CS1010: LAB ASSIGNMENT 2

0 INTRODUCTION

This lab contains 3 exercises.

To receive the attempt mark for this lab assignment, you must submit all 3 programs and get a passing feedback mark for each of the program.

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!

Please be reminded that lab assignments must be done in your own effort.

Important notes applicable to all exercises here:

- You are NOT allowed to use global variables. (A global variable is one that is not declared in any function.)
- You are free to introduce additional functions if you deem it necessary. This must be supported by well-thought-out reasons, not a haphazard decision. By now, you should know that you cannot write a program haphazardly.
- In writing functions, please put function prototypes before the main() function, and the function definitions after the main() function.

Mark allocation is as follows (60% correctness, 20% style, 20% design):

- Exercise 1: 1 mark
- Exercise 2: 2 mark
- Exercise 3: 3 mark

1 EXERCISE 1: CANDLES

1.1 LEARNING OBJECTIVES

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

1.2 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 6th candle. After burning this new candle with candles 4 and 5, he has enough residual wax to roll out the 7th candle. Burning the 7th candle would not result in enough residual wax to roll out anymore new candle. Therefore, in total he can burn 7 candles.

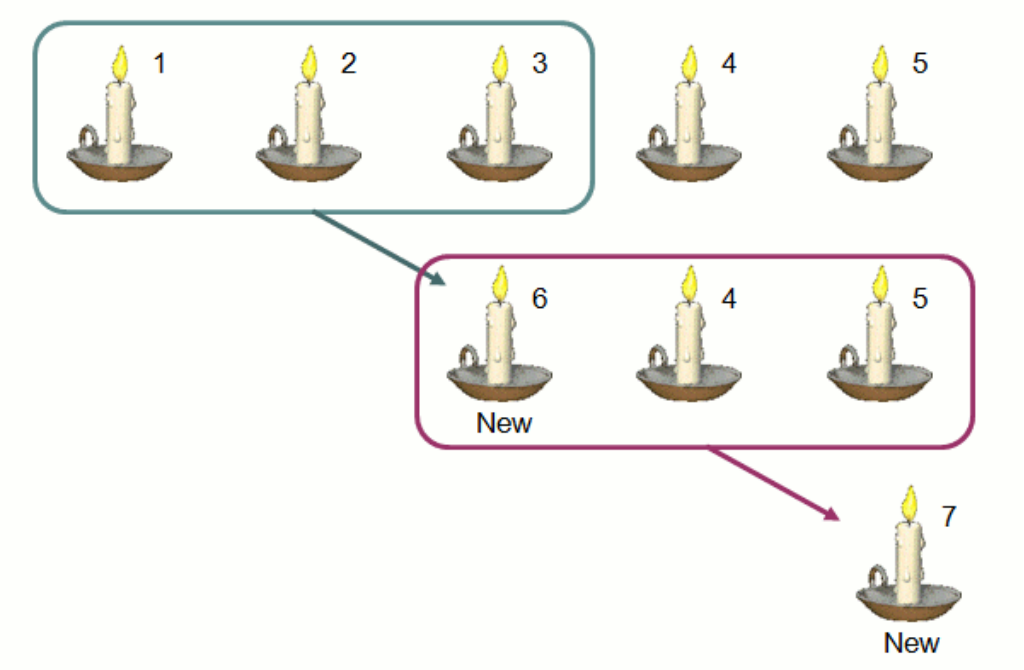

FIGURE 1 CANDLES

1.3 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.

Sample run #1:

```
$ 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
```

Sample run #2:

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

1.4 SKELETON PROGRAM AND TEST DATA

The skeleton program is provided: **candles.c**

Test input: **Input Files**

Test output: **Output Files**

1.5 IMPORTANT NOTES

- Write a function count_candles() to compute the total number of candles burnt. You need to determine the parameter(s) of the function.
- 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 readers (and yourself) to understand the logic better.

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): 15 minutes
- Checking/tracing the algorithm: 15 minutes
- Translating pseudo-code into code: 5 minutes
- Typing in the code: 5 minutes
- Testing and debugging: 20 minutes
- Total: 1 hour

2 EXERCISE 2: WHO ARE THE WINNERS

2.1 LEARNING OBJECTIVES

- Using selection and repetition statements.
- Problem solving.

2.2 TASK STATEMENT

Citizens of Zakadaha hold an annual Gagalafa festival to celebrate the harvest of their prized produce, the well-sought after cocoa beans Kokomoko. A lucky draw is held during the festival. Every participant is given a lucky draw number. Each year, the organizer decides on two non-zero digits, the factor-digit and the must-have-digit. These two digits need not be distinct.

A winning lucky draw number is a number that is a multiple of factor-digit as well as contains the must-have-digit.

