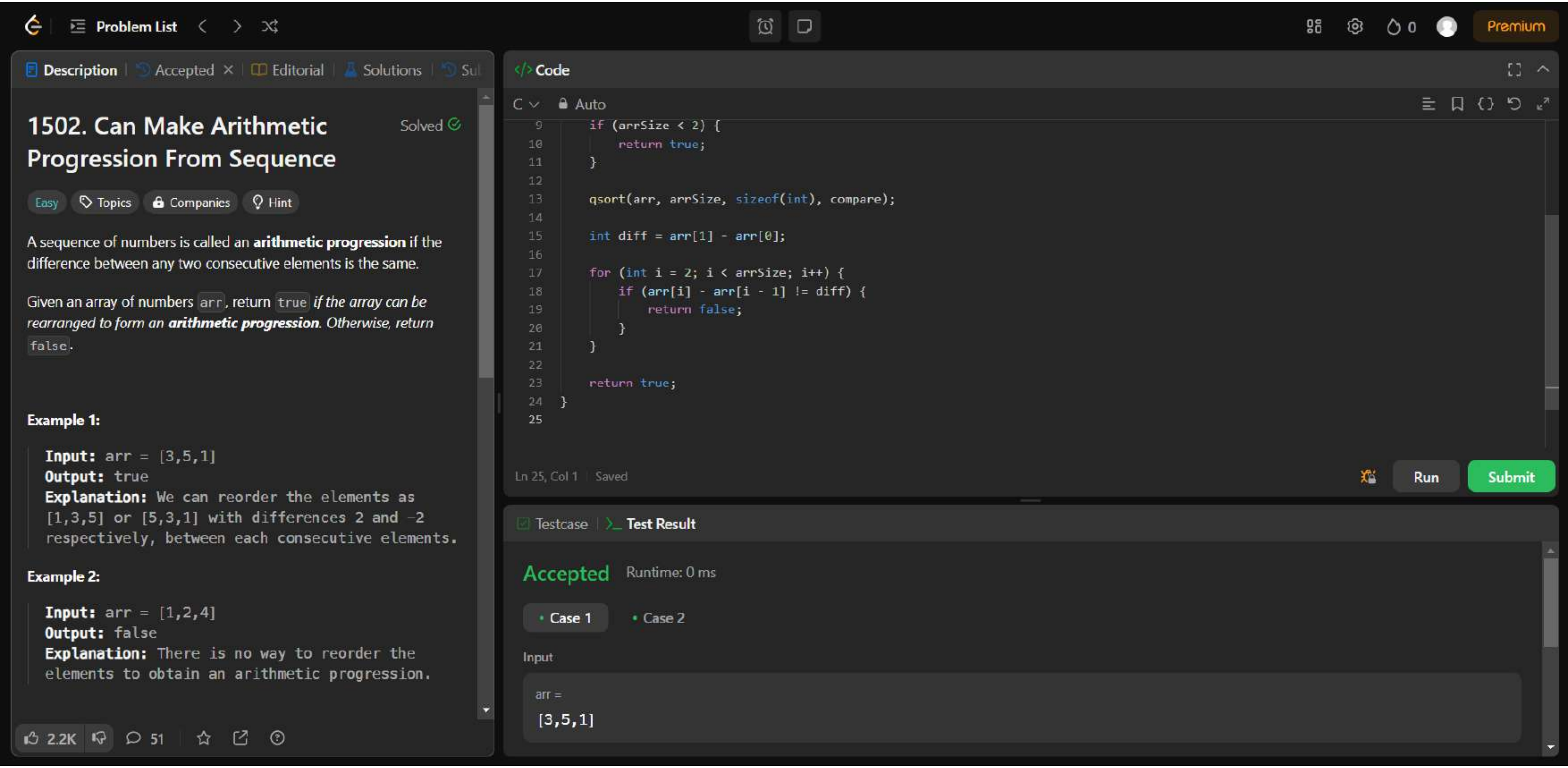


1502. Can Make Arithmetic Progression From Sequence

A sequence of numbers is called an **arithmetic progression** if the difference between any two consecutive elements is the same.

Given an array of numbers `arr`, return `true` *if the array can be rearranged to form an* ***arithmetic progression****. Otherwise, return* `false`.

Example 1:

```
Input: arr = [3,5,1]
Output: true
Explanation: We can reorder the elements as
[1,3,5] or [5,3,1] with differences 2 and -2
respectively, between each consecutive elements.
```

Example 2:

```
Input: arr = [1,2,4]
Output: false
Explanation: There is no way to reorder the
elements to obtain an arithmetic progression.
```

```c
    if (arrSize < 2) {
        return true;
    }

    qsort(arr, arrSize, sizeof(int), compare);

    int diff = arr[1] - arr[0];

    for (int i = 2; i < arrSize; i++) {
        if (arr[i] - arr[i - 1] != diff) {
            return false;
        }
    }

    return true;
}

