Mid-exam Imperative Programming
Sept. 29 2017, 14:00-17:00h

- 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 checks 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 one second.
- 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 a copy) 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 reader, the book, handouts of the lecture slides, a dictionary, and submissions previously made to Themis.
- For each problem, the first three test cases (input files) are available on Themis. These examples are the first three test cases of a test set. 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: Amicable pairs

A pair of positive integers is called an *amicable pair* if the sum of the proper divisors of each is equal to the other number. Note that a proper divisor of a positive integer n is a number d, where $1 \leq d < n$, that evenly divides n.

An example of a pair of amicable numbers is 220 and 284 because the sum of the proper divisors of 220 is $1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284$, while the sum of the proper divisors of 284 is $1 + 2 + 4 + 71 + 142 = 220$.

Write a program that accepts two positive integers on the input, and outputs whether the pair is an amicable pair or not. You may assume that both integers are less than $1000000000 = 10^9$.

Example 1:

input:
```
220 284
```
output:
```
YES
```

Example 2:

input:
```
63020 76084
```
output:
```
YES
```

Example 3:

input:
```
42 21
```
output:
```
NO
```

Problem 2: Equation

In this problem you are asked to solve a simple equation. The equation can have 6 formats:

- `x+<integer>=<integer>`: for example `x+20=42` with solution `x=22`.
- `x-<integer>=<integer>`: for example `x-20=42` with solution `x=62`.
- `<integer>+x=<integer>`: for example `20+x=42` with solution `x=22`.
- `<integer>-x=<integer>`: for example `20-x=42` with solution `x=-22`.
- `<integer>+<integer>=x`: for example `22+20=x` with solution `x=42`.
- `<integer>-<integer>=x`: for example `22-20=x` with solution `x=2`.

Write a program that reads an equation from the input, and outputs the solution of the equation. Note that all values are integers (less than $1000000 = 10^6$), and that there are no spaces in the input.

Example 1:

input:
```
x+20=42
```
output:
```
x=22
```

Example 2:

input:
```
22-20=x
```
output:
```
x=2
```

Example 3:

input:
```
20+x=62
```
output:
```
x=42
```

Problem 3: Highest common prime factor

A *prime* is an integer greater than 1 that has no positive divisors other than 1 and itself. Each positive integer greater than 1 can be written as a product of primes. This is called its prime factorization. For example, the prime factorizations of 42 and 9940 are:

$$42 = 2 \times 3 \times 7 \qquad \text{and} \qquad 9940 = 2 \times 2 \times 5 \times 7 \times 71$$

The greatest prime that divides two positive integers evenly is called their *Highest common prime factor*. If such a prime does not not exist, then their highest common prime factor is defined to be 1. From the given factorization, it is clear that the highest common prime factor of 42 and 9940 is 7.

Write a program that accepts two positive integers on the input, and outputs their highest common prime factor. You may assume that both integers are less than $1000000000 = 10^9$.

Example 1:
input:
```
42 9940
```
output:
```
7
```

Example 2:
input:
```
12345 1646
```
output:
```
823
```

Example 3:
input:
```
55 42
```
output:
```
1
```

Problem 4: Fibonacci's bunnies again

In the lecture you learned about the famous Fibonacci series. Recall the story behind this series: *Fibonacci started with a pair of baby rabbits, a baby boy rabbit and a baby girl rabbit. They were fully grown after one year, and did what rabbits do best, so that the next year two more baby rabbits (again a boy and a girl) were born. The next year these babies were fully grown and the first pair had two more baby rabbits (again, miraculously a boy and a girl). Ignoring problems of in-breeding, the next year the two adult pairs each have a pair of baby rabbits and the babies from last year mature. Fibonacci asked himself how many rabbits a single pair can produce in the course of time with this highly unbelievable breeding process (rabbits never die, and each year an adult pair produces a mixed pair of baby rabbits who mature the next year). He quickly discovered that this question was answered by the series*

$$F_0 = 1,\ F_1 = 1,\ F_n = F_{n-1} + F_{n-2}$$

where F_y denotes the number of pairs in year y.
In this problem we are going to make this process slightly more realistic. We still start with a single pair of baby rabbits, and the breeding process remains the same. However, a rabbit dies after 5 years, in other words a pair produces offspring for four years.

Write a program that reads from the input a year (a positive integer which is at most 50), and outputs the number of rabbit pairs in that year. Note that, like in the original Fibonacci series, we start counting from 0.

Example 1:
input:
```
0
```
output:
```
1
```

Example 2:
input:
```
5
```
output:
```
7
```

Example 3:
input:
```
42
```
output:
```
54270212
```

Problem 5: Stableford

In golf, a scoring system is used, which makes it possible for advanced and novice players to play against each other without the advanced player always winning.

A golf course consists of 18 holes. For each hole its so-called *par* is known: the number of strokes that the average professional player needs to complete the hole.

An amateur player has a so-called *handicap* (an integer), which is the total number of extra strokes that he/she is expected to need more than a professional player to complete the entire golf course. The minimum handicap is 0 and the maximum handicap is 36. These extra strokes are spread over the 18 holes of the golf course. The extra strokes are assigned to holes in order of difficulty, the so-called *indexes* of the holes. The hardest hole has index 1, the easiest has index 18. Each hole has a unique index, so there are no two holes with the same index.

For example, if a player has handicap 20 then he gets 2 extra strokes on the holes with the indexes 1 and 2 (the hardest holes), and 1 extra stroke for the remaining holes (because 20 = 18 + 2). This way, a player has for each hole his/her *personal par*, which is simply the par of the hole plus the number of extra strokes he/she gets for that hole.

In Stableford scoring, a player gets two points for a hole if the number of strokes he/she used is his/her personal par. For each stroke that he needs less, an additional point is awarded. If a player needs one more stroke than his personal par, then the player gets one point. If more strokes are used, no points are awarded for the hole. The total *Stableford score* for the entire round of golf is the sum of the scores of the 18 holes.

Write a program that reads from the input the score card of a player, and outputs the Stableford score. The input consists of 19 lines. The first line contains the handicap of the player. The remaining lines contain per hole the *par*, the *index* and the number of strokes that the player used for the hole.

Example:

input:

```
30
4 2 6
4 4 8
5 11 6
4 9 7
3 13 6
5 18 4
3 7 7
4 15 5
4 5 6
4 1 6
4 3 6
4 12 7
5 10 7
4 14 6
3 17 4
5 8 7
3 16 5
4 6 6
```

output:

```
29
```