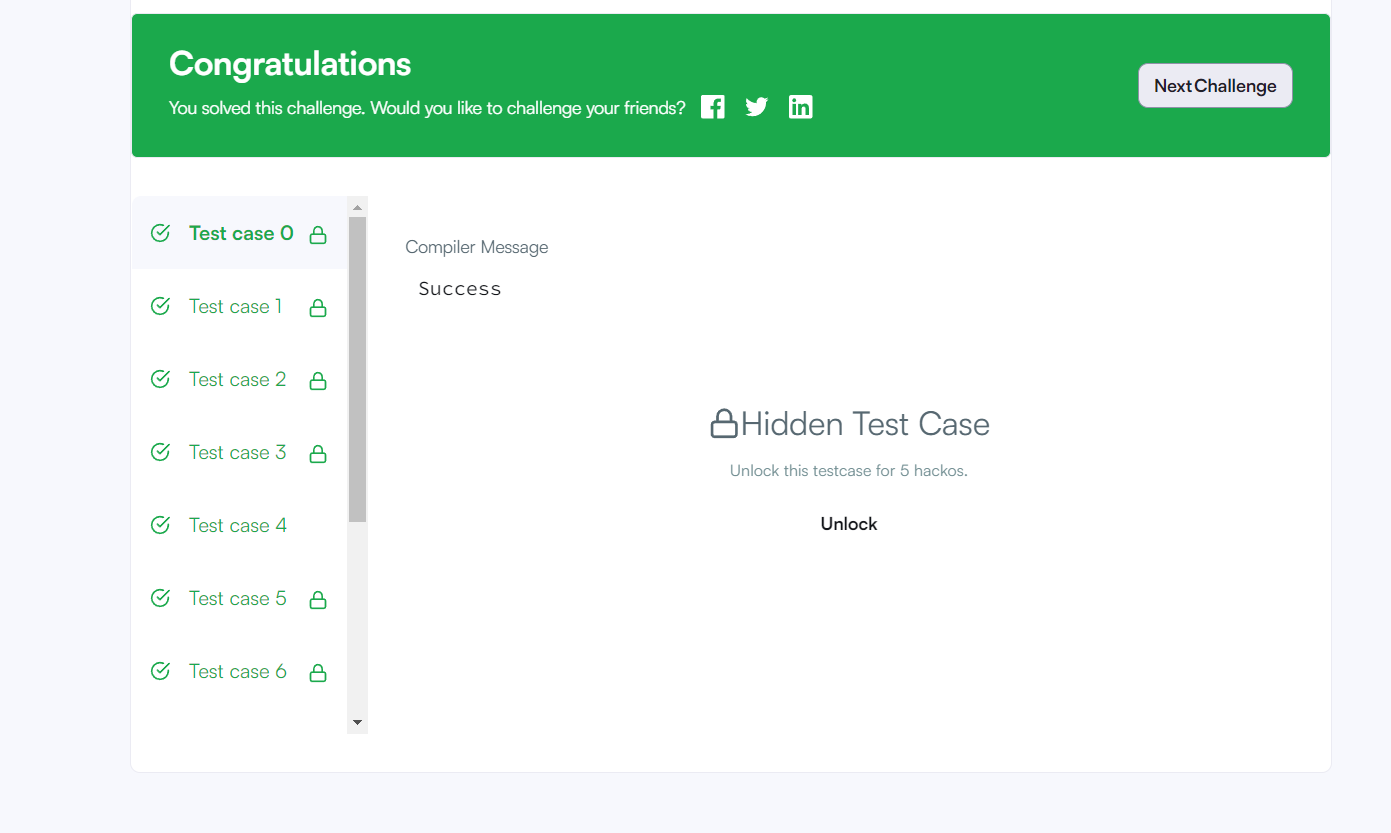
}

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;

