

## Hackerrank Problems

### Objective

In this challenge, we will learn some basic concepts of C that will get you started with the language. You will need to use the same syntax to read input and write output in many C challenges. As you work through these problems, review the code stubs to learn about reading from stdin and writing to stdout.

### Task

This challenge requires you to print on a single line, and then print the already provided input string to `stdout`. If you are not familiar with C, you may want to read about the `printf()` command.

### Example

The required output is:

Hello, World!

Life is beautiful

### Function Description

Complete the `main()` function below.

The `main()` function has the following input:

- string s: a string

### Prints

- \*two strings: \* "Hello, World!" on one line and the input string on the next line.

### Input Format

There is one line of text, .

### Sample Input 0

Welcome to C programming.

### Sample Output 0

Hello, World!

Welcome to C programming.

TestCase:1

### Sample Input 1

Life is Beautiful.

### Sample Output 1

Hello, World!

Life is Beautiful.

TestCase:2

### Sample Input 2

Welcome to Hackerrank.

### Sample Output 2

Hello, World!

Welcome to Hackerrank.

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

int main()
{
    char s[100];
    scanf("%[^\n]%*c", s);
    printf("Hello, World!\n");
    printf("%s",s);
    /* Enter your code here. Read input from STDIN. Print output to STDOUT */
    return 0;
}
```

## 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 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 .

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, , in the first line. Then print in next line. In the last line print the sentence, .

### Input Format

First, take a character, as input.

Then take the string, as input.

Lastly, take the sentence as input.

### Constraints

Strings for and will have fewer than 100 characters, including the newline.

### Output Format

Print three lines of output. The first line prints the character, .

The second line prints the string, .

The third line prints the sentence, .

### Sample Input 0

```
C
Language
Welcome To C!!
```

### Sample Output 0

```
C
Language
Welcome To C!!
```

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

