COMP1511 18s2

Week-05 Laboratory Exercises

Objectives

- **Week-05 Practical Lab Exam**
- simple processing of characters
- using command line arguments
- an introduction to encryption & decryption

Topics

- **Week-05 Practical Lab Exam** : in the first hour of the lab, your tutor will provide instructions on how to get started. Please bring your **student id card**.
- Getting Started
- Exercise 00: Explore the Mandelbrot Set (pair)(warmup)
- Exercise 01: Devowelling Text (pair)
- Exercise 02: Encrypting Text with a Caesar Cipher (pair)
- Exercise 03: Working Out the Letter Frequencies of Text (pair)
- Exercise 04: Cracking A Caesar Cipher (individual) [Challenge]

Preparation

The Week-05 Practical Lab Exam will be held in the first hour of the lab. Your tutor will provide instructions on how to get started. Please bring your **student id card**.

Before the lab you should re-read the relevant lecture slides and their accompanying examples. **We will only assess lab Exercises 01, 02 and 03 to derive marks for this lab.** However, you will learn a lot by attempting and solving the challenge exercises.

Getting Started

Note: if your lab is on Monday, your tutor will tell you in your tutorial how to use commandline arguments in your program.

Create a new directory for this lab called `lab05` by typing:

```
$ mkdir lab05
```

Change to this directory by typing:

```
$ cd lab05
```

Introduction

WWII code-breaking at Bletchley Park was the genesis of modern computing. In this lab you, too, will perform computer-assisted code-breaking, but, don't worry, the ciphers you must break are simpler than the Nazi's Lorenz and Enigma ciphers.

The exercises in this lab require you to read characters. Use the library function **getchar** to do this.

getchar reads the next character from standard input and returns it. If **getchar** is unable to read a character it returns the special value `EOF`.

When input is coming from a file, **getchar** will return `EOF` after the last character in the file is read.

When input is coming from a (Linux/OSX) terminal, you can indicate no more characters can be read by typing `Ctrl+D`. This will cause **getchar** to return `EOF`.

In some of this week's lab exercises you will find it convenient to put the test input in a file, rather than typing it every time you want to test the program.

You can use a < character to indicate to the shell that you want to run a program taking its input from a file.

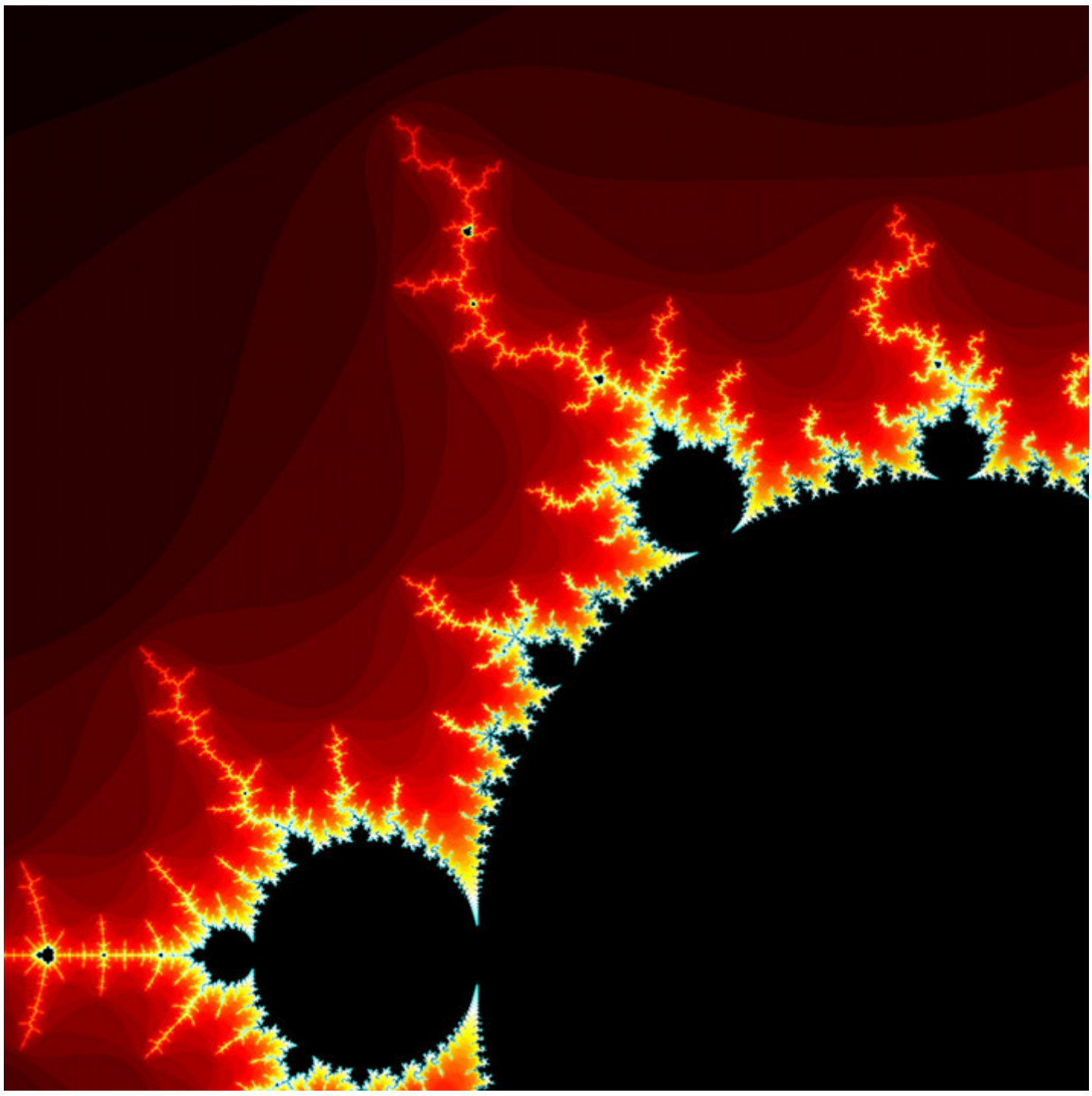
So, for example, you might create the file `input.txt` with gedit and then run `a.out` as below so that it takes its input from the file rather the terminal:

```
$ gedit input.txt &
$ ./a.out <input.txt
```