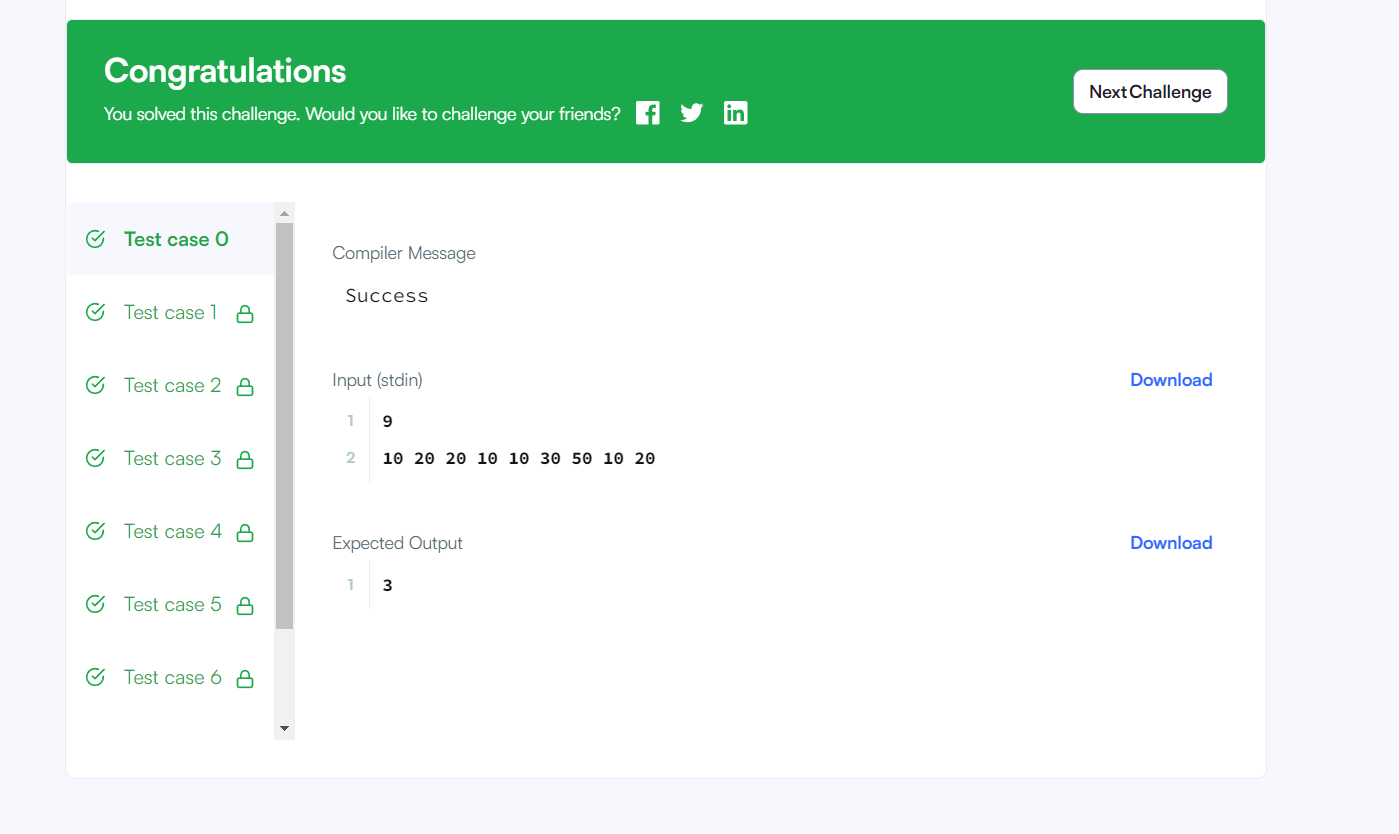
}

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;

