Repetition

Lab #2 (CS1010 AY2011/2 Semester 4)

Date of release: 2 July 2012, Monday, 14:00hr.

Submission deadline: 7 July 2012, Saturday, 23:59hr.

School of Computing, National University of Singapore

0 Introduction

Important: Please read Lab Guidelines before you continue.

This lab requires you to do 3 exercises on **repetition control structure**. **Only the first 2 exercises will be graded**. The third exercise is for your extra practice only.

If you have any questions on the task statements, you may post your queries **on the relevant IVLE discussion forum**. However, do **not** post your programs (partial or complete) on the forum before the deadline!

Note that you are **NOT allowed to use recursion** for the exercises here. Using it would amount to violating the objective of this lab assignment.

1 Exercise 1: The 3x+1 Problem

1.1 Learning objectives

- Using selection and repetition statement.
- Writing function.

1.2 Task statement

The **3x+1 problem** is also known as the **Collatz problem** (and a few other names), named after Lothar Collatz who first proposed it in 1937. The **Collatz conjecture**, still unsolved in mathematics, states the following:

> Take any natural number *n*. If *n* is even, divide it by 2 to get *n*/2; if *n* is odd, triple it and add 1 to obtain 3*n* + 1 (see mathematical expression below). Repeat the process indefinitely. No matter what number your start with, you will always eventually reach 1.

$$f(n) = \begin{cases} n/2 & \text{if } n \equiv 0 \pmod 2 \\ 3n+1 & \text{if } n \equiv 1 \pmod 2 \end{cases}$$

You are not required to prove the Collatz conjecture, but to write a program **`collatz.c`** that reads in a positive integer and determines how many iterations it takes to reach 1.

For example, if the input is 3, then the answer would be 7, as 3 → 10 → 5 → 16 → 8 → 4 → 2 → 1.

1.3 Sample run

Sample run using interactive input (user's input shown in blue; output shown in **bold purple**). Note that the first two lines (in green) below) are commands issued to compile and run your program on UNIX.

```
$ gcc -Wall collatz.c -o collatz
$ collatz
Enter a natural number: 3
Number of iterations = 7
```

1.4 Number of submission

For this exercise, the number of submission is **10**.

1.5 Important notes

- The skeleton program is provided here: collatz.c
- Write a function **`count_iterations`** to compute the number of iterations required for the input value to reach 1.
- In writing functions, we would like you to include function prototypes before the main function, and the function definitions after the main function.

1.6 Estimated development time

The time here is an estimate of how much time we expect you to spend on this exercise. If you need to spend way more time than this, it is an indication that some help might be needed.

- Devising and writing the algorithm (pseudo-code): 10 minutes
- Translating pseudo-code into code: 5 minutes
- Typing in the code: 5 minutes
- Testing and debugging: 10 minutes
- Total: 30 minutes

2 Exercise 2: Candles

2.1 Learning objectives

- Using repetition statement.
- Writing function.
- Applying neat logic in problem solving.

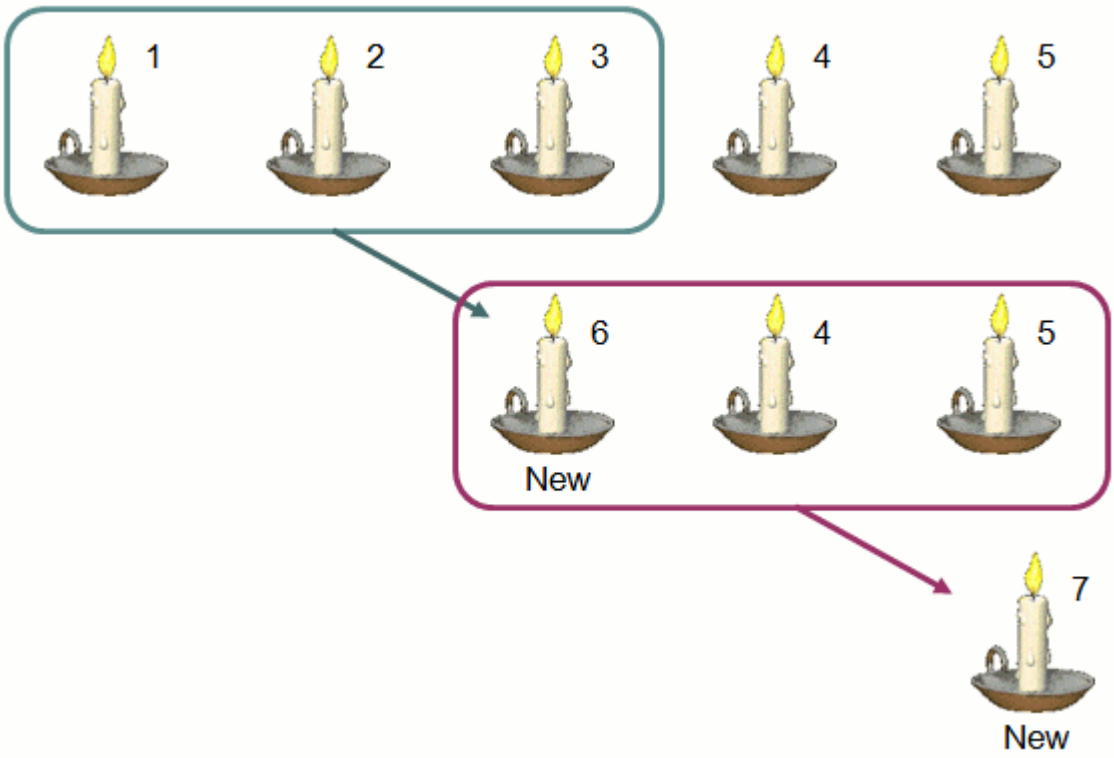
2.2 Task statement

Alexandra has *n* candles. He burns them one at a time and carefully collects all unburnt residual wax. Out of the residual wax of exactly *k* (where $k > 1$) candles, he can roll out a new candle.

