Mid-exam Imperative Programming (7-10-2013)

- You can solve the problems in any order.
- The first problem is worth one grade point, the remaining four exercises are worth two grade points each. You get one grade point for free.
- Your assessment is based solely on the judgment given by Justitia. If Justitia accepts your program, then the problem is considered to be solved.
- There will be no assessment of programming style. However, (very) inefficient programs may be rejected. In such cases, the error will be 'time limit exceeded'. The runtime limit for each problem is one second.
- Note the hints that Justitia 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.
- For each problem, there are three examples of inputs and associated expected outputs. The input files of these examples can be found in Justitia, together with the corresponding output files. These files are called `1.in`, `2.in`, `3.in` and `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. Moreover, on the lab computers the command `spcat` is available. The command `spcat file.txt` shows the content of the file `file.txt` where all spaces are replaced by a tilde (˜), tabs by `\t` and newlines by `\n`. Using this little tool you can see whether your output is exactly the same as that of the requested output.

Problem 1: Happy New Year

A 'normal' year consist of 365 days, while a leap year consists of 366 days. Three criteria must be taken into account to identify leap years:

- The year is an integer multiple of 4;
- If the year is an integer multiple of 100, it is NOT a leap year, unless:
- The year is also an integer multiple of 400. Then it is a leap year.

This means that 2000 and 2400 are leap years, while 1800, 1900, 2100, 2200, 2300 and 2500 are NOT leap years.

New Year's Day falls on January 1 and marks the start of a new year. In 1901 this was a Tuesday. Write a program that reads from the input a year (an integer number `y` such that `1901<=y<=2013`) and prints on the output which day it was on January 1 of that year.

Example 1:
input:
```
1901
```
output:
```
Tuesday
```

Example 2:
input:
```
1970
```
output:
```
Thursday
```

Example 3:
input:
```
2013
```
output:
```
Tuesday
```

Problem 2: Pythagorean triangles

A Pythagorean triangle is a right-angled triangle with integer side lengths. In other words, it has integer side lengths `a`, `b`, and `c` such that `a*a + b*b == c*c`. A well-known Pythagorean triangle has `a=3`, `b=4`, and `c=5`. The area of this triangle is $6 = (3 \times 4)/2$.

Write a program that reads from the input an integer `n`: the area of a triangle. You may assume that `0<n<=10000`. The program should print all Pythagorean triangles that have area `n`. If no such triangle exists, then the program should not produce any output. Note that each triangle must be printed on a separate line as a triplet `a b c` where `a<=b<c`.

Example 1:
input:
```
6
```
output:
```
3 4 5
```

Example 2:
input:
```
600
```
output:
```
30 40 50
```

Example 3:
input:
```
840
```
output:
```
15 112 113
24 70 74
40 42 58
```

Problem 3: Amicable pairs

Let `s(n)` be defined as the sum of the *proper divisors* of a positive integer `n`. A positive integer `d` (where `d < n`) is a proper divisor of `n` if `n` is an integer multiple of `d`.

Now, if `s(a) = b` and `s(b) = a`, where `a` differs from `b`, then `a` and `b` are an *amicable pair*.

For example, the proper divisors of 220 are 1, 2, 4, 5, 10, 11, 20, 22, 44, 55 and 110; therefore `s(220) = 284`. The proper divisors of 284 are 1, 2, 4, 71 and 142; so `s(284) = 220`. So, the numbers 284 and 220 form an amicable pair.

Write a program that reads a positive integer `n` from the input: the length of a series of positive integers. Then a series of `n` positive integers follows on the input. This series is sorted in increasing order. You may assume that `0<n<=100`. Your program should print on the output all amicable pairs `a` and `b` that are present in the input. Note that your program should print the pairs `a` and `b` in increasing order of `a`. Moreover, each pair should be printed only once.

Example 1:
input:
```
3
6
220
284
```
output:
```
220 284
```

Example 2:
input:
```
5
6
220
284
1184
1210
```
output:
```
220 284
1184 1210
```

Example 3:
input:
```
2
6
220
```
output:

Problem 4: Run-length encoding

A *binary image* is a digital image that has only two possible values for each pixel: 0 or 1. A compact way to represent a binary image is to use *run-length encoding*. This works as follows. On the input there are first two positive integer values `h` and `w`: the height and width of the image. You may assume that `h` and `w` are at most 100. Then follows a non-negative number `n`, designating the number of *runs* in the image. A run is a non-empty contiguous series of pixels on a row that all have the pixels value 1. The remaining `n` input lines each contain 3 non-negative values: `r c len`. Since the runs are non-empty, we know for sure that `len>0`. The values `(r,c)` designate the start location of a run, and `len` is the length of this run. Note that the coordinate `(0,0)` is the upper left corner of the image. The location `(r,c)` is the pixel in row `r` and in column `c`.
Write a program that reads from the input the run-length encoding of an image, and prints on the output the decoded binary image.

Example 1:

input:
```
5 6
1
1 0 5
```
output:
```
000000
111110
000000
000000
000000
```

Example 2:

input:
```
5 6
5
2 0 5
0 2 1
1 2 1
3 2 1
4 2 1
```
output:
```
001000
001000
111110
001000
001000
```

Example 3:

input:
```
5 6
5
4 2 1
0 2 1
1 2 1
2 0 5
3 2 1
```
output:
```
001000
001000
111110
001000
001000
```

Problem 5: Circular Primes

A circular prime is a prime number with the property that all the numbers generated at each intermediate step when cyclically permuting its digits are prime. For example, the number 1193 is a circular prime, since 1931, 9311 and 3119 all are also prime. Note that a circular prime with at least two digits can only consist of combinations of the digits 1, 3, 7 or 9, because having 0, 2, 4, 6 or 8 as the last digit makes the number divisible by 2, and having 0 or 5 as the last digit makes it divisible by 5.

Write a program that reads from the input two integers `a` and `b` (where `0<a<b<=100000`, so the values fit in an **int**) and prints all circular primes `p` with the property `a<=p<=b` in increasing order. Each circular prime must be printed on a separate line.

Example 1:

input:
```
5 20
```
output:
```
5
7
11
13
17
```

Example 2:

input:
```
1000 4000
```
output:
```
1193
1931
3119
3779
```

Example 3:

input:
```
90000 100000
```
output:
```
91193
93719
93911
99371
```