Mid-exam Imperative Programming
Sept. 26 2019, 15:00-18:00h

- You can solve the problems in any order. Solutions must be submitted to the automated judgement system Themis. For each problem, Themis will test ten different inputs, and check whether the outputs are correct.
- Grading: you get one grade point for free. The remaining nine points are based solely on the judgment given by Themis. The first problem is worth one grade point. The remaining four problems are worth two grade points each, of which you score the full two points if you passed the complete test set of the problem (i.e. 10 test cases), or one grade point if you passed at least 5 (out of 10) test cases.
- Inefficient programs may be rejected by Themis. In such cases, the error will be 'time limit exceeded'. The time limit for each problem is two seconds.
- The number of submissions to Themis is unlimited. No points are subtracted for multiple submissions.
- There will be no assessment of programming style. However, accepted solutions are checked manually for cheating: for example, precomputed answers will not be accepted, even though Themis accepts them.
- Note the hints that Themis gives when your program fails a test.
- Needless to say: you are not allowed to work together. If plagiarism is detected, both parties (supplier of the code and the person that sends in copied code) will be excluded from any further participation in the course.
- You are not allowed to use email, phones, tablets, calculators, etc. There is a calculator available on the exam computers (see icon on the desktop). You are allowed to consult the ANSI C book and a dictionary. You are not allowed to use a printed copy of the reader or the lecture slides, however they are available digitally (as a pdf) in Themis. You are allowed to access your own submissions previously made to Themis.
- For each problem, the first three test cases (input files) are available on Themis. These input files, and the corresponding output files, are called `1.in`, `2.in`, `3.in`, `1.out`, `2.out` and `3.out`. These files can be used to test whether the output of your program matches the requested layout, so that there can be no misunderstanding about the layout and spaces in the output.
- **If you fail to pass a problem for a specific test case, then you are advised not to lose much time on debugging your program, and continue with another problem. In the last hour of the midterm, all input files will be made visible in Themis (not the output files).**

Problem 1: Decimal Numbers

The number 123 is decimal notation for the number $1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$.

Write a program that reads from the input a positive integer n (where $0 < n \leq 1000000$), and outputs its decimal expansion according to the format in the following examples. Note that spaces enclose the symbol +. Moreover, note that terms of the form `0*10^n` are not printed (see example 3).

Example 1:

input:

```
123
```

output:

```
123=1*10^2 + 2*10^1 + 3*10^0
```

Example 2:

input:

```
42
```

output:

```
42=4*10^1 + 2*10^0
```

Example 3:

input:

```
501
```

output:

```
501=5*10^2 + 1*10^0
```

Problem 2: Balanced Numbers

We call a non-negative integer a *balanced number* if the sum of its even digits equals the sum of its odd digits. For example, the number 8751326 is a balanced number, because $8 + 2 + 6 = 7 + 5 + 1 + 3$. The number 12345 is clearly not balanced, because $2 + 4 \neq 1 + 3 + 5$.

Write a program that reads from the input a non-negative integer n (where n fits in a standard `int`), and outputs `YES` if n is balanced, otherwise it should output `NO`.

Example 1:

input:

```
8751326
```

output:

```
YES
```

Example 2:

input:

```
12345
```

output:

```
NO
```

Example 3:

input:

```
4251
```

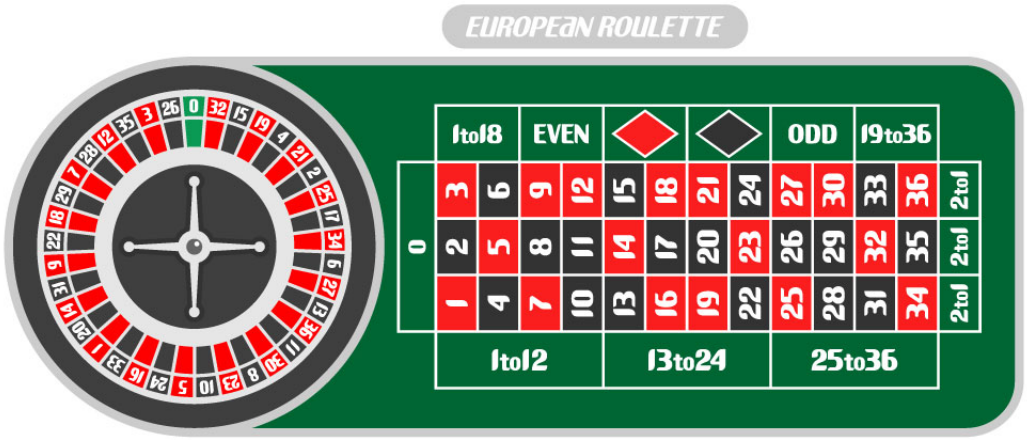
output:

```
YES
```

Problem 3: Martingale Red

Roulette is a casino game named after the French word meaning 'little wheel'. In each play of roulette, a roulette wheel spins in one direction and a ball in the opposite direction. The ball gradually loses momentum, and eventually falls into one of several coloured and numbered pockets along the edge of the wheel.