```

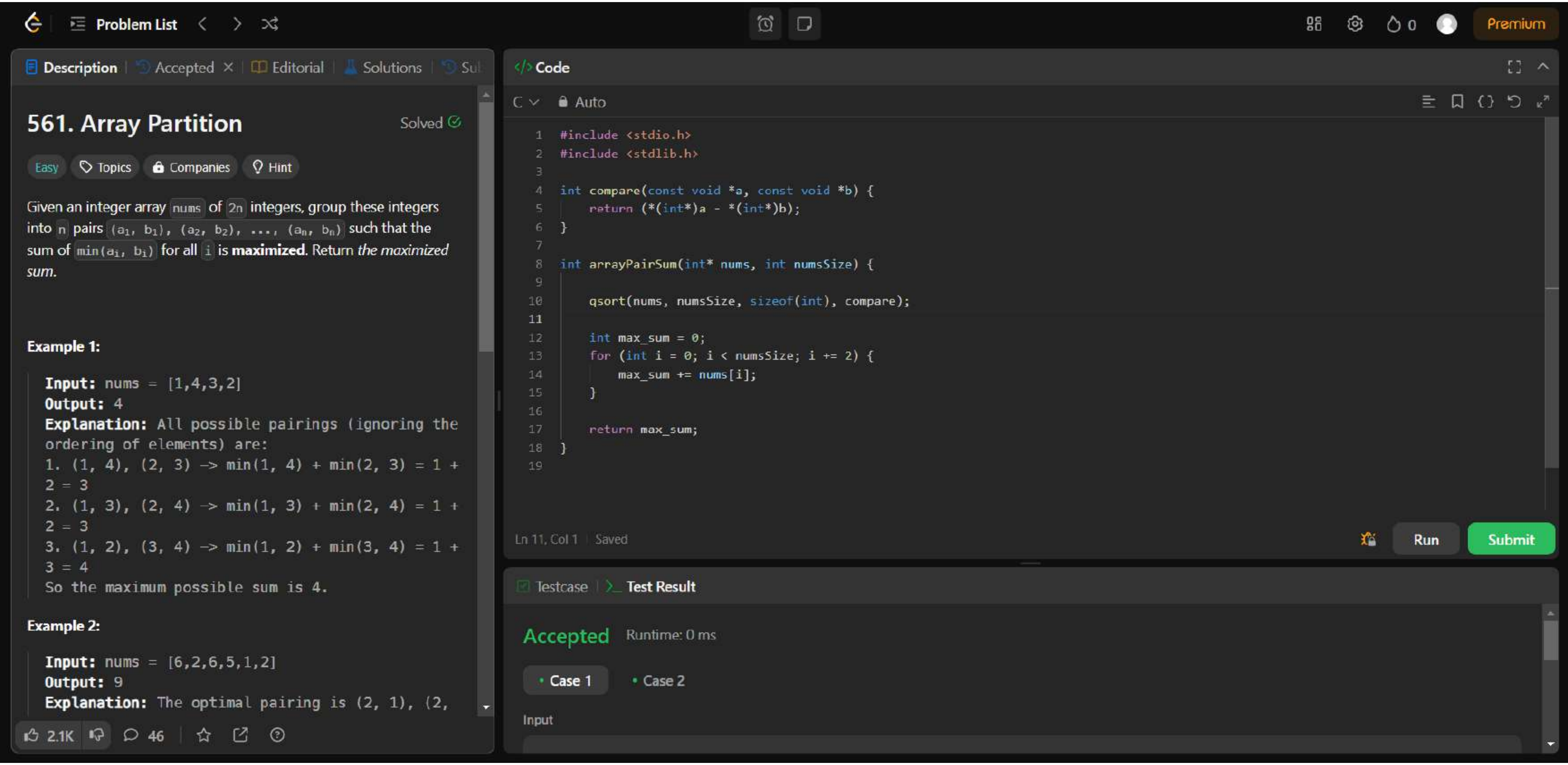
Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

arr =

[3,5,1]



561. Array Partition

Given an integer array `nums` of `2n` integers, group these integers into `n` pairs $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$ such that the sum of $\min(a_i, b_i)$ for all `i` is **maximized**. Return *the maximized sum*.

Example 1:

```
Input: nums = [1,4,3,2]
Output: 4
Explanation: All possible pairings (ignoring the ordering of elements) are:
1. (1, 4), (2, 3) -> min(1, 4) + min(2, 3) = 1 + 2 = 3
2. (1, 3), (2, 4) -> min(1, 3) + min(2, 4) = 1 + 2 = 3
3. (1, 2), (3, 4) -> min(1, 2) + min(3, 4) = 1 + 3 = 4
So the maximum possible sum is 4.
```

Example 2:

```
Input: nums = [6,2,6,5,1,2]
Output: 9
Explanation: The optimal pairing is (2, 1), (2,
```

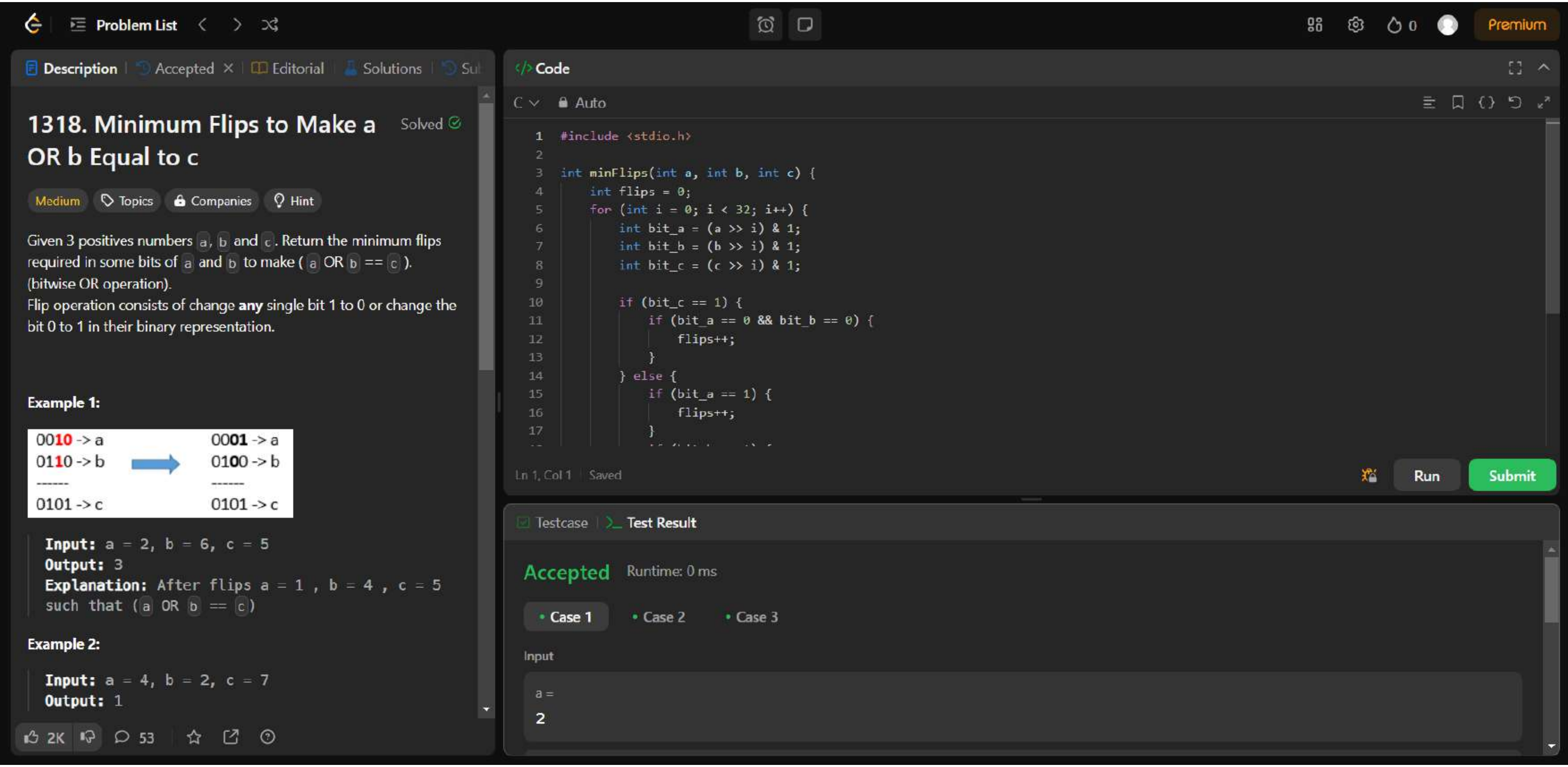
```c
#include <stdio.h>
#include <stdlib.h>

int compare(const void *a, const void *b) {
    return (*(int*)a - *(int*)b);
}

int arrayPairSum(int* nums, int numsSize) {

    qsort(nums, numsSize, sizeof(int), compare);

    int max_sum = 0;
    for (int i = 0; i < numsSize; i += 2) {
        max_sum += nums[i];
    }

    return max_sum;
}
```

