1. Write a program that produce the following output:

```
999998888877777666665555544444333332222211111100000
999998888877777666665555544444333332222211111100000
999998888877777666665555544444333332222211111100000
999998888877777666665555544444333332222211111100000
999998888877777666665555544444333332222211111100000
```

2. Write a method that takes in two integer parameters, a *min* and a *max*, and prints the numbers in the range from *min* to *max* inclusive in a square pattern. The square pattern is easier to understand by example than by explanation, so take a look at the sample method calls and their resulting console output in the table below.

Each line of the square consists of a circular sequence of increasing integers between *min* and *max*. Each line prints a different permutation of this sequence. The first line begins with *min*, the second line begins with *min* + 1, and so on. When the sequence in any line reaches *max*, it wraps around back to *min*.

You may assume the caller of the method will pass a min and a max parameter such that min is less than or equal to max.

Call	`print(1, 5);`	`print(3, 9);`	`print(0, 3);`	`print(5, 5);`
Output	12345 23451 34512 45123 51234	3456789 4567893 5678934 6789345 7893456 8934567 9345678	0123 1230 2301 3012	5

3. Write a program to read an integer number (n>0). Write flowing methods:
- Checking whether the input number is a prime number or not?
- Print out prime submultiples of number **n**
- Generate the first **100** prime numbers.

4. Write a method to read a number (N>0) from keyboard and compute sum of number's numerals.
Example: Suppose N = 25364, N include 5 numerals: 2, 5, 3, 6, 4. The sum of number's numerals is: 20

5. Write a method to enter a integer number (n>0) and reserve that number.
Guide: **The reverse number** is an integer number source from the orgine one but the order of digits are reversed.

Example: The reverse number of:
12345 is 54321
2008 is 8002
1990 is 991
300400 is 4003

6. Write a method named **`lastIndexOf`** that accepts an array of integers and an integer value as its parameters and returns the last index at which the value occurs in the array. The method should return -1 if the value is not found. For example, in the list containing {74, 85, 102, 99, 101, 85, 56}, the last index of the value 85 is 5.

7. Write a program that asks the user to enter numbers, then prints the smallest and largest of all the numbers typed in by the user. You may assume the user enters a valid number greater than 0 for the number of numbers to read. Here is an example dialogue:

```
How many numbers do you want to enter? 4
Number 1: 5
Number 2: 11
Number 3: -2
Number 4: 3
Smallest = -2
Largest = 11
```

8. Write a program that prompts the user for many integers and print the total even sum and maximum of the even numbers. You may assume that the user types at least one non-negative even integer.

```
how many integers? 4
next integer? 2
next integer? 9
next integer? 18
next integer? 4
even sum = 24
even max = 18
```

9. Write a method called **`median`** that accepts an array of integers as its argument and returns the median of the numbers in the array. The median is the number that will appear in the middle if you arrange the elements in order. Assume that the array is of odd size (so that one sole element constitutes the median) and that the numbers in the array are between 0 and 99 inclusive.

For example, the median of `{5, 2, 4, 17, 55, 4, 3, 26, 18, 2, 17}` is `5`, and the median of `{42, 37, 1, 97, 1, 2, 7, 42, 3, 25, 89, 15, 10, 29, 27}` is `25`.

10. Write a method named `minGap` that accepts an integer array as a parameter and returns the minimum 'gap' between adjacent values in the array. The gap between two adjacent values in a array is defined as the second value minus the first value. For example, suppose a variable called `array` is an array of integers that stores the following sequence of values.

int[] array = {1,3,6, 7, 12}

The first gap is 2 (3 - 1), the second gap is 3 (6 - 3), the third gap is 1 (7 - 6) and the fourth gap is 5 (12 - 7). Thus, the call of `minGap(array)` should return `1` because that is the smallest gap in the array. Notice that the minimum gap could be a negative number. For example, if array stores the following sequence of values:

{3, 5, 11, 4, 8}

The gaps would be computed as 2 (5 - 3), 6 (11 - 5), -7 (4 - 11), and 4 (8 - 4). Of these values, -7 is the smallest, so it would be returned.

This gap information can be helpful for determining other properties of the array. For example, if the minimum gap is greater than or equal to 0, then you know the array is in sorted (nondecreasing) order. If the gap is greater than 0, then you know the array is both sorted and unique (strictly increasing).

If you are passed an array with fewer than 2 elements, you should return 0.