COMP1511 18s2

Week-04 Laboratory Exercises

Objectives

- creating functions
- manipulating arrays (Single and Multi dimensional)
- using a while loop for repetition

Topics

Preparation

Before the lab you should re-read the relevant lecture slides and their accompanying examples. **We will assess Exercises 01, 04, 05, 06 and 07 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 `lab04` by typing:

```
$ mkdir lab04
```

Change to this directory by typing:

```
$ cd lab04
```

Exercise - 00: Complex Absolute (warmup: individually or pair)

This is a warmup exercise. It is not compulsory, and may be completed individually or with your lab partner.

If you are not familiar with *complex numbers*, watch the following videos on Khan Academy and/or read the wikipedia entry:

- video: Intro to complex numbers
- video: Adding complex numbers
- video: Subtracting complex numbers
- video: Multiplying complex numbers
- on wikipedia: Complex Number

In this activity you need to complete the function `complexAbsolute` in the provided program.

Download `complexAbs.c`, or copy it into your current directory on a CSE system by running

```
$ cp /web/cs1511/18s2/files/lab/04/complexAbs.c .
```

In this exercise you have been provided with a `struct` to represent a Complex Number, called `complex`. Read the provided file and understand the given data structure.

You have also been given a `main` function and an incomplete `double complexAbsolute(complex c)`. You must edit this function so that it returns the magnitude/absolute value of the complex number it has been given.

The formula for the magnitude of a complex number $z = x + iy$ is

$$r = |z| = \sqrt{x^2 + y^2}$$

We've `#include`d a new header file, `<math.h>`. This declares a function, `double sqrt(double x)`, which returns the square root of `x`.

Some Examples

```
Enter the real part: 3
Enter the imaginary part: 4
The absolute value is 5.00.
```

```
Enter the real part: 17
Enter the imaginary part: 17
The absolute value is 24.04.
```

To run some simple automated tests:

```
$ 1511 autotest complexAbs
```

We have set up a tool to guide you with your programming style. To run it:

```
$ 1511 style complexAbs.c
Looks good!
```

You'll get advice if you need to make changes to your code. Ask your tutor for help if you're not sure.

Submit your work with the *give* command, like so:

```
$ give cs1511 wk04_complexAbs
```

Or, if you are working from home, upload the relevant file(s) to the `wk04_complexAbs` activity on Give Online.

Exercise - 01: Complex Functions (pair)

This is a pair exercise to complete with your lab partner.
You should make sure you have completed the activity "Complex Absolute" before completing this task.

In this activity you need to complete the functions `complexAdd`, `complexMultiply`, and `complexSquare`. in the provided program.

Download `complexFunctions.c`, or copy it into your current directory on a CSE system by running

```
$ cp /web/cs1511/18s2/files/lab/04/complexFunctions.c     .
```

In this exercise you have been provided with a struct to represent a Complex Number, called `complex`. Read the provided file and understand the given data structure.

You have also been given a `main` function and three incomplete functions:

```
complex complexAdd(complex a, complex b);
complex complexMutiply(complex a, complex b);
complex complexSquare(complex c);
```

`complexAdd` takes in two complex numbers and must return a complex number that is the sum of these two numbers. Adding two complex numbers is achieved by adding the real and imaginary parts independently, like so:

$$(a + bi) + (c + di) = (a + c) + (b + d)i$$

Similarly, `complexMultiply` takes in two complex numbers and must return a complex number that is the product of these two numbers. The formula to multiple two complex numbers is as follows:

$$(a + bi)(c + di) = (ac - bd) + (bc + ad)i$$

Finally, `complexSquare` takes in one complex number and must return its square.

We've `#include`d a new header file, `<math.h>`. This declares a function, `double sqrt(double x)`, which returns the square root of `x`.