Ln 11, Col 1 | Saved

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input



1318. Minimum Flips to Make a OR b Equal to c

Solved ✓

Given 3 positives numbers a, b and c. Return the minimum flips required in some bits of a and b to make (a OR b == c). (bitwise OR operation).
Flip operation consists of change **any** single bit 1 to 0 or change the bit 0 to 1 in their binary representation.

Example 1:

```
0010 -> a        0001 -> a
0110 -> b        0100 -> b
-------          -------
0101 -> c        0101 -> c
```

Input: a = 2, b = 6, c = 5
Output: 3
Explanation: After flips a = 1 , b = 4 , c = 5 such that (a OR b == c)

Example 2:

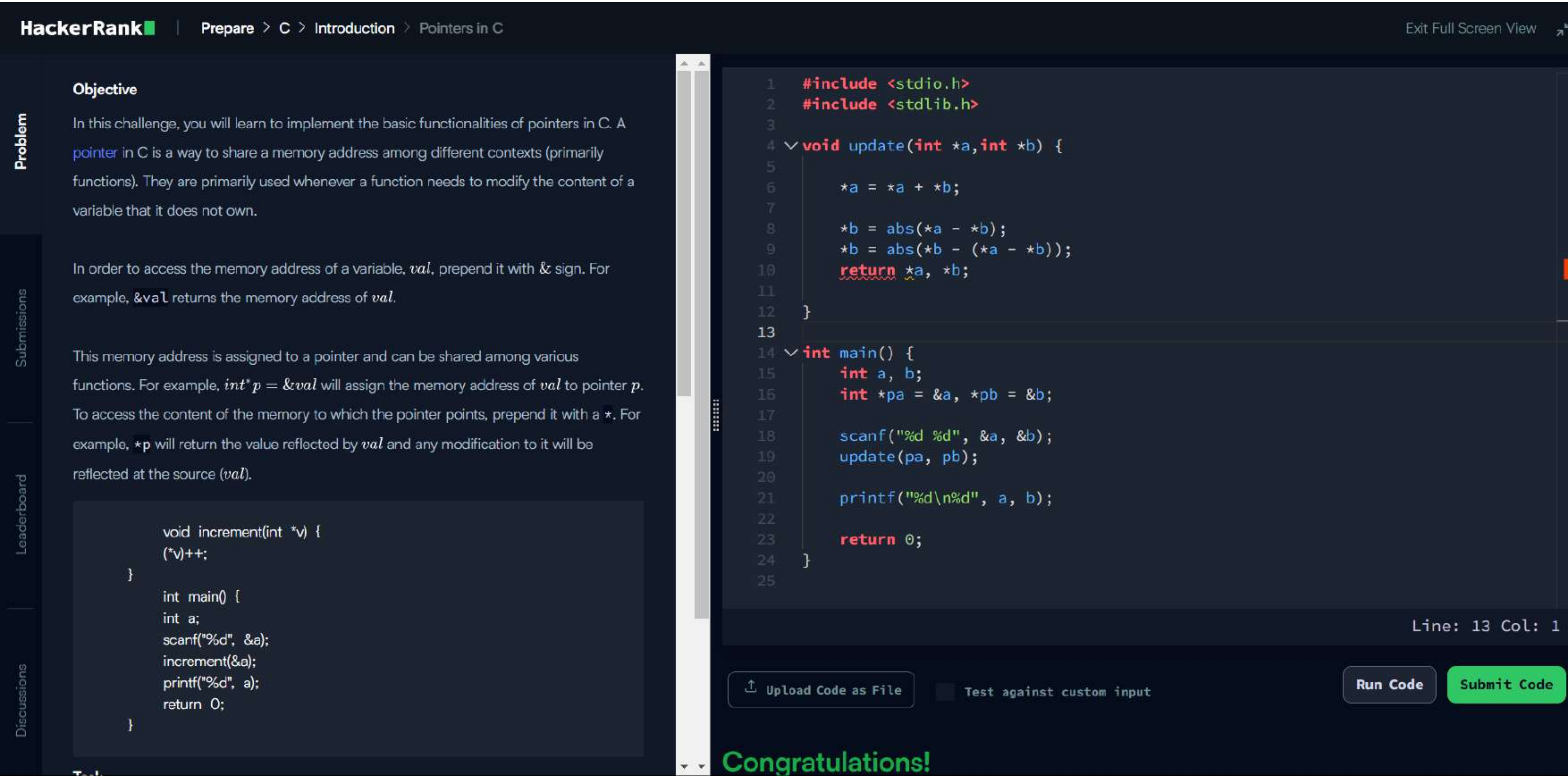
Input: a = 4, b = 2, c = 7
Output: 1

```c
#include <stdio.h>

int minFlips(int a, int b, int c) {
    int flips = 0;
    for (int i = 0; i < 32; i++) {
        int bit_a = (a >> i) & 1;
        int bit_b = (b >> i) & 1;
        int bit_c = (c >> i) & 1;

        if (bit_c == 1) {
            if (bit_a == 0 && bit_b == 0) {
                flips++;
            }
        } else {
            if (bit_a == 1) {
                flips++;
            }
```

Accepted Runtime: 0 ms

Input

a =
2



Objective

In this challenge, you will learn to implement the basic functionalities of pointers in C. A pointer in C is a way to share a memory address among different contexts (primarily functions). They are primarily used whenever a function needs to modify the content of a variable that it does not own.