Players can make several bets before spinning the wheel. One of those bets is based on the colour of the number where the ball lands. This colour can be red or black. If a player bets some amount of money, say n euros, on the colour red and the ball lands in the pocket of a red coloured number, then the player receives $2n$ euros. In other words he wins n euros. If the ball lands in a black coloured pocket, or in the pocket with the number zero (which is the only pocket coloured green), then the player loses the bet (and thus the n euros). The colours of the numbers can be found in the following figure.



The *Martingale Red System* is a roulette strategy. In this strategy the player always bets on the colour red. After a loss the player doubles the next bet. After a win, the player starts over with the initial bet amount. This way, in theory, after each winning bet the player always has a win of one unit.

For example, let the initial bet be 1 euro, and there are 4 consecutive draws of a black number. Hence, the player has lost $1 + 2 + 4 + 8 = 15$ euros. Then, according to the strategy, in the next round the player should bet 16 euros. If the player wins the next round, then the total win is 1 euro (a loss of 15 euros followed by a win of 16 euros).

Of course, the system is not flawless. The problem is that doubling the bet on each loss requires an unlimited amount of money. In practice, players have a finite amount of money. We say that a player is *bust* if he has not enough money to make the next bet. So, if the next bet would be 32 euros, while the player only has 10 euros, then we still consider the player bust.

Write a program that reads from the input a starting amount of money (the budget). Next is a series of draws of numbers (spins of the wheel) terminated by the number -1 (note that valid draws are numbers in the range 0 upto 36). Assume that a player plays the Martingale Red system with the initial bet of 1 euro. The output of your program should be the amount of money that the player has after the series of spins, or `BUST` if the player went bust at some point in the game.

Example 1:

input:
```
10
1 36 -1
```
output:
```
12
```

Example 2:

input:
```
10
2 17 -1
```
output:
```
7
```

Example 3:

input:
```
10
2 17 35 -1
```
output:
```
BUST
```

Problem 4: Armstrong Numbers

A positive integer n is called an *Armstrong number* if the sum of each of its digits raised to the power of the number of digits equals n. For example, the number 153 is an Armstrong number because $1^3 + 5^3 + 3^3 = 153$. Here, each digit is raised to the power 3 because 153 has three digits. If we sum these up, we get the original number again!

The first 9 numbers with this property are, of course, the numbers 1,2,3,4,5,6,7,8, and 9. The 10th number with this property is 153. Write a program that reads from the input a positive integer `n` (where `n` < 25), and outputs the `n`th Armstrong number.

Example 1:

input:
```
1
```
output:
```
1
```

Example 2:

input:
```
9
```
output:
```
9
```

Example 3:

input:
```
10
```
output:
```
153
```

Problem 5: Primal Sums

A *prime* is an integer greater than 1 that has no positive divisors other than 1 and itself. The first 20 primes are:

$$2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71$$

A sum $a_0 + a_1 + \ldots + a_m = n$ (where the sum consists of at least two terms, i.e. $m > 1$) is called a *primal sum* if n is a prime number, and all numbers a_0, a_1, .., a_m are prime numbers as well. Moreover, the series a_0, a_1, .., a_m is increasing (so $a_i < a_{i+1}$ for all $i < m$), and all primes between (and including) a_0 and a_m are used in the sum (in other words, a consecutive series of primes).

As an example, the sum $5 + 7 + 11 + 13 + 17$ is a primal sum, because 5, 7, 11, 13, and 17 are primes, their sum is the prime 53, and all primes between 5 and 17 are included in the sum. The sum $2 + 3 + 11 = 17$ is not a primal sum, because the prime 5 is missing in the sum. The sum $3 + 5 = 8$ is also not a primal sum, since 8 is not a prime number.

Write a program that reads from the input two positive integer a and b (where $1 \leq a \leq b \leq 10000$), and outputs the number of integers n (where $a \leq n \leq b$) for which there exists a primal sum that adds up to n.

Example 1:

input:
```
1 10
```
output:
```
1
```

Example 2:

input:
```
1 100
```
output:
```
11
```

Example 3:

input:
```
1 370
```
output:
```
32
```