Mid-exam Imperative Programming
Oct. 11 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: Binary Representation

The binary representation of the integer 44 is 101100 because:

$$44 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$$

Write a program that reads from the input a positive integer n (where $0 < n \leq 1000000$), and outputs its binary representation according to the format in the following examples. Note that a binary representation starts with a 1 (i.e. no leading zeroes). Morover, note that spaces enclose the symbol +.

Example 1:

input:
```
44
```
output:
```
44=1*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 0*2^0
```

Example 2:

input:
```
42
```
output:
```
42=1*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0
```

Example 3:

input:
```
123
```
output:
```
123=1*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0
```

Problem 2: Divisible by 11

An easy way to test whether a non-negative integer n is divisible by 11 goes as follows. Let p be the sum of the digits of n which are at odd positions (the first digit has position 1), and q the sum of the digits which are at even positions. The number n is divisible by 11 if and only if the difference of p and q is divisble by 11.

For example, let $n = 119777658$, then $p = 1 + 9 + 7 + 6 + 8 = 31$, and $q = 1 + 7 + 7 + 5 = 20$. The difference $p - q = 11$, which is clearly divisible by 11. Hence, 119777658 is divisible by 11.

Write a program that reads from the input a non-negative integer n (where $0 \leq n \leq 10^{100}$, i.e. n need not fit in a standard `int`), and outputs `YES` if n is divisible by 11, otherwise it should output `NO`.

Example 1:

input:
```
121
```
output:
```
YES
```

Example 2:

input:
```
123
```
output:
```
NO
```

Example 3:

input:
```
119777658
```
output:
```
YES
```

Problem 3: Difference of Squares

In this exercises we search, given an integer n (where $0 \leq n \leq 1000000$), for solutions of the equation $x^2 - y^2 - z^2 = n$. Here x, y, and z are non-negative integers with the extra constraint that $0 \leq z \leq y \leq n$. For example, for $n = 42$ there are 7 solutions:

$$\begin{aligned}
22^2 - 21^2 - 1^2 &= 42 \\
10^2 - 7^2 - 3^2 &= 42 \\
26^2 - 25^2 - 3^2 &= 42 \\
34^2 - 33^2 - 5^2 &= 42 \\
22^2 - 19^2 - 9^2 &= 42 \\
38^2 - 31^2 - 21^2 &= 42 \\
50^2 - 37^2 - 33^2 &= 42
\end{aligned}$$

Write a program that reads from the input an integer n (where $0 \leq n \leq 1000$), and outputs the number of combinations that satisfy the equation and constraints that are described above.

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

Example 2:
input:
```
9
```
output:
```
4
```

Example 3:
input:
```
100
```
output:
```
35
```

Problem 4: Palindromic Prime Product Numbers

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 *palindromic number* is a number that remains the same when its digits are reversed. An example of a palindromic number is 1234321.
We call a palindromic number a *palindromic prime product number* if it is the product of two distinct primes. For example, the number 9163619 is a palindromic number, and it is the product of the primes 157 and 58367. Hence, 9163619 is a palindromic prime product number. The number palindromic number 121 is not a palindromic product number since it is not the product of two distinct primes ($121 = 11 \times 11$).
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 palindromic product numbers n with $a \leq n \leq b$.

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

Example 2:
input:
```
1 100
```
output:
```
5
```

Example 3:
input:
```
1 370
```
output:
```
12
```

Problem 5: McCarthy Function

The computer scientist John McCarthy defined the following function $f(n)$ for non-negative integers n:

$$\text{If } n > 100 \text{ then } f(n) = n - 10 \text{ else } f(n) = f(f(n+11))$$

The function has a surprising property (at first sight). For all values n that are less or equal 101, the function returns 91. For example, $f(99) = f(f(110)) = f(100) = f(f(111)) = f(101) = 91$.
The computation of $f(87)$ takes many more steps:
$f(87) = f(f(98)) = f(f(f(109))) = f(f(99)) = f(f(f(110))) = f(f(100)) = f(f(f(111))) = f(f(101)) = f(91) = f(f(102)) = f(92) = f(f(103)) = f(93) = f(f(104)) = f(94) = f(f(105)) = f(95) = f(f(106)) = f(96) = f(f(107)) = f(97) = f(f(108)) = f(98) = f(f(109)) = f(99) = f(f(110)) = f(100) = f(f(111)) = f(101) = 91$

Write a program that reads an integer n from the input, and outputs the computation steps of $f(n)$. So, for $n = 99$ the output must be `f(99)=f(f(110))=f(100)=f(f(111))=f(101)=91` and not just 91.

Example 1:

input:
```
99
```
output:
```
f(99)=f(f(110))=f(100)=f(f(111))=f(101)=91
```

Example 2:

input:
```
97
```
output:
```
f(97)=f(f(108))=f(98)=f(f(109))=f(99)=f(f(110))=f(100)=f(f(111))=f(101)=91
```

Example 3:

input:
```
200
```
output:
```
f(200)=190
```