```
#include <string.h>
```

```
#include <math.h>
```

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

```
int main()
```

```
{
```

```
    char ch,s[10],sen[100] ;
```

```
    scanf("%c \n %s %[^\n]c", &ch, s, sen);
```

```
    printf("%c\n", ch);
```

```
    printf("%s\n", s);
```

```
    printf("%s", sen);
```

```
    return 0;
```

```
}
```

### Characters used in C

Alphabet:

A, B, C, D, ..., Y, Z

a, b, c, d, ..., y, z

Digits:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Special symbols:

~'!@#%^^&\*()\_ - + =|{ } [ ] ; : " ' < > , . ? /

## Variables

Variable is an entity that may be changed during the execution of the program.

By declaring a variable, you allocate space in the computer memory to store either an integer or a float or a character.

An integer variable can store only an integer constant, a float variable can store only a float constant, and a character variable can store only a character constant.

Rules:

- ➤   •A variable name can have 1 to 31 characters (letters or digits or underscores). Some compilers allow up to 247 characters.
- ➤   •First character must be a letter from the English alphabet or an underscore.
- ➤   •Commas or blanks are not allowed.
- ➤   •Other than underscore ' \_ ', no special symbols can be used in the variable.

Examples:

\_pi roi sum gross\_salary num i j a cost cost\_per\_unit

Look at these variables and mention whether they are in the correct representation or not. If not, mention the reason.

2

Variable 7sum sum7 \_average average\_of 89\_

\_89

rate of interest a&b

a\_b

Year Date-of-birth

## Keywords

“Keywords cannot be used as variable names”

Keywords (Reserved words) are the words whose meaning has already been stored in the compiler. There are 32 keywords in C.

List of Keywords:

int while default return

float char Double for switch Case do enum Extern sizeof static Struct

long break register typedef

const goto short union

if else auto continue unsigned signed void volatile

+456 -345 -20302

Must have at least 1 digit. Decimal point is not allowed.  
Can be either positive or negative Default sign is positive  
Commas / blanks are not allowed

## Constants

What is the output for this pseudocode? {

a=5

a=8

print a

}

Output: \_\_\_\_\_

Note: A memory location can hold only one value at a time. We can overwrite but cannot make it hold two values at the same time.

Primary Constants:

Types:

➤ Integer constant

➤ Real constant / Floating point constant

Character constant

Range for integer constants:

16 bit compiler 32 bit compiler 64 bit compiler

: -32768 to 32767  $\rightarrow (-2^{16})$  to  $(2^{16} - 1)$

:  $(-2^{32})$  to  $(2^{32} - 1)$  22

:  $(-2^{64})$  to  $(2^{64} - 1)$  22

Note: In an 'n' bit compiler, the left most bit represents the sign.

'0' is to represent a positive number.

'1' is to represent a negative number.

Represent the given binary numbers in integer form: (4 bit compiler)

1111 : \_\_\_\_\_

1000 : \_\_\_\_\_

0110 : \_\_\_\_\_

0010 : \_\_\_\_\_

Real constant / Floating point constant: Examples:

+231.57 +6654.0 -12.54785 1.46e-2

Rules:

-436.12 -342.4E21

-1.02e-3

- Must have at least 1 digit
- Must have a decimal point
- Could be either positive or negative ➤ Default sign is positive
- No commas or blanks allowed

Exponential form of representation:

34.21e6

The part before the 'e' is mantissa. The part after the 'e' is exponent. 34.21e6

34210000.000000

34.21e-4  
123.45e-6  
123.456e-3

Rules for exponential representation:

- Mantissa and exponent should be separated by the letter e or E
- Mantissa can be either positive or negative. Default is positive.
- Exponent must have at least 1 digit. It can be either positive or negative. Default is positive.

Range for real constants / floating point constants:

-3.4e38 to 3.4e38

Note:

$2.35e4 = 2.35 \times 10^4$   $2.35e-4 = 2.35 \div 10^4$

Character constant:

Examples:

'A' 'a' 'K' '7' '#' '=' Rules:

- Must be a single alphabet or a single digit or a single special symbol.
- Must be enclosed within single inverted commas. The inverted commas should point left.
- 'C' is a valid character constant. 'C' is not valid character constant.
- Maximum length must be 1 character.

## Our First C Program

Each instruction is written as a separate statement. Multiple statements put together for a program. The statements must appear in the order of execution you want to be executed. All statements are entered in small cases.

Every C statement must end with a ;. We call ; as the statement terminator.

Program to calculate the sum of first n natural numbers: Line

Number

- 1 #include<stdio.h>
- 2 void main( )

3{

- 4 int n, sum;
- 5 n=5;
- 6 /\* Formula to find the sum of first n integers is  $n \times (n+1) / 2$  \*/
- 7 sum =  $n \times (n+1) / 2$ ;
- 8 printf("The sum is %d", sum);

9}

Line 1: #include<stdio.h> is a pre-processor directive. It is used to enable the printf() function.

Line 2: main ( ) is a name given to a set of statements. It must be main ( ). All statements that belong to main ( ) must be enclosed within the braces { } as given in lines 3 and 9. "void" is used when we do not want the main ( ) function to return any values after execution of the function.

Line 4: We are declaring the variables that are to be used in the program. int is the type of the variables n and sum.

Line 5: We are assigning a value to the variable n.

Line 6: Comments are enclosed withing /\* \*/. Nested comments are not allowed.

Line 7: The formula to calculate the sum of the first n natural number is expressed in a way the compiler can understand. When mathematical formulae are used, we need to convert them into compiler understandable format.

Line 8: printf( ) function is used to show output on the screen. The content inside the "" will be displayed on the screen once the program is run. %d is used to fetch the value in the memory location of the variable sum. %d is used to fetch integer values.

## Receiving inputs for the program

We use the scanf( ) function to receive the inputs. #include<stdio.h> is mandate to use the scanf( ) function. Sample program:

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

```
main ()
```

```
{
```

```
int num;
```

```
printf("enter a number: ");
```

```
scanf("%d", &num);
```

```
printf("the number you entered is %d", num);
```



Code

5

---

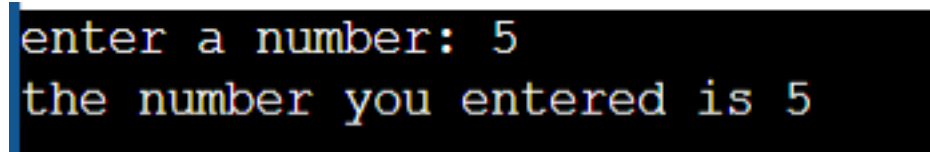
} Output:

Variable declaration