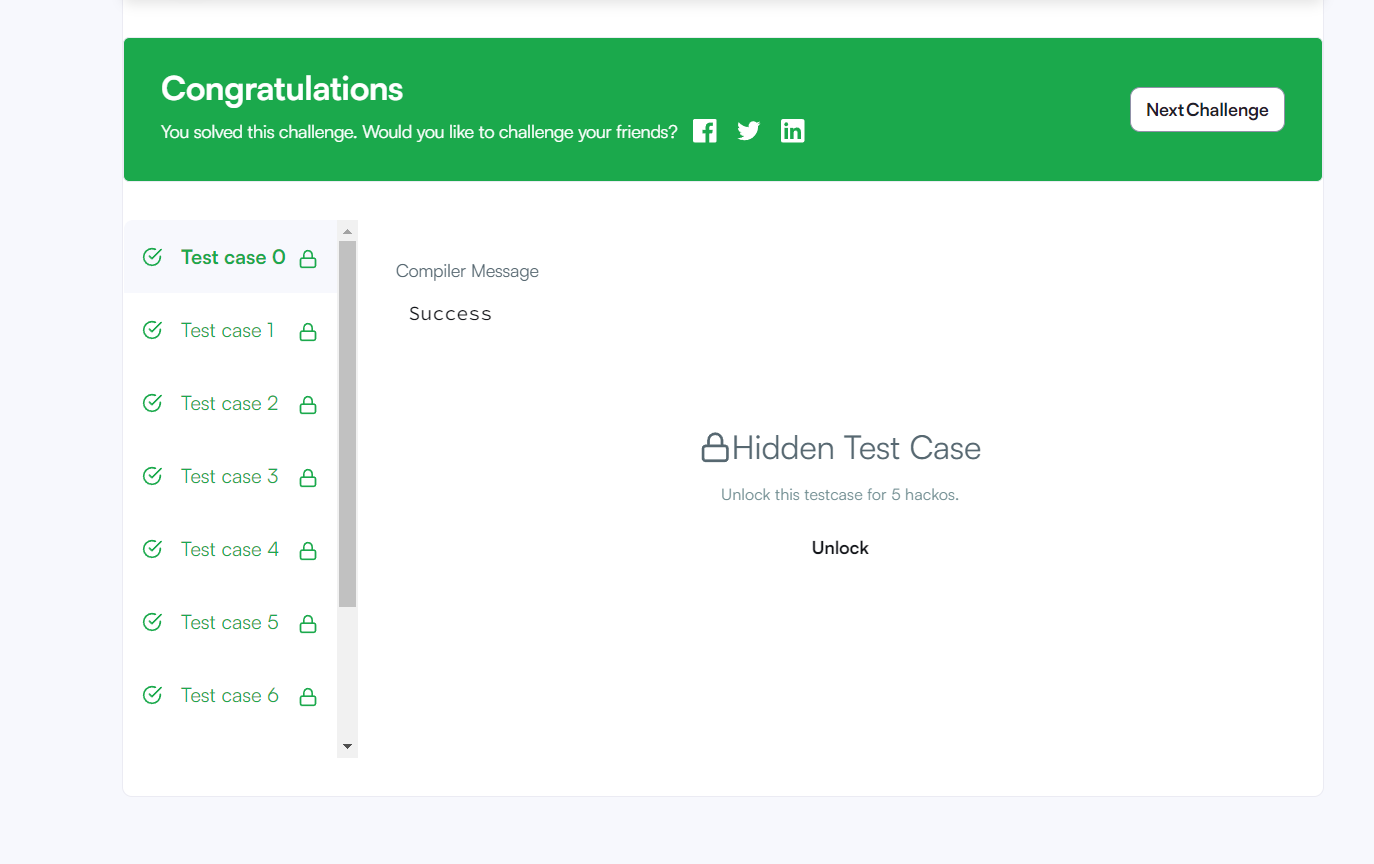
}

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;

