Midterm Imperative Programming
Oct. 11 2021, 18:30-21:30h

- 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: Comparing Fractions

You probably see immediately that $\frac{1}{3} < \frac{1}{2}$ is true. But what about $\frac{75}{43} < \frac{73}{42}$? This is actually false, but is harder to see immediately.

The input of this problem is of the form `a/b<c/d` or `a/b>c/d`, where `a`,`b`,`c`, and `d` are positive integers between 0 and 10000 (both boundaries included). The output of you program should be `YES` if the inequality on the input is true, and `NO` otherwise.

Example 1:
input:
```
1/3<1/2
```
output:
```
YES
```

Example 2:
input:
```
1/6>1/4
```
output:
```
NO
```

Example 3:
input:
```
75/43<73/42
```
output:
```
NO
```

Problem 2: Peculiar Numbers

We call a non-negative integer a *peculiar number* if it satisfies the following two properties:

- the sum of its digits is a prime number,
- the product of its digits is a perfect square.

Recall, that an integer n is a perfect square if there exists an integer $m > 0$ such that $n = m^2$.

An example of a peculiar number is 155, because the sum of its digits is 1+5+5=11, which is a prime number. Moreover, the product of its digits is $1 \times 5 \times 5 = 25$ which is a perfect square (because $25 = 5^2$).

Write a program that reads from the input two integers a and b (where $0 \leq a \leq b \leq 1000000 = 10^6$). The output should be the number of peculiar numbers in the range a up to and including b.

Example 1:
input:
```
2 10
```
output:
```
0
```

Example 2:
input:
```
150 160
```
output:
```
1
```

Example 3:
input:
```
42 4200
```
output:
```
106
```

Problem 3: Good Primes

Let p be a prime number greater than 2. We call the numbers a and b the surrounding primes of p if a is the largest prime number less than p and b is the smallest prime greater than p. For example, the surrounding primes of $p = 11$ are $a = 7$ and $b = 13$.

We call a number n, where $n > 2$, a *good prime* if n is a prime number, and its square n^2 is greater than the product of its two surrounding primes.

For example, the prime number 977 is a good prime, because its surrounding primes are 971 and 983, and $977^2 = 954529 > 954493 = 971 \times 983$. The prime 997 is not a good prime, because its surrounding primes are 991 and 1009, and $991 \times 1009 = 999919 > 994009 = 997^2$.

Write a program that reads from the input two integers a and b (where $2 < a \leq b \leq 10000 = 10^4$). The output should be the number of good primes in the range a up to and including b.

Example 1:

input:
```
10 12
```
output:
```
1
```

Example 2:

input:
```
150 160
```
output:
```
1
```

Example 3:

input:
```
42 4200
```
output:
```
287
```

Problem 4: Absolute Difference Triangles

The input for this problem consists of two lines. The first line holds a single integer n, where $0 < n \leq 1000$. The second line consists of a series of n integers on which you need to perform the following steps:

1. Print the n integers in the same order that they were read from the input,
2. Compute a new series that is one element shorter as follows. The first element of this new series is the absolute difference of the first and the second element of the original series. The second element of the new series is the absolute difference of the second and the third element of the original series, and so on.
3. Print the series obtained in step 2.
4. Using the newly obtained series, repeat step `2` until only one value remains.

For example, let us take the input sequence `[1 1 1 2 2 4 12]`. In step 2, this series is reduced to `[|1-1| |1-1| |1-2| |2-2| |2-4| |4-12|] = [0 0 1 0 2 8]`. Repeating this process eventually ends with a series containing a single element, being `2` (see Example 1).

Example 1:

input:
```
7
1 1 1 2 2 4 12
```
output:
```
1 1 1 2 2 4 12
0 0 1 0 2 8
0 1 1 2 6
1 0 1 4
1 1 3
0 2
2
```

Example 2:

input:
```
5
1 8 2 4 7
```
output:
```
1 8 2 4 7
7 6 2 3
1 4 1
3 3
0
```

Example 3:

input:
```
8
1 1 1 2 3 3 6 18
```
output:
```
1 1 1 2 3 3 6 18
0 0 1 1 0 3 12
0 1 0 1 3 9
1 1 1 2 6
0 0 1 4
0 1 3
1 2
1
```

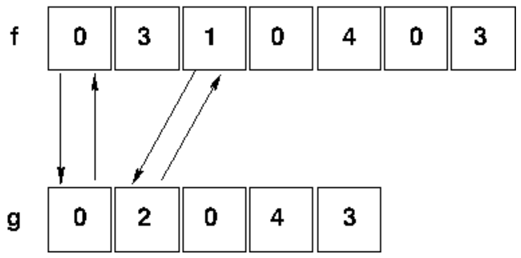
Problem 5: Matching pairs

The input of this problem are two arrays `f` and `g`, both containing non-negative integers. The input format looks like this:

```
n: f[0] f[1] .... f[n-1]
m: g[0] g[1] .... g[m-1]
```

Clearly, `n` and `m` are the lengths of respectively the arrays `f` and `g`. You may assume that for both arrays the length is at most $1000000 = 10^6$. The output of the program must be the number of matching pairs `(a,b)` such that `f[a]==b` and `g[b]==a`.

For example 1, the required output is 2, which is clear from the following figure.



Example 1:

input:

```
7: 0 3 1 0 4 0 3
5: 0 2 0 4 3
```

output:

```
2
```

Example 2:

input:

```
5: 0 2 0 4 3
7: 0 3 1 0 4 0 3
```

output:

```
2
```

Example 3:

input:

```
10: 5 1 6 8 7 4 0 9 9 1
10: 9 1 4 3 5 4 0 1 3 9
```

output:

```
3
```