```
int i;  
int e, c, v;  
float pi;  
float expo, net; char name;  
int yr, mnth; float prcnt, avg; char comp;
```

Receiving input

```
scanf("%d", &i); scanf("%d%d%d", &e, &c, &v); scanf("%f", &pi);  
scanf("%f%f", &expo, &net); scanf("%c", &name);  
  
scanf("%d%d%f%f%c", &yr, &mnth, &prcnt, &avg, &comp);
```



While entering values for multiple inputs, you can either use the spacebar key or the tab key or the enter key to enter the successive inputs.

& is used to store the value in the address of the declared variable.

Fill the blanks in the given program to find the average of three values assigned to a, b, and c, where a and b are integers but c is a real constant. The average value should also be a real constant.

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

```
void main ( )
```

```
{
```

```
_____ a, b;
```

```
_____ c;
```

```
_____ avg;
```

```
printf("Enter the values of a and b: ");
```

```
scanf("%____%____", ____a, ____b); printf("Enter the value of c: ");
```

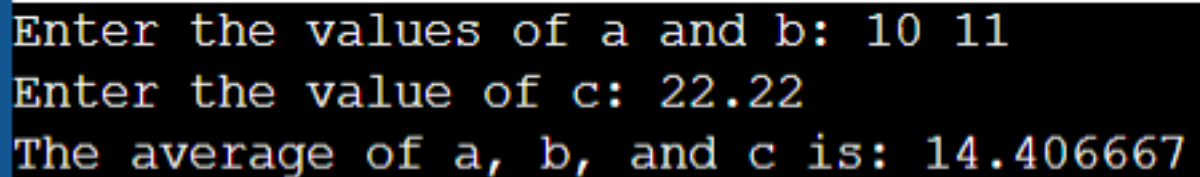
```
scanf("%____", &c); avg = (a+b+c)/3;
```

```
printf("The average of a, b, and c is: _____", avg); }
```

- ➤ •Type Declaration Instruction
- ➤ •Arithmetic Instruction
- ➤ •Control Instruction

Type Declaration Instruction:

- Used to declare the type of variables.



```
Enter the values of a and b: 10 11
Enter the value of c: 22.22
The average of a, b, and c is: 14.406667
```

Instruction

```
int num;
int num1, num2, sum; float prent, avg;
char name, dob;
int a=5, b=2;
float c=1.2, d=34.99; int a=6, b=a+2;
int a=b+2, b=3;
int a,b,c;
a=b=c=20;
int a=b=c=4;
```

Legal / Illegal

```

#include<stdio.h>
void main( )
{
int prin, term;
float interest, si, amt;
prin = 10000;
term = 3;
interest=2.5;
si=prin*term*interest/100;
printf("Simple interest is = %f\n", si);
amt=prin+si;
printf("Amount is = %f", amt);
}

```

Note: There should be only one variable to the left of the assignment operator “=”.

Legal: amt=prin+si; Illegal: prin+si=amt;

+, -, \*, /, % are operators. Variable and constants are the operands. ‘=’ is the assignment operator.

In integer mode arithmetic statement, all operands are integers. Example:

```

int num1, num2, sum;
sum = num1+num2;

```

Right hand side of the assignment operator is executed and then the value is stored in the variable to the left of the assignment operator.

In real mode arithmetic statement, all operands are float values. Example:

```

float a, b, c;
c= a/b;

```

In mixed mode arithmetic statement, some operands are integers, and some are float.

Example:

```

int a, b, c, d;
float percent, frac;

```

```

c= a/b; d=(a/b)*100;

```

Modular division operator (%):

% symbol is used to get the remainder when a value is divided by another. c=a%b; give the remainder when a is divided by b.

Sign of the remainder is same as the sign of the numerator.

Operations on character constants:

Arithmetic operations can be used on int, float, and char. While performing arithmetic operations on char, the ASCII value of the character will be used for the operations.

Example:

```
char a, b;
```

```
int c;
```

```
a='J';
```

```
b='K';
```

```
c=a+b;
```

The ASCII value of J is 74, and the ASCII value of K is 75. Hence, the value stored in 'c' is 149.

Legal:  $c=a*b$ ; Illegal:  $c=a.b$ ; Illegal:  $c=a(b+d)$ ;

Exponent Function: `pow( )`

The mathematical expression  $5^3$  can be represented in C using the `pow( )` function as shown below. `a = pow(5, 3);`

To use the `pow( )` function, we need to include the math library at the beginning of the program.

What is the output of this program? `#include<stdio.h> #include<math.h>`