An Example:

```
Enter the real part of the first number: 1
Enter the imaginary part of the first number: 17
Enter the real part of the second number: 2
Enter the imaginary part of the second number: 3
The sum is 3.00 + 20.00i.
The product is -49.00 + 37.00i.
The square of the first number is -288.00 + 34.00i.
```

To run some simple automated tests:

```
$ 1511 autotest complexFunctions
```

We have set up a tool to guide you with your programming style. To run it:

```
$ 1511 style complexFunctions.c
Looks good!
```

You'll get advice if you need to make changes to your code. Ask your tutor for help if you're not sure.

Submit your work with the *give* command, like so:

```
$ give cs1511 wk04_complexFunctions
```

Or, if you are working from home, upload the relevant file(s) to the `wk04_complexFunctions` activity on Give Online.

Exercise - 02: First, Middle, Last (warmup)

This is a warmup exercise. It is not compulsory, and may be completed individually or with your lab partner.

Download `firstMiddleLast.c`, or copy it into your current directory on a CSE system by running

```
$ cp /web/cs1511/18s2/files/lab/04/firstMiddleLast.c   .
```

The file `firstMiddleLast.c` contains a main function which reads values into an array. You are to implement the `void printFirstMiddleLast(int size, int array[MAX_LENGTH])` function which should print the first, middle, and last elements of the array, each on a new line.

The middle element of the array is the one with index `size/2`.

Some Examples:

```
Enter array size: 5
Enter array values: 1 2 3 4 5
1
3
5
```

```
Enter array size: 6
Enter array values: 1 2 3 4 5 6
1
4
6
```

To run some simple automated tests:

```
$ 1511 autotest firstMiddleLast
```

We have set up a tool to guide you with your programming style. To run it:

```
$ 1511 style firstMiddleLast.c
Looks good!
```

You'll get advice if you need to make changes to your code. Ask your tutor for help if you're not sure.

Submit your work with the *give* command, like so:

```
$ give cs1511 wk04_firstMiddleLast
```

Or, if you are working from home, upload the relevant file(s) to the `wk04_firstMiddleLast` activity on Give Online.

Exercise - 03: Show Array (warmup)

This is a warmup exercise. It is not compulsory, and may be completed individually or with your lab partner.

Download showArray.c, or copy it into your current directory on a CSE system by running

```
$ cp /web/cs1511/18s2/files/lab/04/showArray.c   .
```

The file `showArray.c` contains a main function which reads values into an array. You are to implement the `void showArray(int size, int array[MAX_LENGTH])` function which should print the given array like so:

```
[1, 2, 3]
```

Some Examples:

```
Enter array size: 3
Enter array values: 1 2 3
[1, 2, 3]
```

```
Enter array size: 1
Enter array values: 17
[17]
```

To run some simple automated tests:

```
$ 1511 autotest showArray
```

We have set up a tool to guide you with your programming style. To run it:

```
$ 1511 style showArray.c
Looks good!
```

You'll get advice if you need to make changes to your code. Ask your tutor for help if you're not sure.

Submit your work with the *give* command, like so:

```
$ give cs1511 wk04_showArray
```

Or, if you are working from home, upload the relevant file(s) to the `wk04_showArray` activity on Give Online.

Exercise - 04: Array Average (individually)

You need to complete this exercise individually.

Download the following two files:

- arrayAverage.c
- test_arrayAverage.c

You need to implement the following function that calculates the average of all the values in a given array `a` (of type `int`), and returns the average value (of type `double`).

```
double arrayAverage(int a[], int size)
```

For example,

```
arrayAverage([1, 2, 3], 3)          returns   2.0
arrayAverage([5, 2, 6, 1], 4)       returns   3.5
arrayAverage([1, 2, 6, 1, 1], 5)    returns   2.2
```

For this question, you need to use the provided file `test_arrayAverage.c` to test your function. Please open the file `test_arrayAverage.c` using your favourite text editor, try to understand how your function is tested. If you have any questions, please ask your tutor.

