APS 105 — Computer Fundamentals

Lab #5: Functions

Winter 2019

Important note: You must use the `submit` command to electronically submit your solution by the end of your lab session. Late submissions will not be accepted, and you will receive a grade of zero.

In this lab, you will be writing programs that use functions and pointers. This lab will be due in the your lab section in the week of February 11, 2019.

For each part in this lab, you should write your own `main` function to exercise the functions you have written with different parameter values. The functions should be by themselves in a file—there should not be a `main` function in the file.

Part 1 – Data Encryption

Data Encryption is the process of transforming data, using a cipher, to make it unreadable to anyone except those possessing special knowledge. In this problem you will implement a simple encryption technique on data that comprises of non-negative integers that have at least six digits with no leading zeroes. In order to encrypt the number the following functions are to be implemented:

```
void input(int *num);
int add4(int num);
int shift(int num);
void printOutput(int encryptNum, int originalNum);
```

The function `input` is used to read a positive integer having at least six digits and store it in the variable `num`. All leading zeroes in the input are neglected and are not counted as valid digits. The function will continue requesting for a number until a valid input is provided by the user.
A sample run of the function is shown below with data entered by the user displayed using **bold** font.

```
Please enter an integer greater than 99999: 2348<enter>
Incorrect input.
Please enter an integer greater than 99999: 012345<enter>
Incorrect input.
Please enter an integer greater than 99999: 102345<enter>
The number entered is 102345
```

The functions `add4` and `shift` are used to encrypt the number. First, modify the value of each digit of `num` by adding 4 to the digit. The function `add4` modifies the value of each digit of `num` by adding 4 to each digit, and returns the modified number. Due to addition, if the value of any digit is greater than 9 then subtract 10 from the number so that each digit ranges between 0 and 9. For example, if `num` is `345678` then `add4` should return `789012`. Note that the number of digits of the modified number returned by `add4` can be less than the number of digits in `num`. For example, if `num` is `665325`, then `add4` returns `009769`, which is really `9769`.

The input (num) of the function shift is the modified number (output of the function add4). The function shift shifts the position of each digit to the left by one place. The most significant digit becomes the least significant digit. For example, if the input (num) is 567890 then the output of the function shift is 678905. The modified value is returned by shift and is the final encrypted number. Similar to add4 function, the number of digits returned by shift can be less than the number of digits in num (input to function shift). For example, if num is 107982, then the encrypted number is 79821.

The function printOutput prints the original number (originalNum) and the encrypted number (encryptNum). Refer to the examples below for the output format.

You **are** allowed to use the math.h functions in your solution. Remember to add -lm to your gcc command line.

Shown below are example runs of the program with data entered by the user displayed using **bold** font. Given the same user input, your output should match the output *exactly*, including all punctuation. Any variation from this will result in loss of marks.

- **Example 1:**

```
Please enter an integer greater than 99999: 4569012<enter>
The number entered is 4569012
Original number: 4569012
Encrypted number: 9034568
```

- **Example 2:**

```
Please enter an integer greater than 99999: 563918<enter>
The number entered is 563918
Original number: 563918
Encrypted number: 73529
```

- **Example 3:**

```
Please enter an integer greater than 99999: 70912<enter>
Incorrect input.
Please enter an integer greater than 99999: 093679<enter>
Incorrect input.
Please enter an integer greater than 99999: 0877893<enter>
The number entered is 877893
Original number: 877893
Encrypted number: 112372
```

Note: The encrypted number can have less than 6 digits (as shown in Example 2). It is up to the decryption technique to handle this (not part of this problem). In the case of Example 3, 0877893 is a valid input as it has 6 digits without any leading zero (877893).

When you are ready to submit your work, place all of your functions in a file named Lab5Part1.c (with no main function) for submission.

Part 2 – Craps

In this problem you will implement the popular dice game *craps*. The following is a brief description of how the game is played.

The game of craps is played with two dice. On the first roll, the player wins if the sum of the dice is 7 or 11. The player loses if the sum is 2, 3, or 12. Any other roll is called a "point" and the game continues. On each subsequent roll, the player wins if he or she rolls the point again. The player loses by rolling 7. Any other roll is ignored and the game continues. At the end of the game the user has a choice to start a new game by pressing `y` or `Y`. Pressing any other key will display the number of wins and losses and then exit the program. A sample run of the program is shown below with data entered by the user displayed using **bold** font. Your program output will look exactly as below when you run your program in one of the ECF machines with the matching user inputs.

```
You rolled: 7
You win!
Play again? y<enter>
You rolled: 6
Your point is: 6
You rolled: 8
You rolled: 6
You win!
Play again? y<enter>
You rolled: 6
Your point is: 6
You rolled: 5
You rolled: 5
You rolled: 11
You rolled: 2
You rolled: 10
You rolled: 9
You rolled: 8
You rolled: 6
You win!
Play again? Y<enter>
You rolled: 5
Your point is: 5
You rolled: 4
You rolled: 7
You lose!
Play again? n<enter>

Wins: 3
Losses: 1
```

Write the following functions with the corresponding prototypes.

```
int rollDice(void);
bool playGame(void);
void winLoss(void);
```

The function `rollDice` should generate two random numbers, each between 1 and 6, and return their sum. Use the `rand` function for rolling each dice to obtain a number between `1` and `6`, inclusive. You are *not* allowed to use the function `srand()`. If you use it, you will receive zero marks for this problem. The function `rollDice` is used to determine the outcome of each dice roll.