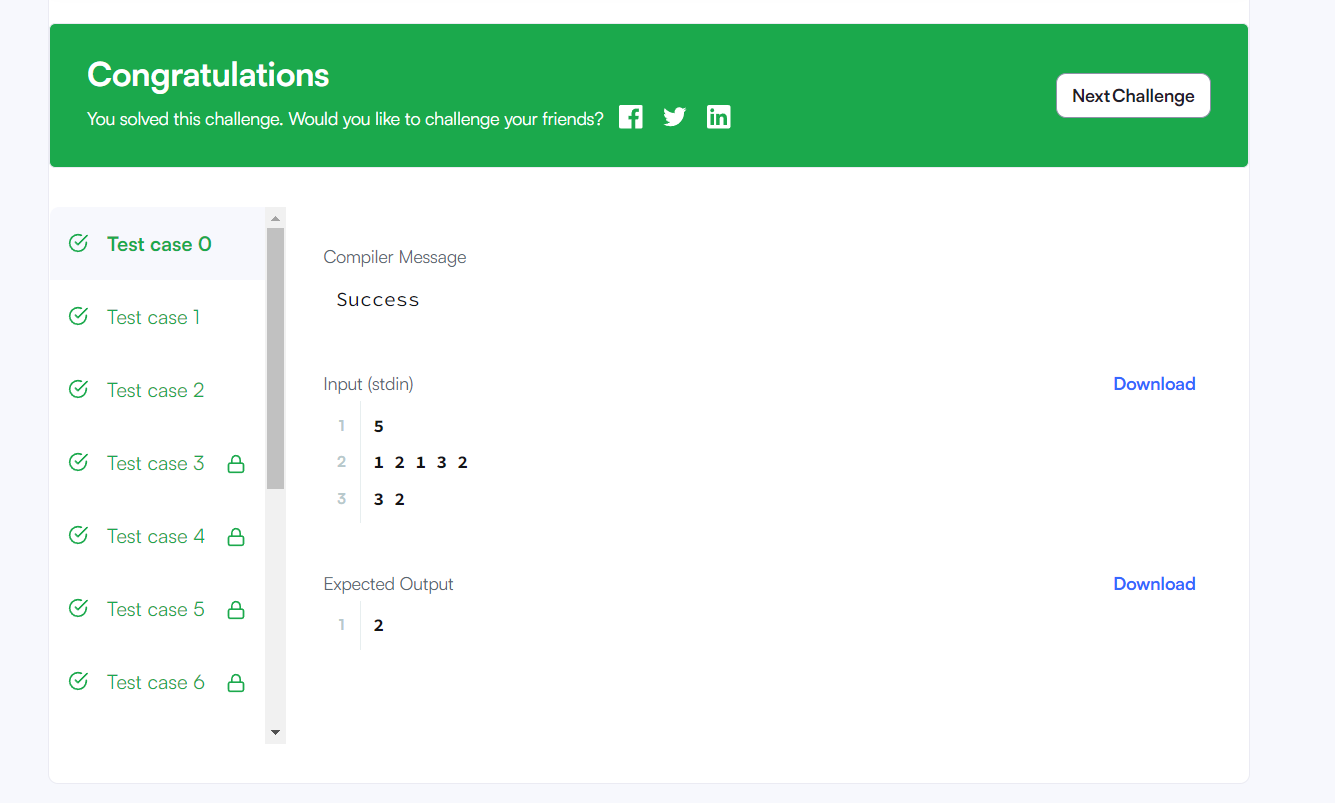
}

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(),