In this exercise, you are to write a program winners.c to read in the following three inputs:

- The factor-digit, which is a non-zero digit (1 – 9).
- The must-have-digit, which is also a non-zero digit (1 – 9).
- The number of participants, n. Lucky draw numbers will be numbered from 1 to n inclusively.

You may assume that all inputs are valid.

For example, if factor-digit is 3, must-have-digit is 5, and the number of participants is 100, then the number of winners is 6 (the winning numbers are 15, 45, 51, 54, 57 and 75).

Your program is to count the number of winners whose lucky draw number is a multiple of factor-digit as well as contains the must-have-digit.

2.3 SAMPLE RUNS

Sample run using interactive input (user's input shown in blue; output shown in **bold purple**).

Sample run #1:

```
Enter factor-digit: 3
Enter must-have-digit: 5
Enter number of participants: 100
Number of winners: 6
```

Sample run #2:

```
Enter factor-digit: 9
Enter must-have-digit: 1
Enter number of participants: 15
Number of winners: 0
```

Sample run #3:

```
Enter factor-digit: 7
Enter must-have-digit: 7
Enter number of participants: 200
Number of winners: 5
```

2.4 SKELETON PROGRAM AND TEST DATA

The skeleton program is provided here: **winners.c**

Test input: **Input Files**

Test output: **Output Files**

2.5 IMPORTANT NOTES

- Write a function count_winners() to compute the total number of winners. You need to determine the parameter(s) of the function.
- You also need to write the precondition in the comment above the function definition.
- 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 readers (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
- Translating pseudo-code into code: 15 minutes
- Typing in the code: 10 minutes
- Testing and debugging: 30 minutes
- **Total: 1 hour 10 minutes**

3 EXERCISE 3: PRINT THE CALENDAR

3.1 LEARNING OBJECTIVES

- Using selection and repetition statements.
- Problem solving.

3.2 TASK STATEMENT

In this task, you are required to print the calendar of certain month given the year and month. To print correct calendar, you may need to consider several factors:

1. The number of days in the given month
2. The start day of the given month (whether 1st is Mon, Tue …).

The formula to get the start day of the given year is

$$\left(Year + \frac{Year-1}{4} - \frac{Year-1}{100} + \frac{Year-1}{400}\right)\%7$$

After getting the first day of the year, you need to add up all the days before that month to decide the first day of that month. (Hint: You need an array to store the number of days for each month. Be careful with the case of leap year.)

3. Good formatting of the calendar (Here you must obey the given printing formatting)

We may assume that:

1. The start day of the week is Sunday.
2. The year should be no earlier than 1900

Formatting:

1. The title of the calendar is given :"Sun Mon Tue Wed Thu Fri Sat", each day is separated by 2 blank spaces.
2. Each day number takes 3 position in right alignment. Each day number is separated with each other with by 2 blank spaces.

To make the things clearer, I put each character into the table, assume the 1st day is Monday, and the number of days in that month is 30.

Sun	Mon	Tue	Wed	Thu	Fri	Sat
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

Hint:

1. You should first write a function ***int is_leap_year ()*** to decide whether it is leap year, add parameters as necessary
2. Then you have to decide the first day in the week of that month. The correct starting day is the key part of the whole program. You are encouraged to explore ways to find the start day of the week given the date.
3. Write a function ***print_calendar()***, add parameters as necessary. You should pay attention to the given formatting of our printing process. You must obey the printing rules to get the full marks.
4. You are free to introduce additional functions if you deem it necessary. This must be supported by well-thought-out reasons, not a haphazard decision. By now, you should know that you cannot write a program haphazardly.
5. In writing functions, please put function prototypes before the main() function, and the function definitions after the main() function.

3.3 SAMPLE RUNS

Sample run using interactive input (user's input shown in blue; output shown in **bold purple**).

Sample run #1:

```
Enter a year: 2015
Enter a month: 1
Sun   Mon   Tue   Wed   Thu   Fri   Sat
                          1     2     3
  4     5     6     7     8     9    10
 11    12    13    14    15    16    17
 18    19    20    21    22    23    24
 25    26    27    28    29    30    31
```

Sample run #2:

```
Enter a year: 2008
Enter a month: 8
Sun   Mon   Tue   Wed   Thu   Fri   Sat
                                1     2
  3     4     5     6     7     8     9
 10    11    12    13    14    15    16
 17    18    19    20    21    22    23
 24    25    26    27    28    29    30
 31
```

3.4 SKELETON PROGRAM AND TEST DATA

The skeleton program is provided here: **calendar.c**

Test input: **input files**

Test output: **output files**

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): 40 minutes
- Translating pseudo-code into code: 15 minutes
- Typing in the code: 15 minutes
- Testing and debugging: 30 minutes
- Total: 100 minutes

4 DEADLINE

The deadline for submitting all programs is **16 Feb, 11:59pm, 2015**. Late submission will NOT be accepted.