Exercise-00: Explore the Mandelbrot Set (pair)(warmup)

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

The Mandelbrot set, named after its inventor, Benoit Mandelbrot is astonishingly complex and detailed at all levels of abstraction.



Explore the Mandelbrot set by going to almondbread.cse.unsw.edu.au. Note any interesting sights you come across, and share the coordinates and your image in the comments below. You?ll be working extensively with the Mandelbrot set over the next few weeks, so make sure you get properly acquainted.

The address for the the tile centred on (-1.4, 0.0) at zoom level 10 is

http://almondbread.cse.unsw.edu.au/mandelbrot/2/10/-1.4/0.0/tile.bmp.

(which you can inspect directly by copy-pasting the URL into your browser?s address bar).

Your task: post interesting findings!
You need to post one or more almondbread tiles in PNG format in the Mandelbrot Set which you discover, which you find beautiful, unexpected, or otherwise interesting. Go to the following page and in the comment text box give the co-ordinates and zoom level for the tile(s) you find interesting. Also blog about the best things you have found and try to make your blog post attractive.

- Post interesting findings (described above) in the "comment text box" on this page

To get the tile, change the **x** and **y** co-ordinates and the **z** (for *zoom*) level in the URL and paste it into your browser?s address bar. This will show you one tile from almondbread, which you can save.

The URL will be of the format: **http://almondbread.cse.unsw.edu.au/mandelbrot/2/z/x/y/tile.bmp**; for example, http://almondbread.cse.unsw.edu.au/mandelbrot/2/10/-1.4/0.0/tile.bmp

Check out your classmates? images in the comments below and marvel at this rich land first discovered by Mandelbrot.

Exercise-01: Devowelling Text (pair)

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

Write a C program `devowel.c` which reads characters from its input and writes the same characters to its output, except it does not write lower case vowels ('a', 'e','i', 'o', 'u').
Your program should stop only at the end of input.

For example:

```
$ ./devowel
```

```
Are you saying 'Boo' or 'Boo-Urns'?
Ar y syng 'B' r 'B-Urns'?
In this house, we obey the laws of thermodynamics!
In ths hs, w by th lws f thrmdynmcs!
Ctrl-d
```

Hint: hint use `getchar` to read characters (don't use `scanf` or `fgets`).
Hint: you need only a single int variable. Don't use an array.

Hint: use `putchar` to output each character.

Hint: make sure you understand this example program which reads characters until end of input.

Hint: make sure you understand this example program which reads characters, printing them with lower case letters converted to upper case.

Hint: create a function with a prototype like this:

```
int is_vowel(int character);
```

which returns 1 the character is a lower case vowel and 0 otherwise.
Hint: To tell the program you have finished typing, you can press `Ctrl+D`.