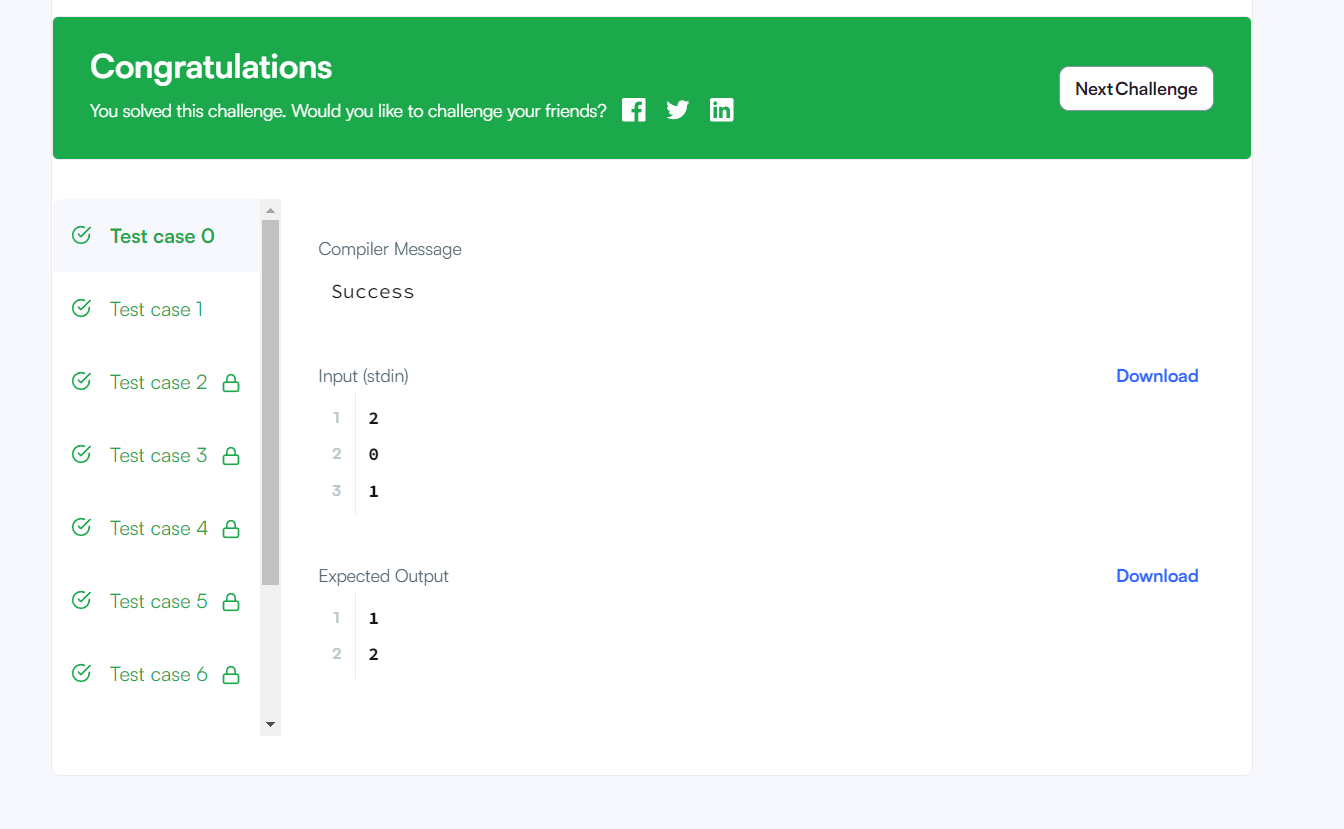
s.end()

);

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;

