COMP1511 18s2

Week-03 Laboratory Exercises

Objectives

- input and output of `numerical` values
- implementing simple numerical calculations
- using complex if statements to control program execution
- creating simple functions
- using a while loop for repetition

Topics

Preparation

Before the lab you should re-read the relevant lecture slides and their accompanying examples. You should also have read the lab assessment guidelines. We will assess Exercises 00 to 08 to derive marks for this lab. However, you will learn a lot by attempting and solving the challenge exercises.

Getting Started

Create a new directory for this lab called `lab03` by typing:

```
$ mkdir lab03
```

Change to this directory by typing:

```
$ cd lab03
```

Exercise - 00: Confusing Code Makeover (pair)

You should try to complete this exercise outside your lab time, so you have more time for the rest of the exercises in this lab. Please note that this exercise will be assessed. This is a pair exercise to complete with your lab partner.

During the tutorial this week, your tutor will show you how to conduct a code review on the following code:

- **confusing.c**

For this lab, you will need to download a copy of the above file and fix it so that it works as expected and complies with the course **style guide**.

Try and figure out what this program is meant to do. Come up with some *test cases* you can use to make sure the code works as expected. Once you have some test cases, see if the program works properly.

Once you have some tests, you will need to fix up the code so that is complies with the course **style guide** and so it passes your tests. How beautiful can you make the code?

When you fix the header comment, make sure to put your and your partner's name in, but don't remove Julian's.

As you fix up the code, keep some notes about what you have had to fix, why it needed fixing, and how you fixed it. Add these comments at the end of your "confusing.c" file as comments.

When you are finished on this exercise you and your lab partner must both submit your work by running **give**:

```
$ give cs1511 wk03_confusingCode confusing.c
```

Note, even though this is a pair exercise, you both must run **give** from your own account before **Sunday 12 August 23:59:59** to obtain the marks for this lab exercise.

Exercise - 01: Ordering Three Integers (pair)

This is a pair exercise to complete with your lab partner.

Write a C program `order3.c` using **if** statements (no loops) that reads 3 integers and prints them from smallest to largest.

Your program should behave exactly like this example:

```
$ ./order3
Enter integer: 23
Enter integer: 5
Enter integer: 27
The integers in order are: 5 23 27
$ ./order3
Enter integer: 3
Enter integer: 6
Enter integer: 27
The integers in order are: 3 6 27
$ ./order3
Enter integer: 9
Enter integer: 8
Enter integer: 7
The integers in order are: 7 8 9
```

You can assume the user supplies 3 integers. You do not have to check the return value from scanf.
When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 1511 autotest order3
```

When you are finished on this exercise you and your lab partner must both submit your work by running **give**:

```
$ give cs1511 wk03_order3 order3.c
```

Note, even though this is a pair exercise, you both must run **give** from your own account before **Sunday 12 August 23:59:59** to obtain the marks for this lab exercise.

Exercise - 02: Is this a leap year? (pair)

This is a pair exercise to complete with your lab partner.

Write a C program `is_leap_year.c` that reads a year and then prints whether that year is a leap year.

Match the examples below exactly

Hint: you only need use the **int** type, modulus (**%**) and **if** statement(s).

For example:

```
$ dcc -o is_leap_year is_leap_year.c
$ ./is_leap_year
Enter year: 2017
2017 is not a leap year.
$ ./is_leap_year
Enter year: 2016
2016 is a leap year.
$ ./is_leap_year
Enter year: 2000
2000 is a leap year.
$ ./is_leap_year
Enter year: 3000
3000 is not a leap year.
```

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 1511 autotest is_leap_year
```

When you are finished on this exercise you and your lab partner must both submit your work by running **give**:

```
$ give cs1511 wk03_is_leap_year is_leap_year.c
```

Note, even though this is a pair exercise, you both must run **give** from your own account before **Sunday 12 August 23:59:59** to obtain the marks for this lab exercise.

Exercise - 03: Is this a leap year? (pair)

This is a pair exercise to complete with your lab partner.

Write a C program `leap_year_function.c` that reads a year and then **uses a function** to calculate whether that year is a leap year. The function prototype must be `int isLeapYear(int year)`.

Your function should return 0 if it is not a leap year, and 1 if it is a leap year.

Your leap year function must not print anything.

You should call this function from your main function, which is where you print the result.

Your function must be named **isLeapYear**.
It must *exactly* match the function prototype given above.

Match the examples below exactly

Hint: copy the logic from your `is_leap_year.c` and put it into a function

For example:

```
$ dcc -o leap_year_function leap_year_function.c
$ ./leap_year_function
Enter year: 2017
2017 is not a leap year.
$ ./leap_year_function
Enter year: 2016
2016 is a leap year.
$ ./leap_year_function
Enter year: 2000
2000 is a leap year.
$ ./leap_year_function
Enter year: 3000
3000 is not a leap year.
```

Beware: autotest will not use your **main** function. It will call your **isLeapYear** function here.