```
void main ( )
```

```
{  
float a=2.0, b=3.0, c; c=pow(a, b); printf("%f", c);  
}
```

Output:\_\_\_\_\_

## Integer and Float Conversions

If in an arithmetic operation,

- ➤   •all the values are integers, the result is also an integer.
- ➤   •all the values are float the result is also a float.
- ➤   •even if one value is a float, the result is a float. In these operations, the integer is converted to float and then the operation is executed. Examples:
  - 5+2 gives integer output
- ➤   5.0+5.2 gives float output
- ➤   4-2.2 gives float output

- 4.2-3 gives float output

•

## Type conversion in assignments

The type of the value of the expression is same as the type of the variable to the left side of the assignment operator.

Image 2 Execute these snippets and write down the outputs:

Program Output Program Output

<pre>#include&lt;stdio.h&gt; void main() {  int ratio; int num1=12,num2=5; ratio=num1/ num2; printf("The ratio is: %d", ratio);</pre>		<pre>#include&lt;stdio.h&gt; void main() {  float ratio; int num1=12,num2=5; ratio=num2/ num1; printf("The ratio is: %f", ratio);</pre>	
---	--	---	--

: \* / %

: + -

: = (Assignment operator)

<pre>#include&lt;stdio.h&gt; void main() {  int ratio; float num1=12.5; int num2=5; ratio=num1/num2; printf("The ratio is: %d", ratio);  }</pre>		<pre>#include&lt;stdio.h&gt; void main() {  float ratio; float num1=12.5; int num2=5; ratio=num2/num1; printf("The ratio is: %f", ratio);  }</pre>	
<pre>#include&lt;stdio.h&gt; void main() {  int ratio; int num1=12; float num2=5.0; ratio=num1/num2; printf("The ratio is: %d", ratio);  }</pre>		<pre>#include&lt;stdio.h&gt; void main() {  float ratio; int num1=12; float num2=5.0; ratio=num2/num1; printf("The ratio is: %f", ratio);  }</pre>	

<pre>#include&lt;stdio.h&gt; void main() { int ratio; float num1=12.5; float num2=5.0; ratio=num1/num2; printf("The ratio is: %d", ratio); }</pre>		<pre>#include&lt;stdio.h&gt; void main() { float ratio; float num1=12.5; float num2=5.0; ratio=num2/num1; printf("The ratio is: %f", ratio); }</pre>	
<pre>#include&lt;stdio.h&gt; void main() { int ratio; float num1=5.0; float num2=10.0; ratio=num1/ num2; printf("The ratio is: %d", ratio); }</pre>		<pre>#include&lt;stdio.h&gt; void main() { float ratio; float num1=5.0; float num2=10.0; ratio=num2/ num1; printf("The ratio is: %f", ratio); }</pre>	

## Hierarchy of operations

If the expression contains brackets ( ), the operations inside the innermost bracket will be executed first, then the operation inside the second innermost brackets will be executed and so on.

Priority among operators:

First priority

Second priority

Third priority

Note: The number of open brackets and close brackets must be equal in an expression.

What is the output for the given programs?

Programs Output Programs Output

---

<pre>#include&lt;stdio.h&gt; void main() {  int a; a=6.0/4+3*6.5-8/4; printf("%d" , a);  }</pre>		<pre>#include&lt;stdio.h&gt; void main() {  int a; a=6/4+3*6.5-8/4;  printf("%d" , a);  }</pre>	
--	--	---	--

Write a program to execute these arithmetic expressions. Output should be stored in k which is an integer. Expression Values Program

<pre>#include&lt;stdio.h&gt; void main() {  float a; a=3/2+3*4+6.8/1.7-2.2/1.1; printf("%f" , a);  }</pre>		<pre>#include&lt;stdio.h&gt; void main() {  float a; a=3.0/2+3*4+6.8/1.7-2.2/1.1; printf("%f" , a);  }</pre>	
--	--	--	--

$(a/b) \times c + d \quad d-a$	a=4 b=2 c=3 d=8	
$4a^2 - 2b + (c \times d)$	a=3 b=2 c=4 d=1	
$(2xy/x+y) - (x-y/xy)$	x=4 y=8	
$(a+b)^3 - 2ab + a^2 + b^2$	a=6 b=8	

## Objective

if and else are two of the most frequently used conditionals in C/C++, and they enable you to execute zero or one conditional statement among many such dependent conditional statements. We use them in the following ways:

1. if: This executes the body of bracketed code starting with if evaluates to true.  

