Resit Mid-exam Imperative Programming
Oct. 12 2020, 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 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: Nine's Complement

The *nine's complement* of a decimal number is obtained by replacing each digit with nine minus that digit. For example, the nine's complement of 123456 is $999999 - 123456 = 876543$. Note that this may lead to leading zeros (see example 3).

Write a program that reads a non-negative integer n from the input (where $0 \leq n \leq 100000000 = 10^8$), and outputs the nine's complement of n.

Example 1:
input:
```
123456
```
output:
```
876543
```

Example 2:
input:
```
846
```
output:
```
153
```

Example 3:
input:
```
9203
```
output:
```
0796
```

Problem 2: Unitary Divisors

A positive integer d is a unitary divisor of a positive integer n if it satisfies the following two requirements:

1. d is a divisor of n,
2. d and n/d have no other common factor other than 1.

For example, 5 is a unitary divisor of 60 since 5 is a divisor of 60 and the only common divisor of 5 and $12 (= 60/5)$ is 1.

Write a program that reads from the input two positive `ints` d and n (in that order) and outputs whether d is a unitary divisor of n.

Example 1:
input:
```
5 60
```
output:
```
YES
```

Example 2:
input:
```
20 120
```
output:
```
NO
```

Example 3:
input:
```
5 140
```
output:
```
YES
```

Problem 3: n-Gap Prime Pairs

A pair (p, q) consisting of two prime numbers p and q is called an *n-gap prime pair* if $q = p + n$. Recall that a *prime* is an integer greater than 1 that has no positive divisors other than 1 and itself.

For example, the pair $(3, 5)$ is a *2-gap prime pair*, since 3 and 5 are both primes and the gap in between them is 2.

Write a program that reads from the input integers a, b, and n, where $1 \leq a \leq b \leq 5000000 = 5 \cdot 10^6$ and $n > 1$. The output of the program should be the number of *n-gap prime pairs* in the range from a up to and including b. For example, for $a = 1$, $b = 20$, and $n = 2$ your program should output 4 since the only four *2-gap prime pairs* (p, q) where both p and q are taken from the range $[1..20]$ are $(3, 5)$, $(5, 7)$, $(11, 13)$ and $(17, 19)$.

Example 1:
input:
```
1 20 2
```
output:
```
4
```

Example 2:
input:
```
1 50 6
```
output:
```
9
```

Example 3:
input:
```
15 200 18
```
output:
```
20
```

Problem 4: Ancestor

Recall that an *ancestor* is any person from whom one is a descendant. So, an ancestor of x can be a parent of x, a grandparent of x, a great-grandparent of x, and so on.

In this problem we give persons an identification number (simply because it is easier to process `int`s than strings) ranging from 0 up to (and not including) some upper bound n. This number is given on the input. You may assume that $1 \leq n \leq 1000$. Next, there is a series of lines of the form `father(X)=Y`, where `X` and `Y` are identification numbers. Of course, this denotes that `X` is the father of `Y`. The series is terminated by a query of the form `ancestor(A,B)`, where `A` and `B` are identification numbers. The output of your program should be `YES` if `A` is an ancestor of `B`, or `NO` otherwise.

Example 1:

input:

```
5
father(1)=0
father(2)=1
father(3)=4
ancestor(0,2)
```

output:

```
YES
```

Example 2:

input:

```
5
father(1)=0
father(2)=1
father(3)=4
ancestor(2,0)
```

output:

```
NO
```

Example 3:

input:

```
5
father(1)=0
father(2)=1
father(3)=4
ancestor(0,3)
```

output:

```
NO
```

Problem 5: Check!

In the game of chess, a king is said to be *in check* when he is being attacked by another chess piece of the opposing colour. In this problem we consider valid chessboard configurations with a black king, zero or more black pawns, a white king, and a white queen. The goal of this exercise is to determine whether the black king is in check, i.e. it is under direct attack by the white queen. This is the case whenever the king and queen are on the same line (either horizontally, vertically or diagonally) without any piece between the black king and the white queen.

The input for this problem consists of 8 rows and 8 columns of characters specifiying the board configuration. The character '`#`' denotes an empty square, the character '`p`' denotes a black pawn, the character '`k`' denotes the black king, the character '`K`' denotes the white king, and the character '`Q`' denotes the white queen.

The output of the program should be `YES` if the black king is in check. Otherwise the output should be `NO`.

Example 1:

input:

```
########
####k###
########
####Q###
########
########
###K####
########
```

output:

```
YES
```

Example 2:

input:

```
########
####k###
###p####
##Q#####
########
###K####
########
########
```

output:

```
NO
```

Example 3:

input:

```
########
k#######
p#pp####
ppQp####
########
######K#
########
########
```

output:

```
YES
```