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 1511 autotest leap_year_function
```

When you are finished on this exercise you and your lab partner must both submit your work by running **give**:

```
$ give cs1511 wk03_leap_year_function leap_year_function.c
```

Note, even though this is a pair exercise, you both must run **give** from your own account before **Sunday 12 August 23:59:59** to obtain the marks for this lab exercise.

Exercise - 04: Fun facts about Circles (pair)

This is a pair exercise to complete with your lab partner.

Here is a program circle_facts.c which calculates some fun facts about circles.
Unfortunately it is incomplete. Your task is to complete it.

Its **main** function is complete. Do not **main** function change. The three functions to change are>

```
double area(double radius);
double circumference(double radius);
double diameter(double radius);
```

Hint use the constant `M_PI` defined in `math.h`

```
$ dcc -o circle_facts  circle_facts.c
$ ./circle_facts
Enter circle radius: 1
Area          = 3.141593
Circumference = 6.283185
Diameter      = 2.000000
$ ./circle_facts
Enter circle radius: 17
Area          = 907.920277
Circumference = 106.814150
Diameter      = 34.000000
$ ./circle_facts
Enter circle radius: 0.0125
Area          = 0.000491
Circumference = 0.078540
Diameter      = 0.025000
```

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 1511 autotest circle_facts
```

When you are finished on this exercise you and your lab partner must both submit your work by running **give**:

```
$ give cs1511 wk03_circle_facts circle_facts.c
```

Note, even though this is a pair exercise, you both must run **give** from your own account before **Sunday 12 August 23:59:59** to obtain the marks for this lab exercise.

Exercise - 05: Countdown (individual)

We use loops in C to do things multiple times. Here is an example of a loop that prints all the numbers from 1 to 17 on a new line in ascending order:

```
int counter = 1; //initialise counter to 1
while (counter <= 17) { // loop until not <= 17
    printf("%d\n", counter); // print counter
    counter = counter + 1; // increment counter
}
```

In this exercise you will use a loop to print a countdown from 10 to 0. Start by creating a file called `countdown.c` in your week 4 directory, and copying the above code. Modify this code so that the loop counts down from 10 until 0.

Example

```
$ ./countdown
10
9
8
7
6
5
4
3
2
1
0
```

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 1511 autotest countdown
```

When you are finished on this exercise you and your lab partner must both submit your work by running **give**:

```
$ give cs1511 wk03_countdown countdown.c
```

Note, you must run **give** from your own account before **Sunday 12 August 23:59:59** to obtain the marks for this lab exercise.

Exercise - 06: Three or Five (pair)

This is a pair exercise to complete with your lab partner.

Write a program that **three_five.c** that reads a positive integer **n** and print all the positive integers **< n** divisible by **3** or **5**.
For example:

```
$  ./three_five
Enter number:  10
3
5
6
9
$ ./three_five
Enter number:  30
3
5
6
9
10
12
15
18
20
21
24
25
27
```

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 1511 autotest three_five
```

When you are finished on this exercise you and your lab partner must both submit your work by running **give**:

```
$ give cs1511 wk03_three_five three_five.c
```

Note, even though this is a pair exercise, you both must run **give** from your own account before **Sunday 12 August 23:59:59** to obtain the marks for this lab exercise.

Exercise - 07: Detect outliers (individual)

This is a pair exercise to complete with your lab partner.

Write a program called `outliers.c` that reads integer values from standard input until the end of input stream or first unsuccessful read. The program finds number of values that are outside the range of 5 to 25 (inclusive), and prints the number of such outliers.

Hint: use "scanf" to terminate "while" loop, see "**while Termination : scanf example**" from the Week-02 lecture on "Iterations using while".

Make your program match the examples below exactly.

```
$ ./outliers
Enter number: 12
Enter number: 4
Enter number: 25
Enter number: 15
Enter number: 27
Enter number: Ctrl + D
Outliers: 2


$ ./outliers
Enter number: 0
Enter number: 26
Enter number: 5
Enter number: -2
Enter number: Ctrl + D
Outliers: 3
```

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 1511 autotest outliers
```

When you are finished on this exercise you and your lab partner must both submit your work by running **give**:

```
$ give cs1511 wk03_outliers outliers.c
```

Note, you both must run **give** from your own account before **Sunday 12 August 23:59:59** to obtain the marks for this lab exercise.

Exercise - 07: X Factor (pair)

This is a pair exercise to complete with your lab partner.

Write a program called `x.c` that reads an integer **n** from standard input. and prints an **n**x**n** pattern of asterisks and dashes in the shape of an "**X**".
You can assume *n* is odd and **>= 5**.

Make your program match the examples below exactly.

You are not permitted to use an array in this exercise.

```
$ ./x
Enter size: 5
*---*
-*-*-
--*--
-*-*-
*---*
$ ./x
Enter size: 9
*-------*
-*-----*-
--*---*--
---*-*---
----*----
---*-*---
--*---*--
-*-----*-
*-------*
$ ./x
Enter size: 15
*-------------*
-*-----------*-
--*---------*--
---*-------*---
----*-----*----
-----*---*-----
------*-*------
-------*-------
------*-*------
-----*---*-----
----*-----*----
---*-------*---
--*---------*--
-*-----------*-
*-------------*
```

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 1511 autotest x
```

When you are finished on this exercise you and your lab partner must both submit your work by running **give**:

```
$ give cs1511 wk03_x x.c
```

Note, even though this is a pair exercise, you both must run **give** from your own account before **Sunday 12 August 23:59:59** to obtain the marks for this lab exercise.