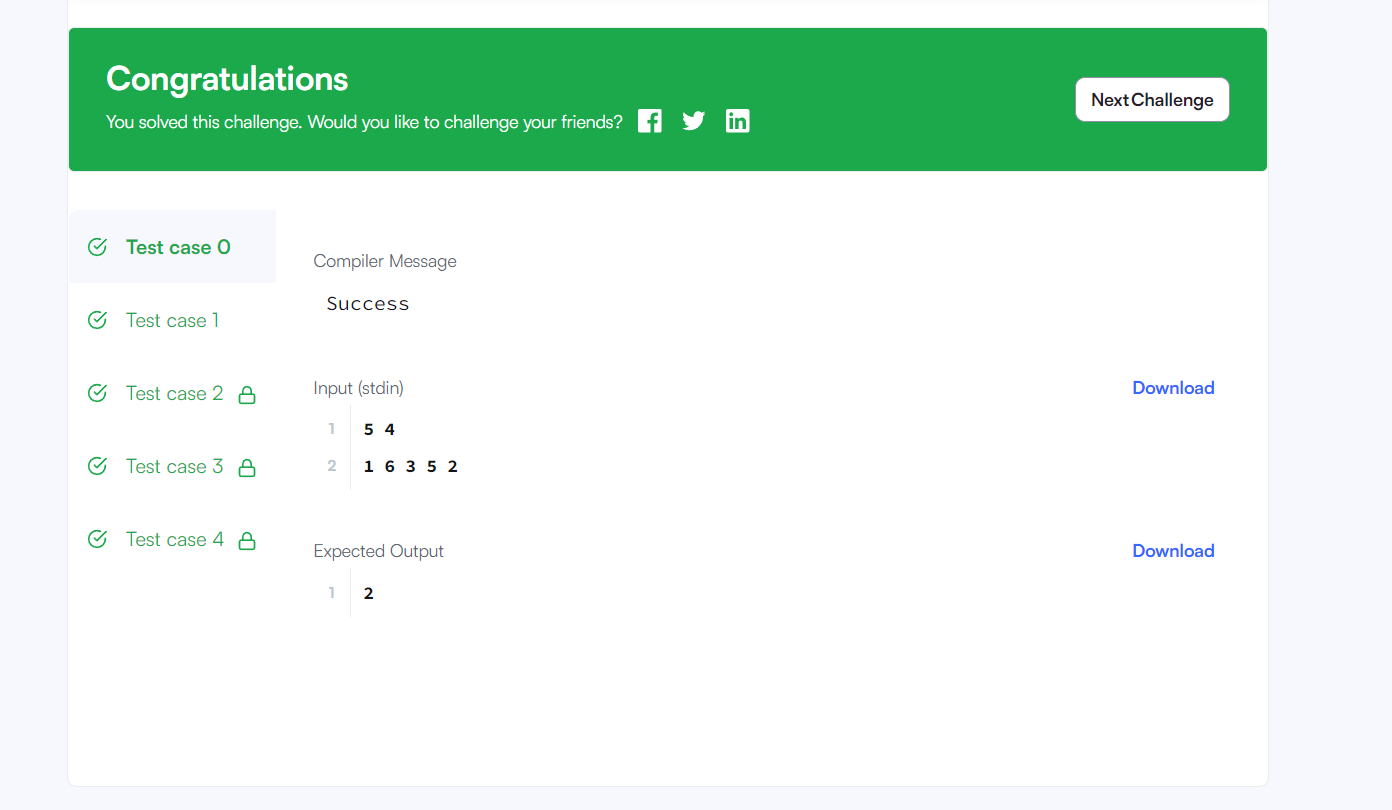
}

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;