You can use the following commands to compile and run few tests already provided in the file `test_arrayAverage.c`. Please make sure that you also test your function extensively by adding more test cases in `test_arrayAverage.c`. To run the simple tests available in `test_arrayAverage.c`, execute the following commands:

```
$ dcc -o arrayAverage arrayAverage.c test_arrayAverage.c
$ ./arrayAverage
```

Again, please make sure that you test your function extensively by adding more test cases in `test_arrayAverage.c`.

We have set up a tool to guide you with your programming style. To run it:

```
$ 1511 style arrayAverage.c
Looks good!
```

You'll get advice if you need to make changes to your code. Ask your tutor for help if you're not sure.

Submit your work with the *give* command, like so:

```
$ give cs1511 wk04_arrayAverage
```

Or, if you are working from home, upload the relevant file(s) to the `wk04_showArray` activity on Give Online.

Exercise - 05: Array - Positive Minimum (individually)

You need to complete this exercise individually.

Download the following two files:

- arrayPositiveMin.c
- test_arrayPositiveMin.c

You need to implement the following function that returns minimum positive value from a given array `a` (of type `int`). A value is a positive value if it is greater than zero. If there are no positive values in a given list, the function should return zero.

```
int arrayPositiveMin(int a[], int size)
```

For example,

```
arrayPositiveMin([4, 2, 9], 3)                    returns  2
arrayPositiveMin([5, -6, 3 , 20], 4)              returns  3
arrayPositiveMin([5, -6, -12 , 5, 0, -2 , 8], 7)  returns  5
arrayPositiveMin([-2, -4, -8, -2], 4)             returns  0
```

For this question, you need to use the provided file `test_arrayPositiveMin.c` to test your function. Please open the file `test_arrayPositiveMin.c` using your favourite text editor, try to understand how your function is tested. If you have any questions, please ask your tutor.

You can use the following commands to compile and run few tests already provided in the file `test_arrayPositiveMin.c`. Please make sure that you also test your function extensively by adding more test cases in `test_arrayPositiveMin.c`. To run the simple tests available in `test_arrayPositiveMin.c`, execute the following commands:

```
$ dcc -o arrayPositiveMin arrayPositiveMin.c test_arrayPositiveMin.c
$ ./arrayPositiveMin
```

Again, please make sure that you test your function extensively by adding more test cases in `test_arrayPositiveMin.c`.

Please make sure that you test your function extensively by considering more test cases.

We have set up a tool to guide you with your programming style. To run it:

```
$ 1511 style arrayPositiveMin.c
Looks good!
```

You'll get advice if you need to make changes to your code. Ask your tutor for help if you're not sure.

Submit your work with the *give* command, like so:

```
$ give cs1511 wk04_arrayPositiveMin
```

Or, if you are working from home, upload the relevant file(s) to the `wk04_showArray` activity on Give Online.

Exercise - 06: Chessboard (pair)

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

In this activity you need to complete the function `drawChessboard` in the provided program.

Download chessboard.c, or copy it into your current directory on a CSE system by running