Exercise - 08: Boxes (individual) [Challenge]

This is an individual exercise to complete by yourself.

For this challenge, make a program called `boxes.c` which reads in a number and then draws that many square boxes inside each other using the character #.

For example:

```
$ ./boxes
How many boxes: 1
###
# #
###
```

```
$ ./boxes
How many boxes: 2
#######
#     #
# ### #
# # # #
# ### #
#     #
#######
```

```
$ ./boxes
How many boxes: 5
###################
#                 #
# ############### #
# #             # #
# # ########### # #
# # #         # # #
# # # ####### # # #
# # # #     # # # #
# # # # ### # # # #
# # # # # # # # # #
# # # # ### # # # #
# # # #     # # # #
# # # ####### # # #
# # #         # # #
# # ########### # #
# #             # #
# ############### #
#                 #
###################
```

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 1511 autotest boxes
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs1511 wk03_boxes boxes.c
```

You must run give before **Sunday 12 August 23:59:59** to obtain the marks for this lab exercise. Note, this is an individual exercise, the work you submit with **give** must be entirely your own.

Exercise - 09: Spiral (individual) [Challenge]

This is an individual exercise to complete by yourself.

Write a program called `spiral.c` that reads an integer **n** from standard input. and prints an **n**x**n** pattern of asterisks and dashes in the shape of a spiral.
You can assume *n* is odd and **>= 5**.

You are only permitted to use C language features covered in weeks 1-3 lectures. In particular, you are not permitted to use array(s).

Make your program match the examples below exactly.

You are not permitted to use an array in this exercise.

```
$ ./spiral
Enter size: 5
*****
----*
***-*
*---*
*****
$ ./spiral
Enter size: 7
*******
------*
*****-*
*---*-*
*-***-*
*-----*
*******
$ ./spiral
Enter size: 9
*********
--------*
*******-*
*-----*-*
*-***-*-*
*-*---*-*
*-*****-*
*-------*
*********
$ ./spiral
Enter size: 17
*****************
----------------*
***************-*
*-------------*-*
*-***********-*-*
*-*---------*-*-*
*-*-*******-*-*-*
*-*-*-----*-*-*-*
*-*-*-***-*-*-*-*
*-*-*-*---*-*-*-*
*-*-*-*****-*-*-*
*-*-*-------*-*-*
*-*-*********-*-*
*-*-----------*-*
*-*************-*
*---------------*
*****************
```

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 1511 autotest spiral
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs1511 wk03_spiral spiral.c
```

You must run give before **Sunday 12 August 23:59:59** to obtain the marks for this lab exercise. Note, this is an individual exercise, the work you submit with **give** must be entirely your own.

Exercise - 10 : Decimal Spiral (individual - attempt if you dare) [Extra-hard Challenge]

This is an individual exercise to complete by yourself.

Write a program called `decimal_spiral.c` that reads an integer **n** from standard input. and prints an **n**x**n** pattern of decimal digits and dashes in the shape of a spiral.
You can assume *n* is odd and **>= 5**.

You are only permitted to use C language features covered in weeks 1-3 lectures. In particular, you are not permitted to use array(s).

Make your program match the examples below exactly.

You are not permitted to use an array in this exercise.

```
$ ./decimal_spiral
Enter size:  5
65432
----1
210-0
3---9
45678
$ ./decimal_spiral
Enter size:  7
0987654
------3
87654-2
9---3-1
0-012-0
1-----9
2345678
$ ./decimal_spiral
Enter size:  9
876543210
--------9
8765432-8
9-----1-7
0-210-0-6
1-3---9-5
2-45678-4
3-------3
456789012
$ ./decimal_spiral
Enter size:  15
654321098765432
--------------1
2109876543210-0
3-----------9-9
4-210987654-8-8
5-3-------3-7-7
6-4-87654-2-6-6
7-5-9---3-1-5-5
8-6-0-012-0-4-4
9-7-1-----9-3-3
0-8-2345678-2-2
1-9---------1-1
2-01234567890-0
3-------------9
456789012345678
```

When you think your program is working you can use `autotest` to run some simple automated tests:

```
$ 1511 autotest decimal_spiral
$ 1511 autotest decimal_spiral
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs1511 wk03_decimal_spiral decimal_spiral.c
```

You must run give before **Sunday 12 August 23:59:59** to obtain the marks for this lab exercise. Note, this is an individual exercise, the work you submit with **give** must be entirely your own.

Submission

When you are finished each exercises make sure you submit your work by running **give**.
You can run **give** multiple times. Only your last submission will be marked.

Don't submit any exercises you haven't attempted.

If you are working at home, you may find it more convenient to upload your work via give's web interface.

Remember you have until **Sunday 12 August 23:59:59** to submit your work.

Automarking will be run several days after the submission deadline for the test. When complete you can view automarking here and you can view the the resulting mark via give's web interface

You can read more about lab assessment here