The function `playGame` should play one craps game. It returns `true` if the player wins and `false` if the player loses. For every game of crap, the function `playGame` calls the function `rollDice` one or more times, depending upon the value returned by `rollDice`. `playGame` is also responsible for displaying messages showing results of the player's dice rolls.

The function `winLoss` keeps track of the number of wins and losses in one session. It repeatedly calls the function `playGame` until the user requests to end the program. `winLoss` should also report the number of wins and losses before it ends.

When you are ready to submit your work, place all of your functions in a file named `Lab5Part2.c` (with no `main` function) for submission.

Part 3 – Zeller's Algorithm

In 1883, German mathematician Christian Zeller devised an algorithm, popularly known as Zeller's congruence, to calculate the day of the week on which a given date will fall or fell. In this exercise you will implement the following two functions:

```
void inputDate(int *day, int *month, int *year);
void calculateDay(int day, int month, int year);
```

The user must enter the date as `dd/mm/yyyy` and the function `inputDate` stores it in the variables `day`,`month` and `year`. Assume that the user enters a valid date.

The function `calculateDay` will calculate the day of the week and print it. First, the month of the year must be transformed as an integer between 1–12 where March is 1 and February is 12. Thus, if the month is Jan or Feb, the year must also be modified to the previous year (see examples below). Zeller's algorithm is defined as follows:

Let A, B, C, D denote integer variables that have the following values:

A = the month of the year, with March having the value 1, April the value 2,. . . , December the value 10, and January and February being counted as months 11 and 12 of the preceding year (in which case, subtract 1 from C)
B = the day of the month (1, 2, 3,. . . , 30, 31)
C = the year of the century (e.g. C = 89 for the year 1989)
D = the century (e.g. D = 19 for the year 1989)

The following examples show the values of A, B, C, D for a given date.

- **Example 1:** For input date 5/12/2006

  A = 10, B = 5, C = 6, D = 20

- **Example 2:** For input date 03/02/1974

  A = 12, B = 3, C = 73, D = 19

- **Example 3:** For input date 8/1/2000

  A = 11, B = 8, C = 99, D = 19

- **Example 4:** For input date 18/3/2020

  A = 1, B = 18, C = 20, D = 20

Let W, X, Y, Z, R also denote integer variables. Compute their values in the following order using integer arithmetic:

W = (13*A - 1) / 5
X = C / 4
Y = D / 4
Z = W + X + Y + B + C - 2*D
R = the remainder when Z is divided by 7

If the value of R is negative, then add 7 to it. At the end of the computation R is a number between 0 and 6 and is the day of the week, where 0 represents Sunday, 1 is Monday,. . . , 6 is Saturday. Shown below are example runs of the program. Given the same user input, your output should match the output *exactly*, including all punctuation. Any variation from this will result in loss of marks.

- **Example 1:**

```
Please enter a date: 10/10/2012<enter>
The day 10/10/2012 is a Wednesday.
```

- **Example 2:**

```
Please enter a date: 05/02/1985<enter>
The day 5/2/1985 is a Tuesday.
```

When you are ready to submit your work, place all of your functions in a file named `Lab5Part3.c` (with no `main` function) for submission.

Submission Instructions

There are ten (10) marks available in this lab.

TA Grading (4 marks)

Your solution will be marked by a Teaching Assistant during your scheduled lab period for programming style and your understanding of the code. You can discuss programming style with your TA and on Piazza. Here are some quick guidelines:

- Good choices for variable names that indicate their purpose.
- A consistent naming convention. Use camelCase for normal variable names.
- Comments that explain code that is difficult to understand.
- Proper indentation.
- Appropriate white space between lines for better readability.
- A limited number of, or ideally zero, global variables.

Automated Grading (6 marks)

Your solution must be submitted electronically by the end of your scheduled lab period. Submission of the lab requires you to use a terminal. In the terminal, you must:

1. Go to the directory that contains `Lab5Part1.c`, `Lab5Part2.c`, and `Lab5Part3.c` (i.e., use the `cd` command in the terminal).

2. Type in the following command: `/share/copy/aps105s/lab5/submit`

This command will run an exercise program that will check to make sure everything looks okay. If it finds a problem, it will ask you if you are sure that you want to submit.
Note that you may submit your work as many times as you want prior to the deadline; only the most recent submission is marked. You can also run the exerciser on your own with the following command:
`/share/copy/aps105s/lab5/exercise`
Finally, you can also check to see if what you think you have submitted is actually there, for peace of mind, using the following command:
`/share/copy/aps105s/lab5/viewsubmitted`

Obtaining Your Automated Grade

After all lab sections have finished, a short time later, you will be able to run the auto-marker to determine the auto-marked fraction of your grade on the code you have submitted. To do so run the following command:
`/share/copy/aps105s/lab5/marker`
This command will compile and run your code and test it with all the test cases used to determine the auto-mark grade. You will be able to see those test cases' output and what went right or wrong.