# More Exercise: Data Types and Variables

You can check your solutions in Judge



## 1. Data Type Finder

You will receive input until you receive "**END**". Find what **data type** is the input. Possible data types are:

- Integer
- Floating point
- Characters
- Boolean
- Strings

Print the result in the following format: "**`{input}`** `is` **`{data type}`** `type`".

### Examples

| Input | Output |
|---|---|
| 5<br>2.5<br>true<br>END | 5 is integer type<br>2.5 is floating point type<br>true is boolean type |
| a<br>asd<br>-5<br>END | a is character type<br>asd is string type<br>-5 is integer type |

## 2. From Left to the Right

You will receive a number that represents how many lines we will get as input. On the next **N** lines, you will receive a string with 2 numbers, separated by a single space. You need to compare them. If the left number is greater than the right number, you need to print the sum of all digits in the left number, otherwise, print the sum of all digits in the right number.

### Examples

| Input | Output |
|---|---|
| 2<br>1000 2000 | 2<br>2 |

| 2000 1000 | |
|---|---|
| 4<br>123456 2147483647<br>5000000 -500000<br>97766554 97766554<br>9999999999 8888888888 | 46<br>5<br>49<br>90 |

## 3. Floating Equality

Write a program that safely compares floating-point numbers (double) with precision **`eps = 0.000001`**. Note that we cannot directly compare two floating-point numbers **a** and **b** by **`a == b`**, because of the nature of the floating-point arithmetic. Therefore, we assume two numbers are equal if they are more close to each other than some fixed constant eps.

You will receive two lines, each containing a floating-point number. Your task is to compare the values of the two numbers.

### Examples

| Number a | Number b | Equal (with precision eps=0.000001) | Explanation |
|---|---|---|---|
| 5.3 | 6.01 | False | The difference of 0.71 is too big (> eps) |
| 5.00000001 | 5.00000003 | True | The difference 0.00000002 < eps |
| 5.00000005 | 5.00000001 | True | The difference 0.00000004 < eps |
| -0.0000007 | 0.00000007 | True | The difference 0.00000077 < eps |
| -4.999999 | -4.999998 | False | Border case. The difference 0.0000001== eps. We consider the numbers are different. |
| 4.999999 | 4.999998 | False | Border case. The difference 0.0000001 == eps. We consider the numbers are different. |

## 4. Refactoring: Prime Checker

You are given a program that checks if numbers in a given range [**`2...N`**] are prime. For each number is printed "**`{number} -> {true or false}`**". The code, however, is not very well written. Your job is to modify it in a way that is easy to read and understand.

### Code

| Sample Code |
|---|

```
int ___Do___ = int.Parse(Console.ReadLine());
for (int takoa = 2; takoa <= ___Do___; takoa++)
{
   bool takovalie = true;
   for (int cepitel = 2; cepitel < takoa; cepitel++)
   {
       if (takoa % cepitel == 0)
       {
          takovalie = false;
          break;
       }
   }
Console.WriteLine("{0} -> {1}", takoa, takovalie);
}
```

### Examples

| Input | Output |
|---|---|
| 5 | 2 -> true<br>3 -> true<br>4 -> false<br>5 -> true |

## 5. Decrypting Messages

You will receive a **key** (**integer**) and **n** characters afterward. Add the key to each of the characters and append them to a **message**. At the end print the message, which you decrypted.

### Input

- On the **first line,** you will receive the **key.**
- On the **second line,** you will receive **n** – the number of **lines**, which will **follow.**
- On the next **n lines,** you will receive **lower** and **uppercase** characters from the **Latin** alphabet.

### Output

Print the **decrypted message**.

### Constraints

- The **key** will be in the interval **[0…20].**
- **n** will be in the interval **[1…20].**
- The **characters** will always be **upper** or **lower**-case letters from the **English alphabet.**
- You will receive **one letter** per **line.**

### Examples

| Input | Output |
|---|---|
| 3<br>7<br>P<br>l<br>c<br>q<br>R<br>k<br>f | SoftUni |

| Input | Output |
|---|---|
| 1<br>7<br>C<br>d<br>b<br>q<br>x<br>o<br>s | Decrypt |

## 6. Balanced Brackets

You will receive **n** lines. On **those lines**, you will receive **one** of the following:

- Opening bracket – '**(**',
- Closing bracket – '**)**' or
- **Random string**

Your task is to find out if the **brackets** are **balanced**. That means after every **opening** bracket should follow an **closing** one. Nested parentheses are **not valid** and if **two consecutive opening brackets** exist, the expression should be marked as **unbalanced**.

## Input

- On the **first line,** you will receive **n** – the number of lines, which will follow.
- On the next **n** lines, you will receive '**(**', '**)**' or **another** string.

## Output

You have to print "**BALANCED**", if the parentheses are balanced and "**UNBALANCED**" otherwise.

## Constraints

- **n** will be in the interval **[1…20].**
- The length of the stings will be between **[1…100]** characters.

## Examples

| Input | Output |
|---|---|
| 8<br>(<br>5 +<br>10<br>)<br>* 2 +<br>(<br>5<br>)<br>-12 | BALANCED |

| Input | Output |
|---|---|
| 6<br>12 *<br>)<br>10 + 2<br>-<br>(<br>5 + 10<br>) | UNBALANCED |