Write a program **`candles.c`** to help Alexandra find out how many candles he can burn in total, given two positive integers *n* and *k*.

The output should print the total number of candles he can burn.

The diagram below illustrates the case of $n = 5$ and $k = 3$. After burning the first 3 candles, Alexandra has enough residual wax to roll out the 6^{th} candle. After burning this new candle with candles 4 and 5, he has enough residual wax to roll out the 7^{th} candle. Burning the 7^{th} candle would not result in enough residual wax to roll out anymore new candle. Therefore, in total he can burn 7 candles.



2.3 Sample runs

Sample run using interactive input (user's input shown in blue; output shown in **bold purple**). Note that the first two lines (in green) below) are commands issued to compile and run your program on UNIX.

```
$ gcc -Wall candles.c -o candles
$ candles
Enter number of candles and
number of residuals to make a new candle: 5 3
Total candles burnt = 7
```

Second sample run:

```
Enter number of candles and
number of residuals to make a new candle: 100 7
Total candles burnt = 116
```

2.4 Number of submission

For this exercise, the number of submission is **10**.

2.5 Important notes

- The skeleton program is provided here: candles.c
- Write a function **`count_candles`** to compute the total number of candles burnt.
- In writing functions, we would like you to include function prototypes before the main function, and the function definitions after the main function.
- This is a problem-solving task where we look for **neat logic** in your program. Using descriptive variable names, and adding appropriate comments will help the reader (and yourself) to understand the logic better.

2.6 Estimated development time

The time here is an estimate of how much time we expect you to spend on this exercise. If you need to spend way more time than this, it is an indication that some help might be needed.

- Devising and writing the algorithm (pseudo-code): 15 minutes
- Checking/tracing the algorithm: 15 minutes
- Translating pseudo-code into code: 10 minutes
- Typing in the code: 10 minutes
- Testing and debugging: 20 minutes
- Total: 1 hour 10 minutes

3 Exercise 3: Fortune Cookies

3.1 Learning objectives

- Using nested loops. (See 3.5 Important notes below.)
- Writing functions.

3.2 Task statement

Write a program **`cookies.c`** to read in a positive integer and add up its digits repeatedly until the sum is a single digit. For example, if the integer is 12345, then adding its digits (1 + 2 + 3 + 4 + 5) yields 15, and adding its digits again (1 + 5) yields 6. Hence the answer is 6.

Using this single digit result, print out the corresponding Fortune Cookie message according to the table below:

Digit	Fortune Cookie message
1	You will have a fine capacity for the enjoyment of life.
2	Now is the time to try something new.
3	Don't let doubt and suspicion bar your progress.
4	Your principles mean more to you than any money or success.
5	Accept the next proposition you hear.
6	A handful of patience is worth more than a bushel of brains.
7	You have an active mind and a keen imagination.
8	You are talented in many ways.
9	Treat everyone as a friend.

3.3 Sample runs

Sample run using interactive input (user's input shown in blue; output shown in **bold purple**). Note that the first two lines (in green) below) are commands issued to compile and run your program on UNIX.

```
$ gcc -Wall cookies.c -o cookies
$ cookies
Enter a positive integer: 12345
A handful of patience is worth more than a bushel of brains.
```

Second sample run:

```
Enter a positive integer: 67890
Don't let doubt and suspicion bar your progress.
```

Third sample run:

```
Enter a positive integer: 11111111
You are talented in many ways.
```

3.4 Number of submission

For this exercise, the number of submission is **10**.

3.5 Important notes

- The skeleton program is provided here: cookies.c
- Write a function **`sum_digits`** to repeatedly sum the digits until it gets to a single digit answer, and return this answer.
- Write a function **`print_cookie`** to print the corresponding cookie message depending on the digit computed.
- Although the learning objectives indicate "using nested loops", this problem can be done with a single loop. See if you are able to come out with an algorithm using a single loop. It is okay if you cannot; a doubly nested loop is just as acceptable for this exercise. You may discuss this at your discussion session.
- In writing functions, we would like you to include function prototypes before the main function, and the function definitions after the main function.
- Note that this exercise **will not be graded**.

3.6 Estimated development time

The time here is an estimate of how much time we expect you to spend on this exercise. If you need to spend way more time than this, it is an indication that some help might be needed.

- Devising and writing the algorithm (pseudo-code): 20 minutes
- Translating pseudo-code into code: 10 minutes
- Typing in the code: 10 minutes
- Testing and debugging: 30 minutes
- Total: 1 hour 10 minutes

4 Deadline

The deadline for submitting all programs is **7 July 2012, Saturday, 23:59hr**. Late submission will NOT be accepted.

Go back to CS1010 Labs page.

Aaron Tan
Tuesday, August 30, 2011 06:40:35 PM SGT