In order to access the memory address of a variable, val, prepend it with $\&$ sign. For example, `&val` returns the memory address of val.

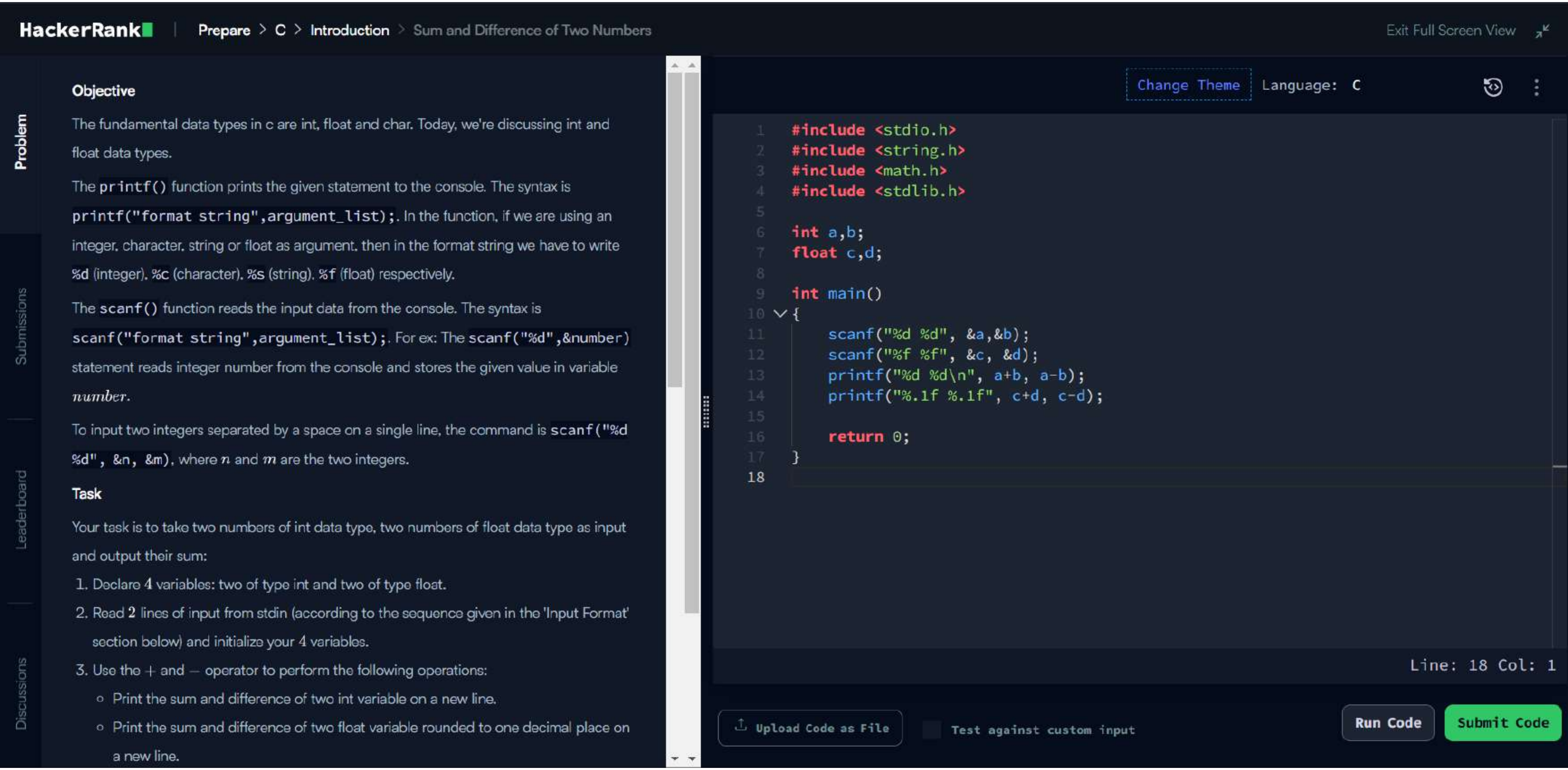
This memory address is assigned to a pointer and can be shared among various functions. For example, $int^{*}p = \&val$ will assign the memory address of val to pointer p. To access the content of the memory to which the pointer points, prepend it with a `*`. For example, `*p` will return the value reflected by val and any modification to it will be reflected at the source (val).

```
        void increment(int *v) {
        (*v)++;
    }
        int main() {
        int a;
        scanf("%d", &a);
        increment(&a);
        printf("%d", a);
        return 0;
    }
```

```c
#include <stdio.h>
#include <stdlib.h>

void update(int *a,int *b) {

    *a = *a + *b;

    *b = abs(*a - *b);
    *b = abs(*b - (*a - *b));
    return *a, *b;

}

int main() {
    int a, b;
    int *pa = &a, *pb = &b;

    scanf("%d %d", &a, &b);
    update(pa, pb);

    printf("%d\n%d", a, b);

    return 0;
}
```

Line: 13 Col: 1

Congratulations!



Objective

The fundamental data types in c are int, float and char. Today, we're discussing int and float data types.

The `printf()` function prints the given statement to the console. The syntax is `printf("format string",argument_list);`. In the function, if we are using an integer, character, string or float as argument, then in the format string we have to write %d (integer), %c (character), %s (string), `%f` (float) respectively.

The `scanf()` function reads the input data from the console. The syntax is `scanf("format string",argument_list);`. For ex: The `scanf("%d",&number)` statement reads integer number from the console and stores the given value in variable $number$.

