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, 2022)

Deadline: 2022-03-10 23:59:59

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

Problem 1. Find the Second Maximum and Minimum

In this problem, you need to find the second maximum and minimum number in an array of integers. Since this function generates two results, you will need to pass them by pointers. Please implement the following function:

```
void FindSecondMaxAndMin(int numbers[], int size, int* secondMax, int* secondMin);
```

The parameter `numbers` is an `int` array whose size is the second parameter `size`. `secondMax` and `secondMin` are pointers to valid `int` addresses, where you need to store the second maximum and minimum number you found. It is guaranteed that $3 \leq$ `size` ≤ 1000, and all numbers in the array are **different** integers between -10000 and 10000.

For example, if `numbers = {2, 3, 4, 7, 6, 10, 9, 8}` and `size = 8`, you are supposed to get `(*secondMax) = 9` and `(*secondMin) = 3` after calling your function.

- Although not a requirement, we encourage you to solve this problem efficiently. Can you come up with a solution that visits each number in the array only once?

Submission Guidelines:

This problem does not have an input/output part. It only asks you to write a function, and judges the correctness of your implementation.

When you submit your code, your `main` function will be replaced by one on OJ, and the functions you implemented will be directly called with parameters. Therefore, you **MUST NOT** modify the function names, add or remove parameters, or put your code in `main` function. Otherwise, you will **NOT** receive any scores.

We have provided a `main` function for you, please see the code templates.

Problem 2. Vertical Calculation

Vertical calculation is probably the method that you learned from Maths class during kindergarten. It tells you how to perform elementary arithmetic. In this problem, you need to visualize the vertical calculation for addition, subtraction and multiplication. An example of vertical form addition is shown in the following figure.

Addition

The components of a vertical addition are 2 input numbers, 1 output number, the plus sign and a horizontal line. To make it more pretty, we have the following rules for vertical addition:

- You need to output 4 lines and each line should have **the same length**.
- The first two lines are the numbers to be added and the last line is the number after addition.
- The third line is the horizontal line where each position should be the character '-'.
- All numbers should be **right-aligned**.
- All numbers should start at least from **the third position** of each line. In other words, the first two positions of a line with number should be left empty.
- The **plus sign** '+' should appear at the first position of the second line.

For example, $12 + 34 = 46$ and $1 + 99 = 100$ in vertical form are shown in figure 1:

		1	2	'\n'
+		3	4	'\n'
-	-	-	-	'\n'
		4	6	'\n'

				1	'\n'
+			9	9	'\n'
-	-	-	-	-	'\n'
		1	0	0	'\n'

Figure 1: **Left**: $12 + 34 = 46$ **Right**: $1 + 99 = 100$

In the figure above, each box stands for a place in a line which contains only one character or digit. A box with no characters or digits stands for a space ' '. Each character or digit has the same width. '\n' stands for a newline.

Do not forget to add appropriate spaces before the numbers.

Subtraction

Subtraction in vertical form is quite similar to addition, except that:

- The first two lines are the numbers to be subtracted and the last line is the number after subtraction.
- The minus sign '-' should appear at the first position of the second line.
- The minus sign '-' of **negative number** is treated as part of the number, which should also start at least from the third position of the line.

For example, $12 - 34 = -22$ and $100 - 99 = 1$ in vertical form are shown in figure 2:

			1	2	'\n'
-			3	4	'\n'
-	-	-	-	-	'\n'
		-	2	2	'\n'

		1	0	0	'\n'
-			9	9	'\n'
-	-	-	-	-	'\n'
				1	'\n'

Figure 2: **Left**: $12 - 34 = -22$ **Right**:$100 - 99 = 1$

Multiplication

Additional rules are needed to output the vertical form of multiplication:

- The first two lines are the numbers to be multiplied and the last line is the number after multiplication.
- The third and **the second to last line** are the horizontal lines where each position should be the character '-'.
- The multiplication sign 'x' should appear at the first position of the second line.
- The multiplication starts with the rightmost digit in the second line. You need to multiply this digit by each digit of the number in the first line from right to left. After each multiplication, the individual answers are written below the horizontal line working right to left. Then the multiplication continues by shifting to the next digit left of the number in the second line. The important thing is to start the placement of the answers on **a new row** and **one place to the left**.
- **You don't need to output the line if the intermediate result is 0.**

For example, $11 \times 101 = 1111$ in vertical form is shown in figure 3:

Note that the line whose result is 0 has been ignored.

				1	1	'\n'
x			1	0	1	'\n'
-	-	-	-	-	-	'\n'
				1	1	'\n'
		1	1	'\n'		
-	-	-	-	-	-	'\n'
		1	1	1	1	'\n'

Figure 3: $11 \times 101 = 1111$

We have provided you with prototypes of all three functions, respectively `add`, `subtract` and `multiply`. You need to implement the output of vertical calculation in each function.

```
void add(int a, int b);
void subtract(int a, int b);
void multiply(int a, int b);
```

It is guaranteed that:

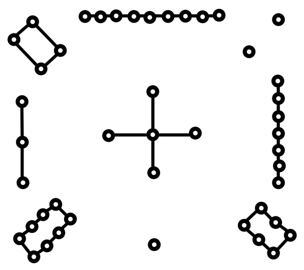
- Two **positive** integers will be given.
- `int` is sufficient to store the results. No need to worry about overflow or underflow.