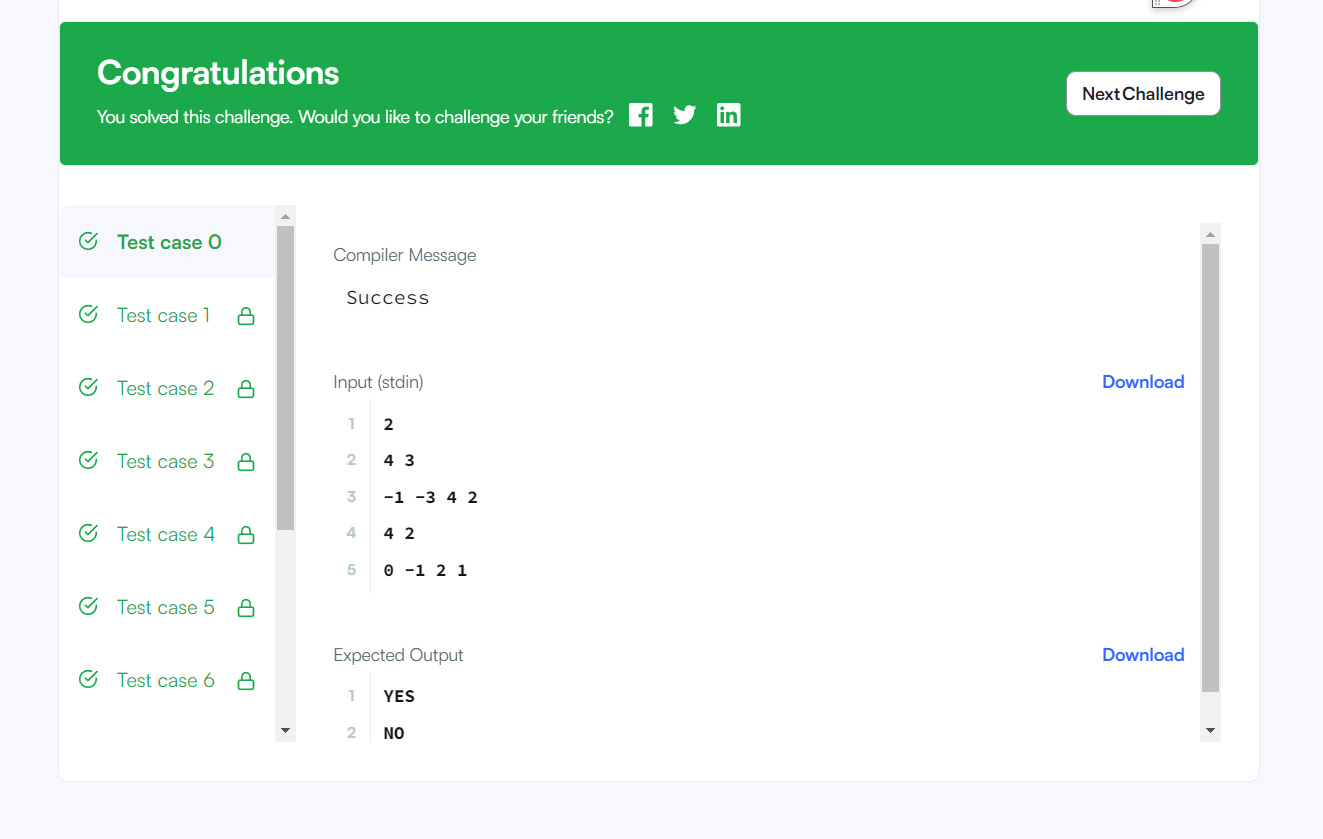
}

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;

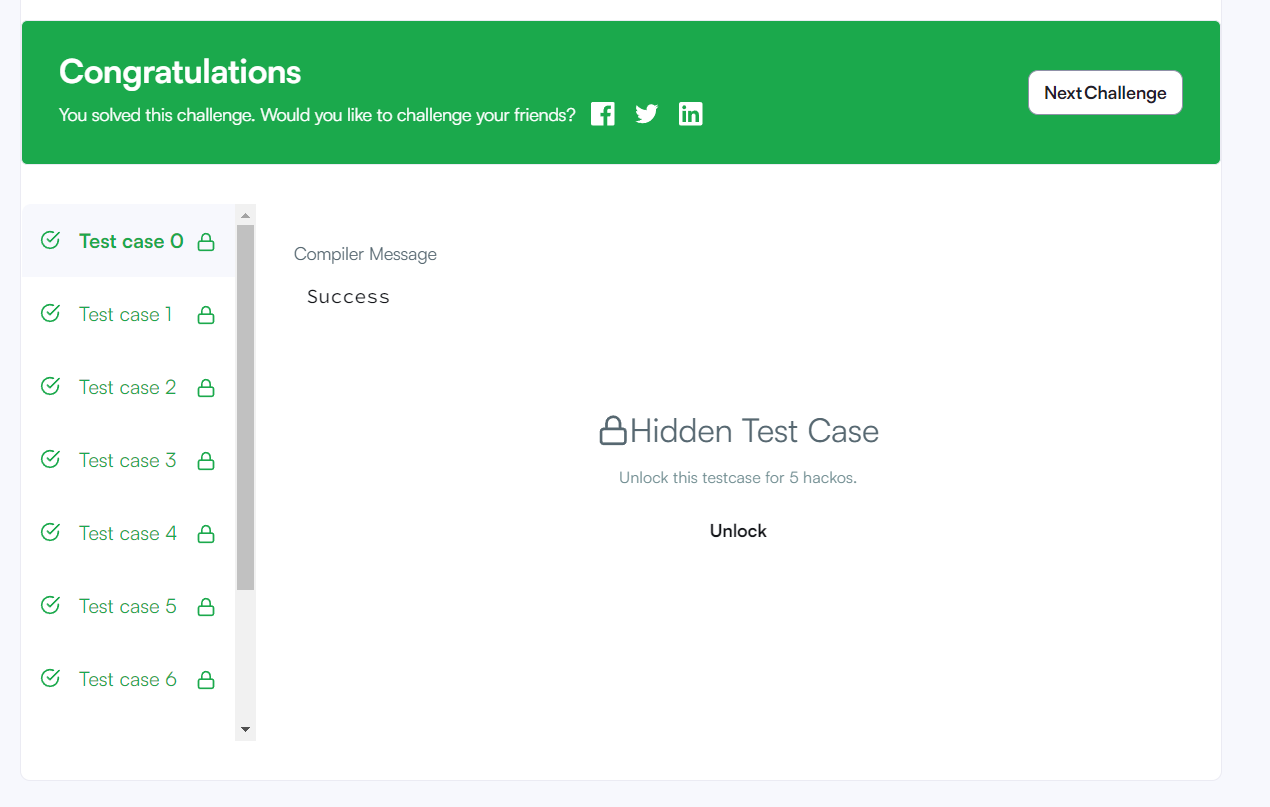
}

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);