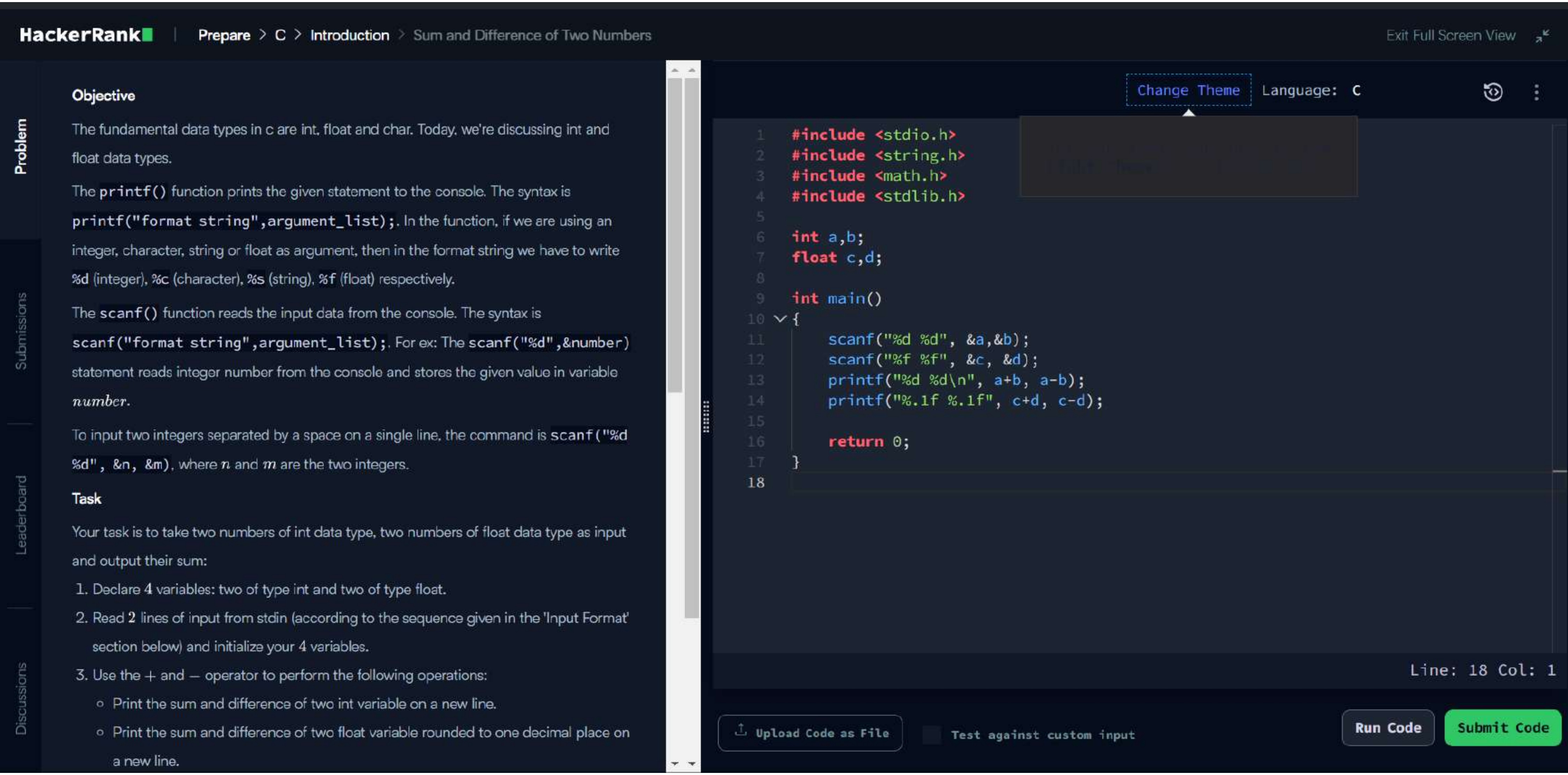
To input two integers separated by a space on a single line, the command is `scanf("%d %d", &n, &m)`, where n and m are the two integers.

Task

Your task is to take two numbers of int data type, two numbers of float data type as input and output their sum:

1. Declare 4 variables: two of type int and two of type float.
2. Read 2 lines of input from stdin (according to the sequence given in the 'Input Format' section below) and initialize your 4 variables.
3. Use the $+$ and $-$ operator to perform the following operations:
   - Print the sum and difference of two int variable on a new line.
   - Print the sum and difference of two float variable rounded to one decimal place on a new line.

```c
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

int a,b;
float c,d;

int main()
{
    scanf("%d %d", &a,&b);
    scanf("%f %f", &c, &d);
    printf("%d %d\n", a+b, a-b);
    printf("%.1f %.1f", c+d, c-d);

    return 0;
}
```



Objective

The fundamental data types in c are int, float and char. Today, we're discussing int and float data types.

The `printf()` function prints the given statement to the console. The syntax is `printf("format string",argument_list);`. In the function, if we are using an integer, character, string or float as argument, then in the format string we have to write %d (integer), %c (character), %s (string), `%f` (float) respectively.

The `scanf()` function reads the input data from the console. The syntax is `scanf("format string",argument_list);`. For ex: The `scanf("%d",&number)` statement reads integer number from the console and stores the given value in variable $number$.

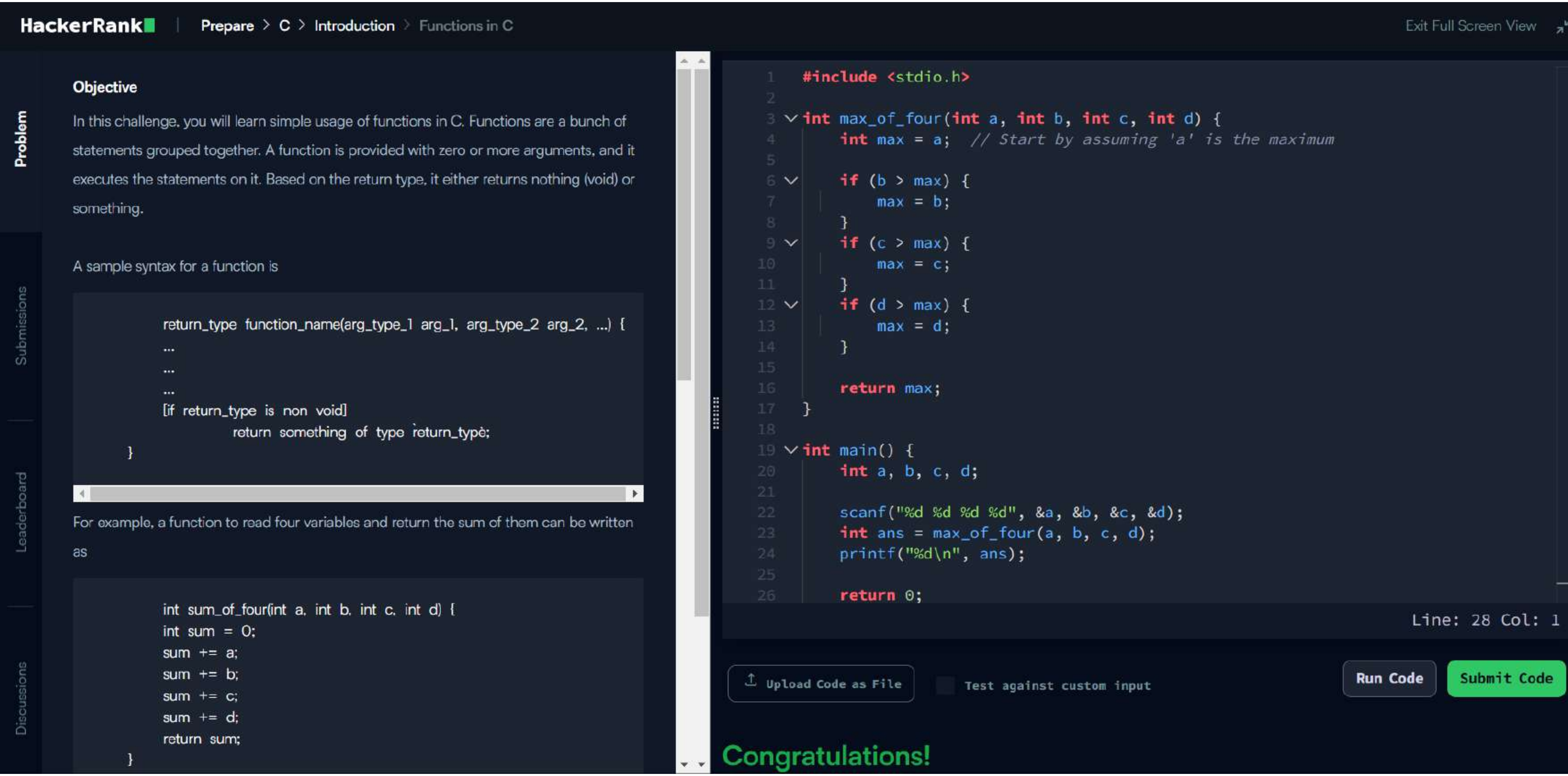
To input two integers separated by a space on a single line, the command is `scanf("%d %d", &n, &m)`, where n and m are the two integers.

