The 5th Annual Newport News Computer Challenge

Wednesday, March 1, 2006

Team Packet

Java Problems

The 5th Annual Newport News Computer Challenge

Wednesday, March 1, 2006

Java Problems

```
1. INPUT n
2. PRINT n
3. IF n = 1 THEN STOP
● ● ●
```

The 3n + 1 Problem ~ 10 points

Ugly Numbers ~ 10 points

Credit Card Validation ~ 20 points

Maya Calendar ~ 30 points

The 5th Annual Newport News Computer Challenge

Wednesday, March 1, 2006

The 3n + 1 Problem (10 points)

Consider the following algorithm, expressed in psuedo-code of no particular computer language:

1. INPUT n
2. PRINT n
3. IF n = 1 THEN STOP
4. IF n is odd THEN n ← 3n + 1
5. ELSE n ← n/2
6. GOTO 2

(The ← symbol means assignment.)

Given the input 22, the following sequence of numbers will be printed 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1.

It is conjectured that the algorithm above will terminate (when a 1 is printed) for any integral input value. Despite the simplicity of the algorithm, it is unknown whether this conjecture is true. It has been verified, however, for all integers *n* such that $0 < n < 1{,}000{,}000$ (and, in fact, for many more numbers than this.)

Given an input *n*, it is possible to determine the number of numbers printed (including the 1). For a given *n* this is called the *cycle-length* of *n*. In the example above, the cycle length of 22 is 16.

For any two numbers *i* and *j* you are to determine the maximum cycle length over all numbers between *i* and *j* inclusive.

Sample run:

DO NOT attempt to use a GUI interface for this problem.

```
The 3n + 1 Problem, by [your school & team name here]
Enter two positive integers, on separate lines below, zeros to quit.
1
10
Maximum cycle length: 20

Enter two positive integers, on separate lines below, zeros to quit.
100
200
Maximum cycle length: 125

Enter two positive integers, on separate lines below, zeros to quit.
201
210
Maximum cycle length: 89

Enter two positive integers, on separate lines below, zeros to quit.
900
1000
Maximum cycle length: 174

Enter two positive integers, on separate lines below, zeros to quit.
0
0

Press Enter to continue . . .
```

The 5th Annual Newport News Computer Challenge

Wednesday, March 1, 2006

Ugly Numbers (10 points)

Ugly numbers are numbers whose only prime factors are 2, 3 or 5.

The sequence

1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, ...

shows the first 11 ugly numbers. By convention, 1 is included.

Write a program that prompts the user to input a positive integer *n* and outputs the *nth* ugly number followed by the next four ugly numbers.

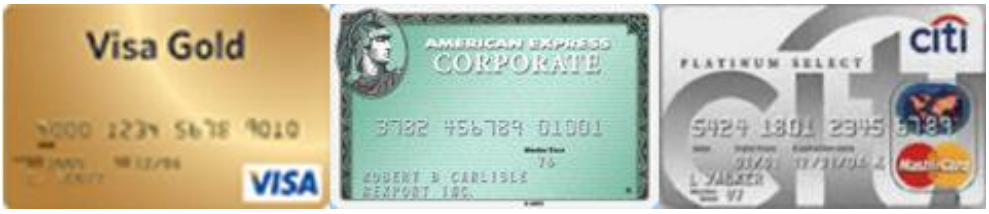
Same points with or without a GUI.

The 5th Annual Newport News Computer Challenge

Wednesday, March 1, 2006

Credit Card Validation (20 points)

Same points with or without a GUI.



This problem lets you in on a little secret used by banks and lending institutions to determine whether or not a credit card number is valid. What we're talking about here is not whether a credit card number is currently being used by someone, but rather whether or not the number COULD be used by someone, should a bank or lending institution issue that number.