```
if (condition) {
    statement1;
    ...
}
```
- 2.
- 3.
- 4.
- 5.

6. **if - else:** This executes the body of bracketed code starting with if evaluates to true, or it executes the body of code starting with if evaluates to false. Note that only one of the bracketed code sections will ever be executed.

```
if (condition) {  
7.     statement1;  
8.     ...  
9. }  
10. else {  
11.     statement2;  
12.     ...  
13. }  
14.
```

15. **if - else if - else:** In this structure, dependent statements are chained together and the for each statement is only checked if all prior conditions in the chain are evaluated to false. Once a evaluates to true, the bracketed code associated with that statement is executed and the program then skips to the end of the chain of statements and continues executing. If each in the chain evaluates to false, then the body of bracketed code in the else block at the end is executed.

```
if(first condition) {  
16.     ...  
17. }  
18. else if(second condition) {  
19.     ...  
20. }  
21. .  
22. .  
23. .  
24. else if((n-1)'th condition) {  
25.     ....  
26. }  
27. else {  
28.     ...  
29. }  
30.
```

## Task

Given a positive integer denoting , do the following:

- If , print the lowercase English word corresponding to the number (e.g., one for , two for , etc.).



- If , print Greater than 9.

### Input Format

The first line contains a single integer, .

### Constraints

- 

### Output Format

If , then print the lowercase English word corresponding to the number (e.g., one for , two for , etc.); otherwise, print Greater than 9 instead.

### Sample Input

5

### Sample Output

five

### Sample Input #01

8

### Sample Output #01

eight

### Sample Input #02

44

### Sample Output #02

Greater than 9

```
while (true) {
    char* cursor = data + data_length;
    char* line = fgets(cursor, alloc_length - data_length, stdin);

    if (!line) { break; }

    data_length += strlen(cursor);

    if (data_length < alloc_length - 1 || data[data_length -
1] == '\n') { break; }

    size_t new_length = alloc_length << 1;
    data = realloc(data, new_length);

    if (!data) { break; }

    alloc_length = new_length;
}
```

```
if (data[data_length - 1] == '\n') {  
    data[data_length - 1] = '\0';  
}
```

```
data = realloc(data, data_length);
```

```
return data;  
}
```

```
printf("nine");  
else  
printf("Greater than 9");  
return 0;  
}
```

```
char* readline() {  
    size_t alloc_length = 1024;  
    size_t data_length = 0;  
    char* data = malloc(alloc_length);  
    ...    data[data_length - 1] = '\0';  
}
```

```
data = realloc(data, data_length);
```

```
return data;  
}
```

## 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 .

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 variables: two of type int and two of type float.
2. Read lines of input from stdin (according to the sequence given in the 'Input Format' section below) and initialize your 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.

### Input Format

The first line contains two integers.

The second line contains two floating point numbers.

### Constraints

- integer variables
- float variables

### Output Format

Print the sum and difference of both integers separated by a space on the first line, and the sum and difference of both float (scaled to decimal place) separated by a space on the second line.

### Sample Input

```
10 4
4.0 2.0
```

### Sample Output

```
14 6
6.0 2.0
```

### Explanation

When we sum the integers `10` and `4`, we get the integer `14`. When we subtract the second number from the first number `10`, we get `6` as their difference.

When we sum the floating-point numbers `4.0` and `2.0`, we get `6.0`. When we subtract the second number from the first number `4.0`, we get `2.0` as their difference.

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

int main()
{
    int m,n;
    float a,b;
    scanf("%d %d",&m,&n);
    scanf("%f %f",&a,&b);
    printf("%d %d\n",m+n,m-n);
    printf("%.1f %.1f",a+b,a-b);
    return 0;
}
```