Task

Your task is to take two numbers of int data type, two numbers of float data type as input and output their sum:

1. Declare 4 variables: two of type int and two of type float.
2. Read 2 lines of input from stdin (according to the sequence given in the 'Input Format' section below) and initialize your 4 variables.
3. Use the $+$ and $-$ operator to perform the following operations:
   - Print the sum and difference of two int variable on a new line.
   - Print the sum and difference of two float variable rounded to one decimal place on a new line.

```c
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

int a,b;
float c,d;

int main()
{
    scanf("%d %d", &a,&b);
    scanf("%f %f", &c, &d);
    printf("%d %d\n", a+b, a-b);
    printf("%.1f %.1f", c+d, c-d);

    return 0;
}
```



Objective

In this challenge, you will learn simple usage of functions in C. Functions are a bunch of statements grouped together. A function is provided with zero or more arguments, and it executes the statements on it. Based on the return type, it either returns nothing (void) or something.

A sample syntax for a function is

```
        return_type function_name(arg_type_1 arg_1, arg_type_2 arg_2, ...) {
        ...
        ...
        ...
        [if return_type is non void]
                return something of type `return_type`;
    }
```

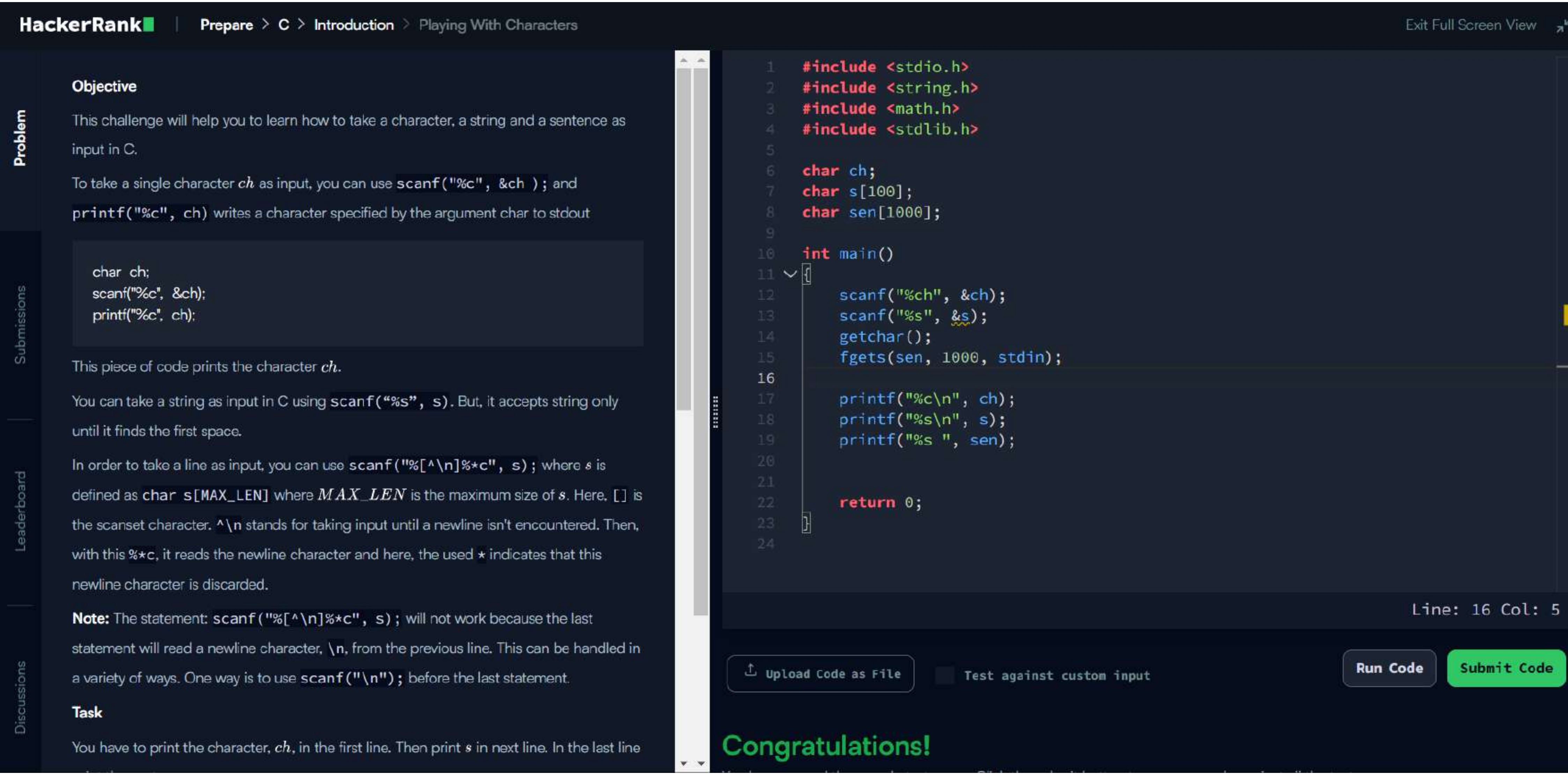
For example, a function to read four variables and return the sum of them can be written as

```
        int sum_of_four(int a, int b, int c, int d) {
        int sum = 0;
        sum += a;
        sum += b;
        sum += c;
        sum += d;
        return sum;
    }
```

```
#include <stdio.h>

int max_of_four(int a, int b, int c, int d) {
    int max = a;  // Start by assuming 'a' is the maximum

    if (b > max) {
        max = b;
    }
    if (c > max) {
        max = c;
    }
    if (d > max) {
        max = d;
    }

    return max;
}

int main() {
    int a, b, c, d;

    scanf("%d %d %d %d", &a, &b, &c, &d);
    int ans = max_of_four(a, b, c, d);
    printf("%d\n", ans);

    return 0;
```