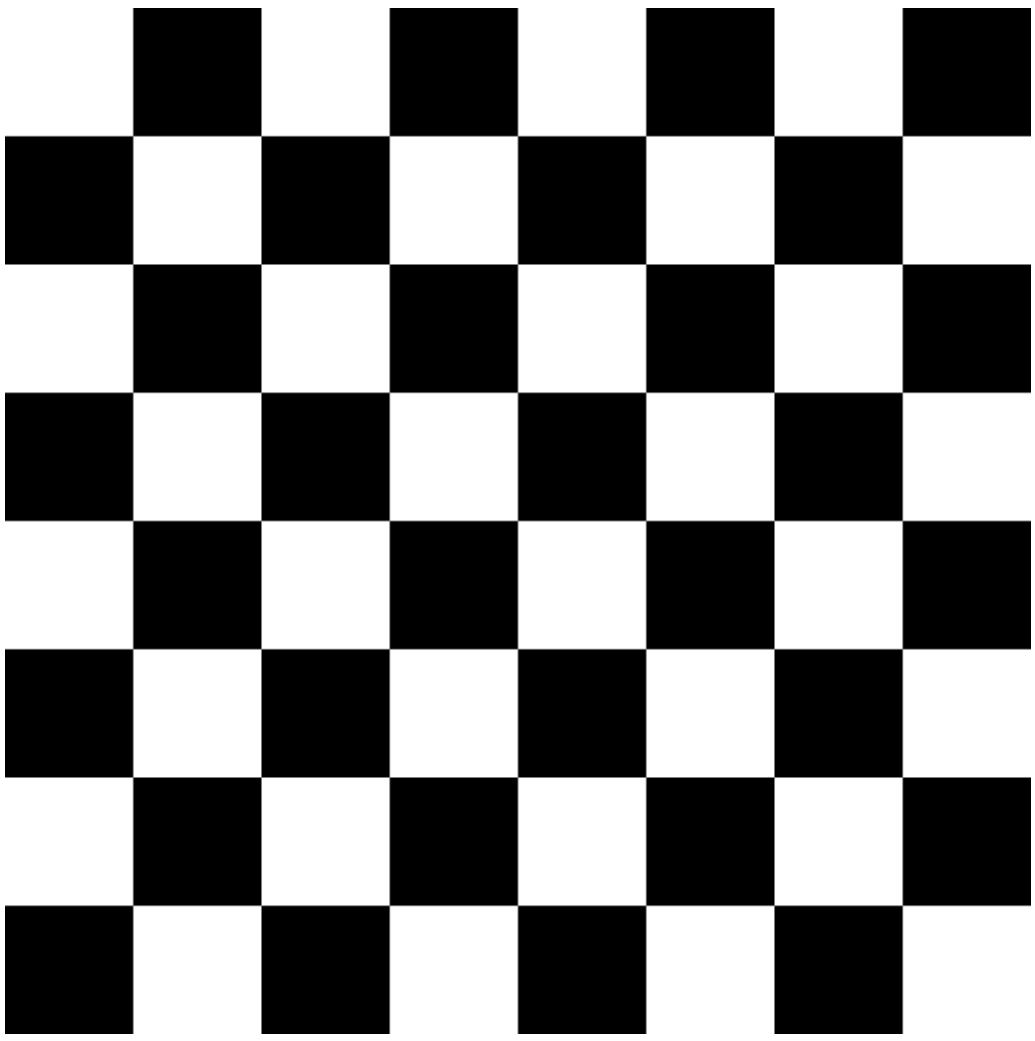
```
$ cp /web/cs1511/18s2/files/lab/04/chessboard.c .
```

You are provided with the following representation of a pixel:

```
typedef struct _pixel {
        unsigned char red;
        unsigned char green;
        unsigned char blue;
    } pixel;
```

You need to complete the `drawChessboard` function so that it fills in its given 2-dimensional pixel buffer with a chessboard image of 8 columns and 8 rows with alternating white and black squares and a white square in the top left corner.

Your image should look identical to this one



Remember, the pixel buffer is used `pixels[y][x]`, with rows first. The bottom-left corner of the image is the start of the pixels and corresponds with `pixels[0][0]`.

The program will send the data for the image to the screen output. To send it into a file instead run it using the following commands:

```
$ dcc -o chessboard chessboard.c
$ ./chessboard > chessboard.bmp
$ eog chessboard.bmp
```

To run some simple automated tests:

```
$ 1511 autotest chessboard
```

We have set up a tool to guide you with your programming style. To run it:

```
$ 1511 style chessboard.c
Looks good!
```

You'll get advice if you need to make changes to your code. Ask your tutor for help if you're not sure.

Submit your work with the *give* command, like so:

```
$ give cs1511 wk04_chessboard
```

Or, if you are working from home, upload the relevant file(s) to the `wk04_chessboard` activity on Give Online.

