Important message on plagiarism

The single most important point for you to realize before the beginning of your studies at ShanghaiTech is the meaning of “plagiarism”:

Plagiarism is the practice of taking someone else's work or ideas and passing them off as one's own. It is the misrepresentation of the work of another as your own. It is academic theft; a serious infraction of a University honor code, and the latter is your responsibility to uphold. Instances of plagiarism or any other cheating will be reported to the university leadership, and will have serious consequences. Avoiding any form of plagiarism is in your own interest. If you plagiarize and it is unveiled at a later stage only, it will not only reflect badly on the university, but also on your image/career opportunities.

Plagiarism is academic misconduct, and we take it very serious at ShanghaiTech. In the past we have had lots of problems related to plagiarism especially with newly arriving students, so it is important to get this right upfront:

You may…
• … discuss with your peers about course material.
• … discuss generally about the programming language, some features, or abstract lines of code. As long as it is not directly related to any homework, but formulated in a general, abstract way, such discussion is acceptable.
• … share test cases with each other.
• … help each other with setting up the development environment etc.

You may not …
• … read, possess, copy or submit the solution code of anyone else (including people outside this course or university)!
• … receive direct help from someone else (i.e. a direct communication of some lines of code, no matter if it is visual, verbal, or written)!
• … give direct help to someone else. Helping one of your peers by letting him read your code or communicating even just part of the solution in written or in verbal form will have equal consequences.
• … gain access to another one’s account, no matter if with or without permission.
• … give your account access to another student. It is your responsibility to keep your account safe, always log out, and choose a safe password. Do not just share access to your computer with other students without prior lock--out and disabling of automatic login functionality. Do not just leave your computer on without a lock even if it is just for the sake of a 5--minute break.
• … work in teams. You may meet to discuss generally about the material, but any work on the homework is to be done individually and in privacy. Remember, you may not allow anyone to even just read your source code.

With the Internet, "paste", and "share" are easy operations. Don’t think that it is easy to hide and that we will not find you, we have just as easy to use, fully automatic and intelligent tools that will identify any potential cases of plagiarism. And do not think that being the original author will make any difference. Sharing an original solution with others is just as unethical as using someone else’s work.

CS100 Homework 2 (Spring, 2021)

In this homework, you are required to do more practices with C programming language, starting to get familiar with the concepts of IO, control flow and function, etc. This homework tries to pave the way for your later homework and projects. Have fun and enjoy coding. Practice, practice and practice.

Submission deadline:

2021-03-21 23:59:59

Late submission will open for 24 hours when the deadline is reached, with -50% point deduction.

Problem 1. RGB Color Map (30 points)

The RGB color model is an additive color model in which **red**, **green**, and **blue** light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors, Red, Green, and Blue. Generally, we can use 3 numbers between 0 and 255 to specify some color. For example, (255, 0, 0) represents red, (0, 255, 0) represents green, and (0, 0, 255) represents blue. To save storage space, these three numbers are usually merged into a single integer. If written in hexadecimal, it's easier to extract the RGB value it represents. For example, the color **0x66ccff**:

Of the six hexadecimal digits, the first two digits represent value for **red**, the following two digits for **green**, and the last two for **blue**. Therefore, R = 0x66, G = 0xcc and B = 0xff.

In this problem, you need to implement a color map which maps a number to an RGB value using linear interpolation. That is, you are given the maximum and minimum of a series of numbers, where the minimum maps to **black** (0x000000), and the maximum maps to `maxValColor`. For any number between max and min, we can calculate the color it maps to by linear interpolation (https://en.wikipedia.org/wiki/Linear_interpolation).
So, your work is just to finish the function `ColorMap`. Its prototype has been given:

```
int ColorMap(float minVal, float maxVal, int maxValColor, float val)
```

This function takes 4 parameters, `minVal` is the minimum of some series, `maxVal` is the maximum of this series, `maxValColor` is the color that `maxVal` maps to, and `val` is the number you want to query. The return value of this function is the RGB value of `val` maps to.

For example, if we call

```
C = ColorMap(50, 150, 0x39c5bb, 80);
```

Then we will get C = 0x113b38. Because in this case, 150 maps to 0x39c5bb (the Miku's color!), 50 maps to 0x000000, and we want to find the color that 80 maps to. By linear interpolation,

$$R_{80} = \frac{\text{val - minVal}}{\text{maxVal} - \text{minVal}} \times (R_{max} - R_{min}) + R_{min} = \frac{80 - 50}{150 - 50} \times 0x39 = 17.1 = 0x11$$

Similarly, we can calculate

$$G_{80} = \frac{80-50}{150-50} \times 0xc5 = 59.1 = 0x3b$$

$$B_{80} = \frac{80-50}{150-50} \times 0xbb = 56.1 = 0x38$$

Notice that if the RGB value you calculated is not an integer, you need to **round it down** to the nearest integer, e.g., if R=59.8, you should round it to 59 (0x3b).