Line: 28 Col: 1

Congratulations!



Objective

This challenge will help you to learn how to take a character, a string and a sentence as input in C.

To take a single character ch as input, you can use `scanf("%c", &ch );` and `printf("%c", ch)` writes a character specified by the argument char to stdout

```
char ch;
scanf("%c", &ch);
printf("%c", ch);
```

This piece of code prints the character ch.

You can take a string as input in C using `scanf(“%s”, s)`. But, it accepts string only until it finds the first space.

In order to take a line as input, you can use `scanf("%[^\n]%*c", s);` where s is defined as `char s[MAX_LEN]` where MAX_LEN is the maximum size of s. Here, `[]` is the scanset character. `^\n` stands for taking input until a newline isn't encountered. Then, with this `%*c`, it reads the newline character and here, the used `*` indicates that this newline character is discarded.

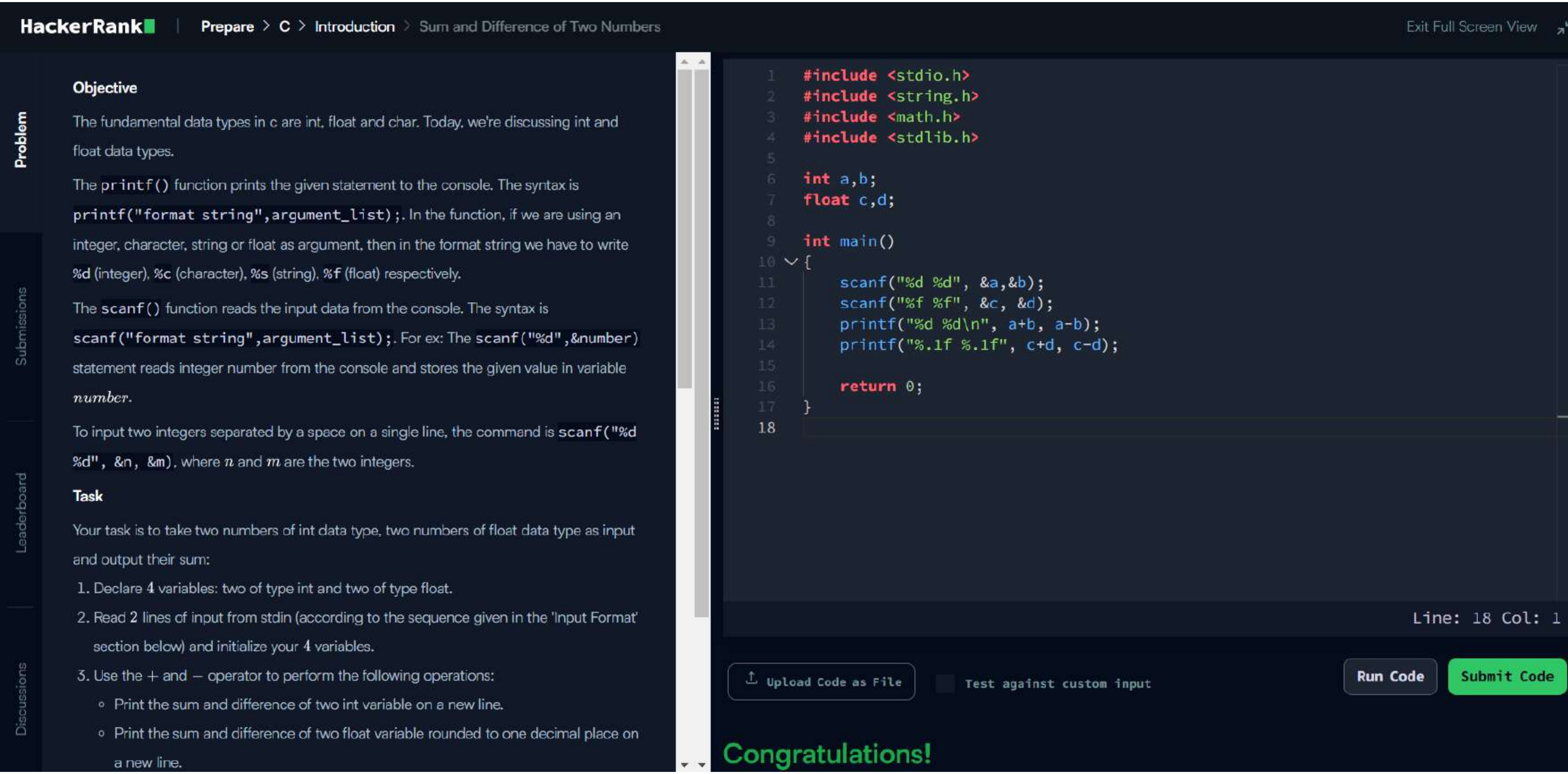
Note: The statement: `scanf("%[^\n]%*c", s);` will not work because the last statement will read a newline character, `\n`, from the previous line. This can be handled in a variety of ways. One way is to use `scanf("\n");` before the last statement.