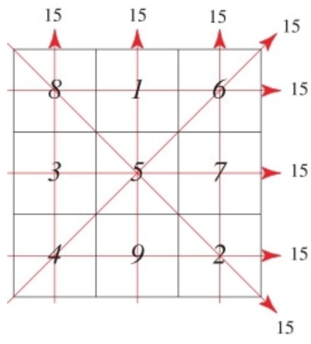
Submission Guidelines:

When you submit your code, your `main` function will be replaced by one on OJ, and the function you implemented will be directly called with parameters. Therefore, you **MUST NOT** modify the function names, add or remove parameters, or put your code in `main` function. Otherwise, you will **NOT** receive any scores.

We have provided a `main` function for you, please see the code templates.

Problem 3. Magic Square

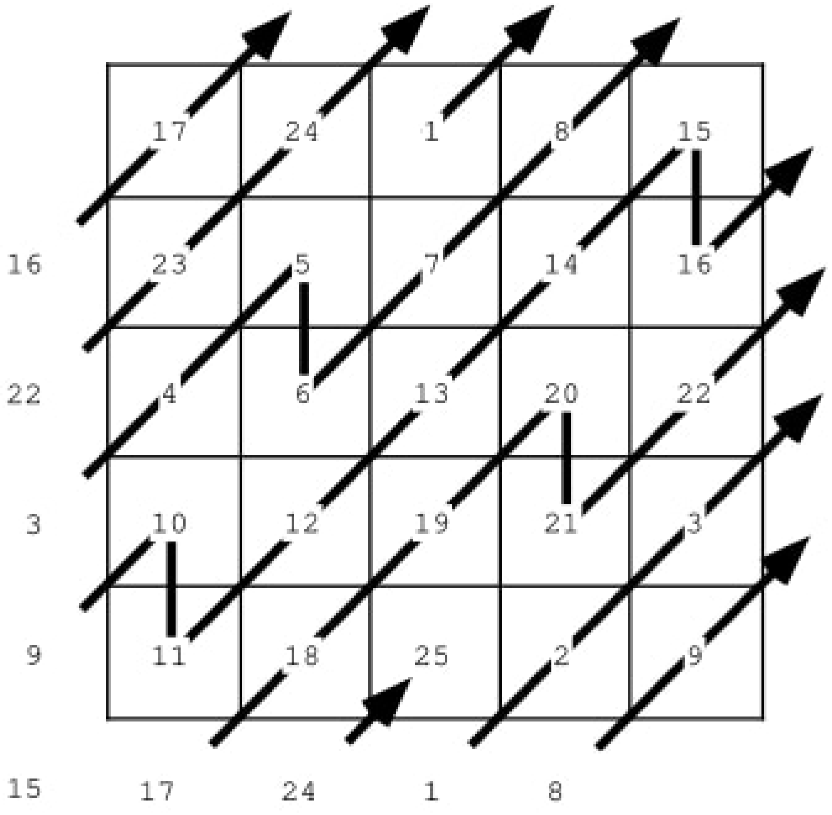




A magic square of order n is a square array of numbers consisting of distinct positive integers $1, 2, ..., n^2$ arranged such that the sum of the n numbers in any horizontal, vertical, or main diagonal line is always the same.

There is a straightforward way to construct a magic square of order n when n is an odd number:

- To begin, place the number 1 at the center of the top row.
- Place each subsequent number at the next top-right square (both one unit above and to the right).
- If the next top-right square is off-boundary, the place goes to the other side of the boundary. (In other words, the boundaries are wrapped-around.)
- If the square to place a new number is already filled, the new number is instead placed below the previous one.



Now let's generate magic squares by this method. Please implement the following function:

```c
int** MagicSquare(int n);
```

This function should return a **magic square of order n** by first allocating memory of **an** $n \times n$ **2-dimensional `int` array** with the function `malloc`, and then filling that array by the method given above. The 2-dimensional array should be indexed **rows first and then columns**, so that the index `[0][0]` stores the number at the top-left corner, `[0][n-1]` the top-right corner, and `[n-1][n-1]` the bottom-right corner. It is guaranteed that n is a positive odd number and that $n < 100$.

- Please be aware that your function will **FAIL** to return correct results if you declare your array as a local variable rather than by using `malloc`, as local variables are destroyed when exiting the function (exiting its local scope).

Since the memory for our magic squares is allocated using `malloc`, we must free it after use, or **memory leak** will happen. Please implement the following function:

```c
void FreeMagicSquare(int** magicSquare, int n);
```

This function should free **ALL** memory allocated for the $n \times n$ 2-dimensional array `magicSquare` using the function `free`.

- If your implementation does not use the parameter `n`, it is very probably incorrect!

Submission Guidelines:

This problem does not have an input/output part. It only asks you to write functions, and judges the correctness of your implementation.

When you submit your code, your `main` function will be replaced by one on OJ, and the functions you implemented will be directly called with parameters. Therefore, you **MUST NOT** modify the function names, add or remove parameters, or put your code in `main` function. Otherwise, you will **NOT** receive any scores.

We have provided a `main` function for you, please see the code templates.