Note: The main function is already provided for you. You don't need to consider too much on inputs and outputs. Your job is just to complete the required functions.

Input description:

The input contains 4 numbers, the first number is **`minVal`**, the second is **`maxVal`**, the third one is **`maxValColor`**, and the final one **`val`**. It's guaranteed that

$$0 < minVal < val < maxVal < FLOAT\_MAX$$

and **`maxValColor`** is a valid RGB color (between 0x000000 and 0xffffff).

Output description:

The output contains only one number, which is the number **`val`** maps to.

A correct program runs like this: (red indicates input and black indicates your output)

```
50 150 0x39c5bb 80
0x113b38
```

Problem 2. Drawing Shapes (30 points)

In this problem, we would like you to do some ASCII drawing. In particular, you will need to print the following three shapes to the console: squares, diamonds, and (horizontal) hexagons.

```
******
*    *
*    *
*    *
*    *
******
```

a square (size = 6)

```
   *
  * *
 *   *
*     *
 *   *
  * *
   *
```

a diamond (size = 4)

```
  ***
 *   *
*     *
 *   *
  ***
```

a hexagon (size = 3)

We have provided you with prototypes of all three functions, respectively `DrawSquare`, `DrawDiamond`, and `DrawHexagon`. All three functions have `void` type, meaning that they will not return any value. Also, they all take one single parameter, `int size`, which more particularly means the length of an edge.

To better show how to draw the "correct" shapes, here are some details:

1. You need to draw the outline of a shape with asterisk, the character `'*'`, or call it "star" if you like. The inner area of any shape, and also the empty spaces to the left of a diamond or a hexagon, are filled with the empty space character `' '`.
2. In one line, after the last (right-most) `'*'` is drawn, put a new line character `'\n'` immediately and do not fill any empty spaces to the right.
3. No empty lines between lines, and no empty spaces between horizontally adjacent `'*'` stars. This makes your square look more like a rectangle, and it's okay.
4. The left-most points in any shape should occupy the left-most column. You should not have any empty space on the left side of these points.
5. In a diamond or a hexagon, any two diagonally adjacent `'*'` stars have both horizontal and vertical distances of one character.

```
...*\n
..*.*\n
.*...*\n
*.....*\n
.*...*\n
..*.*\n
...*\n
```

If these rules confuse you, the illustration is also quite explaining. To make you clear, spaces are replaced with red dots `'.'`.

Hint: The function `putchar()` is useful in printing out a single character.

Note: The main function is already provided for you. You don't need to consider too much on inputs and outputs. Your job is just to complete the required functions.

Input description:

The input contains two numbers in one line, separated by a space.

The first number will be either 1, 2, or 3, specifying the type of shape (1 for square, 2 for diamond, 3 for hexagon).

The second number is **size**. It is guaranteed that $2 \le \textbf{size} \le 10$.

Output description:

You should draw a specified shape with correct size.

A correct program runs like this: (red indicates input and black indicates your output)

```
1 6
******
*    *
*    *
*    *
*    *
******
```

```
2 4
   *
  * *
 *   *
*     *
 *   *
  * *
   *
```

Problem 3. Finding the Second Maximum and Minimum (40 points)

In this problem, you need to write a function `FindSecondMaxAndMin` to find the second maximum and minimum of a series of integers, and pass your results by pointers.

```
void FindSecondMaxAndMin(int* secondMax, int* secondMin);
```

Your function needs to receive input numbers from the console, find the second maximum and minimum, and finally save them to the pointers `secondMax` and `secondMin`.

For example, if the input is [2, 3, 4, 7, 6, 10, 9, 8], then we will get `(*secondMax) = 9` and `(*secondMin) = 3` after calling your function.

Note: The main function is already provided for you. You don't need to consider too much on the output format. Your job is just to handle the input, process these numbers, and pass the results out by pointers.

Input description:

The input contains two lines. The first line contains only one integer **`n`**, indicates the number of integers you need to process. The second line contains **`n`** integers separated by a single whitespace, which is the series mentioned before. It's guaranteed that **`3≤n≤1000`** for all testcases, and all numbers are **different** integers between **`-10000`** and **`10000`**.

Output description:

Two numbers, the first maximum and the second minimum.

A correct program runs like this: (red indicates input and black indicates your output)

```
8
2 3 4 7 6 10 9 8
9 3
```

```
3
100 200 300
200 200
```