Task

You have to print the character, ch, in the first line. Then print s in next line. In the last line

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

char ch;
char s[100];
char sen[1000];

int main()
{
    scanf("%ch", &ch);
    scanf("%s", &s);
    getchar();
    fgets(sen, 1000, stdin);

    printf("%c\n", ch);
    printf("%s\n", s);
    printf("%s ", sen);


    return 0;
}
```

Congratulations!



Objective

The fundamental data types in c are int, float and char. Today, we're discussing int and float data types.

The `printf()` function prints the given statement to the console. The syntax is `printf("format string",argument_list);`. In the function, if we are using an integer, character, string or float as argument, then in the format string we have to write %d (integer), %c (character), %s (string), %f (float) respectively.

The `scanf()` function reads the input data from the console. The syntax is `scanf("format string",argument_list);`. For ex: The `scanf("%d",&number)` statement reads integer number from the console and stores the given value in variable $number$.

To input two integers separated by a space on a single line, the command is `scanf("%d %d", &n, &m)`, where n and m are the two integers.

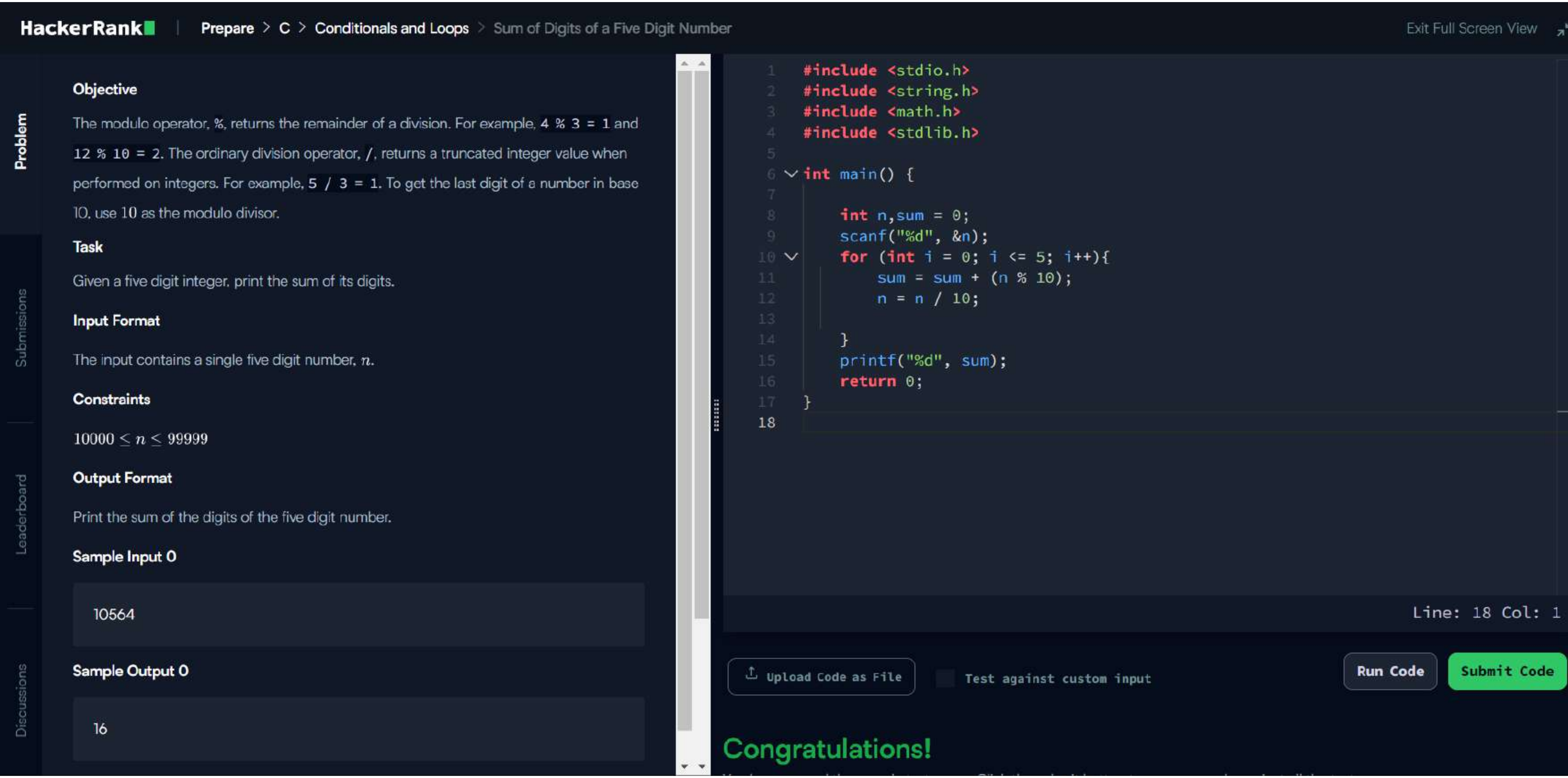
Task

Your task is to take two numbers of int data type, two numbers of float data type as input and output their sum:

1. Declare 4 variables: two of type int and two of type float.
2. Read 2 lines of input from stdin (according to the sequence given in the 'Input Format' section below) and initialize your 4 variables.
3. Use the $+$ and $-$ operator to perform the following operations:
   - Print the sum and difference of two int variable on a new line.
   - Print the sum and difference of two float variable rounded to one decimal place on a new line.

```c
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

int a,b;
float c,d;

int main()
{
    scanf("%d %d", &a,&b);
    scanf("%f %f", &c, &d);
    printf("%d %d\n", a+b, a-b);
    printf("%.1f %.1f", c+d, c-d);

    return 0;
}

```

Congratulations!



Objective

The modulo operator, `%`, returns the remainder of a division. For example, `4 % 3 = 1` and `12 % 10 = 2`. The ordinary division operator, `/`, returns a truncated integer value when performed on integers. For example, `5 / 3 = 1`. To get the last digit of a number in base 10, use 10 as the modulo divisor.

Task

Given a five digit integer, print the sum of its digits.

Input Format

The input contains a single five digit number, n.

Constraints

$10000 \leq n \leq 99999$

Output Format

Print the sum of the digits of the five digit number.

Sample Input 0

```
10564
```

Sample Output 0

```
16
```

```c
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

int main() {

    int n,sum = 0;
    scanf("%d", &n);
    for (int i = 0; i <= 5; i++){
        sum = sum + (n % 10);
        n = n / 10;

    }
    printf("%d", sum);
    return 0;
}

```

Line: 18 Col: 1

Congratulations!