COMP1511 18s2

Week-02 Laboratory Exercises

Objectives

- using C input/output (I/O) facilities
- creating simple arithmetic expressions
- programming with `if` statements
- creating relational expressions
- displaying varying strings

Topics

Preparation

Before the lab you should re-read the relevant lecture slides and their accompanying examples.

Getting Started

Create a new directory for this lab called `lab02` by typing:

```
$ mkdir lab02
```

Change to this directory by typing:

```
$ cd lab02
```

Exercise - 01: Don't Be So Negative! (pair)

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

Let's start by creating a `lab02` directory and moving into it like so:

```
$ mkdir lab02
$ cd lab02
```

Now create and open a new file called `negative.c` for this exercise.

```
$ gedit negative.c &
```

Write a program that uses `scanf` to get a number from a user and prints "Don't be so negative!" if they entered a negative number.
If the number is positive, the program should print "You have entered a positive number."
If the user enters the number 0, the program should print "You have entered zero."
Your program should behave as follows:

```
$ dcc -o negative negative.c
$ ./negative
3
You have entered a positive number.
$ ./negative
-3
Don't be so negative!
$ ./negative
0
You have entered zero.
```

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

```
$ 1511 autotest negative
```

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

```
$ give cs1511 wk02_negative negative.c
```

Note, even though this is a pair exercise, you both must run **give** from your own account before **Sunday 05 Aug 23:59:59** to obtain the marks for this lab exercise.

Exercise - 02: Icecream Scoops (pair)

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

Create and open a new file called `icecream.c` for this exercise.

```
$ gedit icecream.c &
```

Matilda wants to buy some ice-cream, but she only has $10. Write a program so that she can input how many scoops of ice-cream she wants and how much each scoop costs and it will let her know if she has enough money.

Your program should behave as follows:

```
$ dcc -o icecream icecream.c
$ ./icecream
How many scoops? 5
How many dollars does each scoop cost? 1
You have enough money!
$ ./icecream
How many scoops? 5
How many dollars does each scoop cost? 3
Oh no, you don't have enough money :(
```

You can assume that Matilda will only give you positive integers.

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

```
$ 1511 autotest icecream
```

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

```
$ give cs1511 wk02_icecream icecream.c
```

Note, even though this is a pair exercise, you both must run **give** from your own account before **Sunday 05 Aug 23:59:59** to obtain the marks for this lab exercise.

Exercise - 03: Addition (pair)

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

Create a program called `addition.c`.

This program should ask for two integers using the message `Please enter two integers:` and then display the sum of the integers as *`n + n = sum`*.

Make sure to replace the ***n*** with the numbers entered in the same order and the ***sum*** with the sum of the two numbers.

Some Examples

```
$ ./addition
Please enter two integers: 2 5
2 + 5 = 7
```

```
$ ./addition
Please enter two integers: 3 5
3 + 5 = 8
```

```
$ ./addition
Please enter two integers: -1 5
-1 + 5 = 4
```

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

```
$ 1511 autotest addition
```

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

```
$ give cs1511 wk02_addition addition.c
```

Note, even though this is a pair exercise, you both must run **give** from your own account before **Sunday 05 Aug 23:59:59** to obtain the marks for this lab exercise.

Exercise - 04: Numbers to Words (pair)

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

Make a program called `numberWords.c`.

This program will ask for a number with the message `Please enter an integer:` .

For numbers between 1 and 5, display the number as a word in the message `You entered` *`number`* .

For numbers less than 1, display the message `You entered a number less than one.`

For numbers greater than 5, display the message `You entered a number greater than five.`

Some Examples

```
$ ./numberWords
Please enter an integer: 2
You entered two.
```

```
$ ./numberWords
Please enter an integer: 5
You entered five.
```

```
$ ./numberWords
Please enter an integer: 0
You entered a number less than one.
```

```
$ ./numberWords
Please enter an integer: 1000
You entered a number greater than five.
```

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

```
$ 1511 autotest numberWords
```

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

```
$ give cs1511 wk02_numberWords numberWords.c
```

Note, even though this is a pair exercise, you both must run **give** from your own account before **Sunday 05 Aug 23:59:59** to obtain the marks for this lab exercise.

Exercise - 05: Dating Range (individual) [Challenge Exercise]

This is an individual exercise to complete by yourself.

A popular rule for dating is that you should not date anyone younger than half your age + 7. COMP1511 does not endorse this rule and is not responsible if it leads to you having a bad date.

Write a C program `dating_range.c` that reads a person's age and calculates the upper and lower age limits of people they should date according to this rule.

Hint: you'll need to use symmetry to calculate the upper bound.

Hint: you only need to use the `int` type.

For example:

```
$ dcc -o dating_range dating_range.c
$ ./dating_range
Enter your age: 18
Your dating range is 16 to 22 years old.
$ ./dating_range
Enter your age: 22
Your dating range is 18 to 30 years old.
$ ./dating_range
Enter your age: 40
Your dating range is 27 to 66 years old.
```

The above rule implies an empty range for young ages. When the rule produces an empty range program should behave like this:

```
$ ./dating_range
Enter your age: 12
You are too young to be dating.
```

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

```
$ 1511 autotest dating_range
```

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

```
$ give cs1511 wk02_dating_range dating_range.c
```

You must run give before **Sunday 05 Aug 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 - 06 : Easter (individual) [Challenge Exercise]

This is an individual exercise to complete by yourself.

Write a program `easter.c` which allows the user to enter a year, then calculates the date of Easter Sunday for that year. Use the formula developed in 1876 by Samuel Butcher, Bishop of Meath,.

Follow the output format in in the example below **exactly**:

```
$ dcc easter.c -o easter
$ ./easter
Enter Year: 2017
Easter is April 16 in 2017.
$ ./easter
Enter Year: 2018
Easter is April 1 in 2018.
$ ./easter
Enter Year: 2019
Easter is April 21 in 2019.
```

Hints

Cut-and-paste the formula from the above Web site and then fill in the C program around it.
Make sure every variable is declared.

Make sure every statement ends with a semicolon.

> Note that the original proposal of this formula had only single letter variable names, and no explanation of how it works.
>
> Because of this, even though we know *that* it works, no-one knows *how* it works.
>
> Make sure to always comment your code and have sensible variable names so that people can understand how your code works!

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

```
$ 1511 autotest easter
```

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

```
$ give cs1511 wk02_easter easter.c
```

You must run give before **Sunday 05 Aug 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 05 Aug 23:59:59** to submit your work.

Automarking will be run several days after the submission deadline for the test. Later we will inform you how to view automarking and resulting mark.