Exercise - 07: Print Wondrous (pair)

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

Download print_wondrous.c here, or copy it to your CSE account using the following command:

```
$ cp -n /web/cs1511/18s2/files/lab/04/print_wondrous.c    .
```

You have been given a main function and an incomplete `print_wondrous` function to complete.

This function takes in an integer and must print out the Wondrous Sequence starting with that number, followed by a single new line character.

The Wondrous Sequence is generated by the simple rule:

- if the current term is even: the next term is half the current term.
- if the current term is odd: the next term is three times the current term, plus 1.

For example the sequence generated by starting with 3 is: `3 10 5 16 8 4 2 1`

Your function is to print a single new line character '\n' after printing the final 1, and then nothing else.

Apart from printing out the sequence, your function must return the number of terms in the printed sequence (which the main function will print out).

Some Examples:

```
Enter a number: 17
17 52 26 13 40 20 10 5 16 8 4 2 1
The count is 13.
```

```
Enter a number: 1
1
The count is 1.
```

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

```
$ 1511 autotest print_wondrous
```

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

```
$ give cs1511 wk04_print_wondrous print_wondrous.c
```

Note, even though this is a pair exercise, you both must run **give** from your own account before **Sunday 19 August 23:59:59** to obtain the marks for this lab exercise.

Exercise - 08: Most Wondrous (individual) [Challenge]

This is an individual exercise to complete by yourself.

Below is the length of the **wondrous sequence** I get when starting with certain numbers. For example, the sequence starting with 97 has length 119.

Write a C program to find the **smallest starting value which has length over 1234.**

tip: be careful of your numbers getting too big and overflowing. You might want to investigate unsigned long long numbers, and/or think of your own way of representing big numbers using two or more smaller numbers somehow.

Don't use any c libraries except those we have already covered in class.

- 1 has length 1
- 2 has length 2
- 3 has length 8
- 6 has length 9
- 7 has length 17
- 9 has length 20
- 18 has length 21
- 25 has length 24
- 27 has length 112
- 54 has length 113
- 73 has length 116
- 97 has length 119
- 129 has length 122
- 171 has length 125
- 231 has length 128

- 313 has length 131
- 327 has length 144
- 649 has length 145
- 703 has length 171
- 871 has length 179
- 1161 has length 182
- 2223 has length 183
- 2463 has length 209
- 2919 has length 217
- 3711 has length 238
- 6171 has length 262
- 10971 has length 268
- 13255 has length 276
- 17647 has length 279
- 23529 has length 282
- 26623 has length 308
- 34239 has length 311
- 35655 has length 324
- 52527 has length 340
- 77031 has length 351
- 106239 has length 354
- 142587 has length 375
- 156159 has length 383
- 216367 has length 386
- 230631 has length 443
- 410011 has length 449
- 511935 has length 470
- 626331 has length 509
- 837799 has length 525
- 1117065 has length 528
- 1501353 has length 531
- 1723519 has length 557
- 2298025 has length 560
- 3064033 has length 563
- 3542887 has length 584
- 3732423 has length 597
- 5649499 has length 613
- 6649279 has length 665
- 8400511 has length 686
- 11200681 has length 689
- 14934241 has length 692
- 15733191 has length 705
- 31466382 has length 706
- 36791535 has length 745
- 63728127 has length 950
- 127456254 has length 951
- 169941673 has length 954
- 226588897 has length 957
- 268549803 has length 965
- 537099606 has length 966
- 670617279 has length 987
- 1341234558 has length 988

Save your answer in a text file called `mostwondrous.txt`, and submit that when you have found the answer.

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

```
$ give cs1511 wk04_most_wondrous mostwondrous.txt
```

You must run give before **Sunday 19 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 19 August 23:59:59** to submit your work.

Automarking will be run several days after the submission deadline for the test.

COMP1511 18s2: Programming Fundamentals is brought to you by
the School of Computer Science and Engineering at the University of New South Wales, Sydney.
For all enquiries, please email the class account at cs1511@cse.unsw.edu.au

CRICOS Provider 00098G