Below is the algorithm that tests for credit card validity:

Step	**Sample (using 5318-2795-1234-5678)**
1. Determine whether the total number of digits is odd or even. If even begin with the first digit, if odd begin with the second digit.	1. The card number has 16 digitis, which is even, so we will begin with the first digit.
2. Then use every other digit. (called the odd-positioned digits or even-positioned digits as appropriate)	2. The odd-positioned digits are 5, 1, 2, 9, 1, 3, 5, and 7.
3. Perform this function on each of these digits: double the digit and (if the doubled result has two digits) add the digits of the doubled result.	5 × 2 = 10, 1 + 0 = 1 1 × 2 = 2 2 × 2 = 4 9 × 2 = 18, 1 + 8 = 9 1 × 2 = 2 3 × 2 = 6 5 × 2 = 10, 1 + 0 = 1 7 × 2 = 14, 1 + 4 = 5
4. Add the sum of these results.	4. The sum of 1, 2, 4, 9, 2, 6, 1, and 5 is 30.
5. Add the sum of the other digits (in this case, the even-positioned digits).	5. The sum of 3, 8, 7, 5, 2, 4, 6, and 8 is 43.
6. Add the two sums.	6. 30 plus 43 is 73.
7. If the final result is divisible by 10, the credit card number is valid. If not, it is invaliid	7. 73 is not divisible by 10. Therefore the credit card number is invalid.

Your job is not just to test for validity, but rather to allow the user to enter the first ***n-1*** digits of a ***n***-digit credit card number and then output the digit that, if used as the n^{th} digit, would make the card number valid. Input should allow dashes or spaces to be inserted anywhere in the number.

Samples:

First n-1 digits	nth digit
`5318-2795-1234-567`	5
`1234 12 123`	2
`000-000-100`	8
`000-000-1000`	9

The 5th Annual Newport News Computer Challenge

Wednesday, March 1, 2006

Maya Calendar (30 points)

25 points for solving the problem. 5 points for using a GUI. No GUI points if input fails.

The Maya civilization used a 365 day long year, called *Haab*, which had 19 months. Each of the first 18 months was 20 days long, and the names of the months were *pop, uo, zip, zotz, tzec, xul, yaxkin, mol, chen, yax, zac, ceh, mac, kankin, muan, pax, kayab, cumku*. Instead of having names, the days of the months were denoted by numbers starting from 0 to 19 (using their vigesimal, or base 20, numbering system). The last month of *Haab* was called *uayet* and had 5 days denoted by numbers 0, 1, 2, 3, 4. The Maya believed that this month was unlucky, the court of justice was not in session, the trade stopped, people did not even sweep the floor.

For religious purposes, the Maya used another calendar in which the year was called *Tzolkin* (holly year). The year was divided into thirteen periods, each 20 days long. Each day was denoted by a pair consisting of a number and the name of the day. They used 20 names: *imix, ik, akbal, kan, chicchan, kimi, manik, lamat, muluk, ok, chuwen, eb, ben, ix, mem, kib, kaban, etznab, kawac, ajaw* and 13 numbers; both in cycles.

Notice that each day has an unambiguous description. For example, at the beginning of the year the days were described as follows:

1 imix, 2 ik, 3 akbal, 4 kan, 5 chicchan, 6 kimi, 7 manik, 8 lamat, 9 muluk, 10 ok, 11 chuwen, 12 eb, 13 ben, 1 ix, 2 mem, 3 kib, 4 kaban, 5 etznab, 6 kawac, 7 ajaw, and again in the next period *8 imix, 9 ik, 10 akbal...*

Years (both *Haab* and *Tzolkin*) were denoted by numbers 0, 1, ..., where the number 0 was the beginning of the world. Thus, the first day was:

Haab: `0 pop 0` (Format: *NumericalDayOfTheMonth Month Year*)
Tzolkin: `1 imix 0` (Format: *Number NameOfTheDay Year*)

Write a program to convert any date in the *Haab* calendar up to and including the year 9999 to its corresponding date in the *Tzolkin* calendar. Your program should reject invalid input.

Sample conversions:

Haab	*Tzolkin*
0 pop 0	1 imix 0
5 kayab 500	7 kimi 703
8 zip 27	11 kan 38
0 pop 2006	5 chuwen 2816

Haab	*Tzolkin*
0 pop 1	2 kimi 1
3 uayet 1001	13 muluk 1406
14 cumku 9999	6 ok 14038
5 uayet 20	*invalid input*