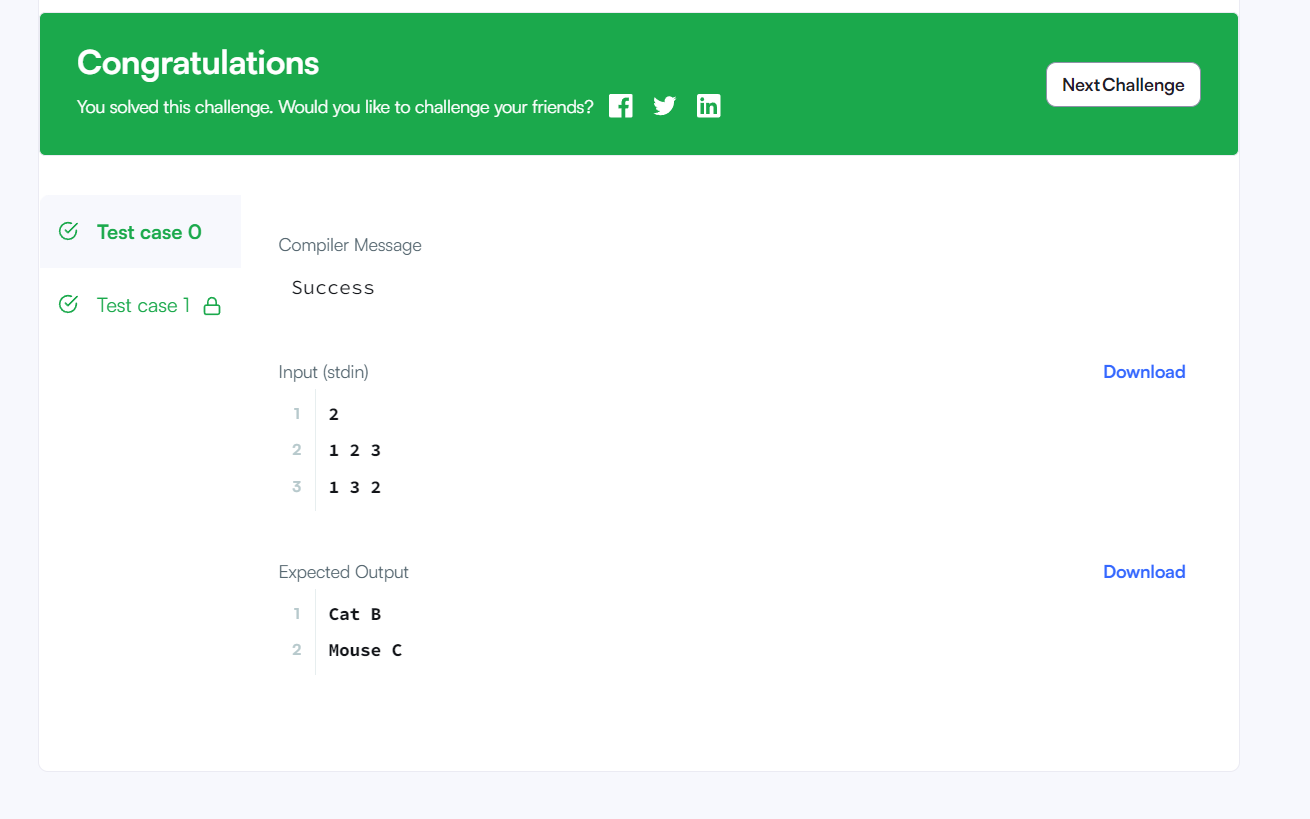
}

splits.push\_back(input\_string.substr(i, min(pos, input\_string.length()) - i + 1));

return splits;

}

Output :-



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 :-