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

```
$ 1511 autotest devowel
```

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

```
$ give cs1511 wk05_devowel devowel.c
```

Note, even though this is a pair exercise, you both must run **give** from your own account before **Sunday 26 August 23:59:59** to obtain the marks for this lab exercise.

Exercise-02: Encrypting Text with a Caesar Cipher (pair)

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

Write a C program `caesar.c` which reads characters from its input and writes the characters to its output encrypted with a Caesar cipher.
A Caesar cipher shifts each letter a certain number of positions in the alphabet.

The number of positions to shift will be given to your program as a command line argument.

Characters other than letters should not be encrypted.

Your program should stop only at the end of input.

Your program should contain at least one function other than main.

For example:

```
$ ./caesar 1
This life well it's slipping right through my hands
Uijt mjgf xfmm ju't tmjqqjoh sjhiu uispvhi nz iboet
These days turned out nothing like I had planned
Uiftf ebzt uvsofe pvu opuijoh mjlf J ibe qmboofe
Ctrl-d
$ ./caesar 10
abcdefghijklmnopqrstuvwxyz
klmnopqrstuvwxyzabcdefghij
ABCDEFGHIJKLMNOPQRSTUVWXYZ
KLMNOPQRSTUVWXYZABCDEFGHIJ
Ctrl-d
$ ./caesar -42
Control well it's slipping right through my hands
Myxdbyv govv sd'c cvszzsxq bsqrd drbyeqr wi rkxnc
These days?
Droco nkic?
Ctrl-d
```

Hint: handle upper and lower case letters separately

Hint: use `%`

Hint: use `atoi` to convert the first command-line argument to an int.

Hint: make sure you understand this example program which uses a `atoi` to convert command-line arguments to an ints.

Hint: create a function with a prototype like this:

```
int encrypt(int character, int shift);
```

which returns the character shifted by the specified amount.

Manually Cracking a Caesar Cipher

Here is some (New Zealand) English text that has been encrypted with a Caesar cipher.

```
Z uf dp drbvlg ze jfdvsfup vcjv'j tri
Nv fiuvi uzwwvivek uizebj rk kyv jrdv srij
Z befn rsflk nyrk pfl uzu reu Z nreer jtivrd kyv kilky
Jyv kyzebj pfl cfmv kyv svrty, pfl'iv jlty r urde czri
```

Use the program you have just written to discover the secret text.

Hint:: try different shifts until you see English.

You program will only be tested with an appropriate command line argument - but a good programmer would check the command line argument is present and appropriate.

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

```
$ 1511 autotest caesar
```

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

```
$ give cs1511 wk05_caesar caesar.c
```

Note, even though this is a pair exercise, you both must run **give** from your own account before **Sunday 26 August 23:59:59** to obtain the marks for this lab exercise.

Exercise-03: Working Out the Letter Frequencies of Text (pair)

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

Write a C program `frequency_analysis.c` which reads characters from its input until end of input.
It should then print the occurrence frequency for each of the 26 letters 'a'..'z'.

The frequency should be printed as a decimal value and an absolute number in exactly the format below.

Note upper and lower case letters are counted together.

For example:

```
$ ./frequency_analysis
Hello and goodbye.
Ctrl-d
'a' 0.066667 1
'b' 0.066667 1
'c' 0.000000 0
'd' 0.133333 2
'e' 0.133333 2
'f' 0.000000 0
'g' 0.066667 1
'h' 0.066667 1
'i' 0.000000 0
'j' 0.000000 0
'k' 0.000000 0
'l' 0.133333 2
'm' 0.000000 0
'n' 0.066667 1
'o' 0.200000 3
'p' 0.000000 0
'q' 0.000000 0
'r' 0.000000 0
's' 0.000000 0
't' 0.000000 0
'u' 0.000000 0
'v' 0.000000 0
'w' 0.000000 0
'x' 0.000000 0
'y' 0.066667 1
'z' 0.000000 0
$ ./frequency_analysis
Hey! Hey! Hey!
I don't like walking around this old and empty house
So hold my hand, I'll walk with you my dear
Ctrl-d
'a' 0.072289 6
'b' 0.000000 0
'c' 0.000000 0
'd' 0.084337 7
'e' 0.084337 7
'f' 0.000000 0
'g' 0.012048 1
'h' 0.096386 8
'i' 0.072289 6
'j' 0.000000 0
'k' 0.036145 3
'l' 0.084337 7
'm' 0.036145 3
'n' 0.060241 5
'o' 0.084337 7
'p' 0.012048 1
'q' 0.000000 0
'r' 0.024096 2
's' 0.036145 3
't' 0.048193 4
'u' 0.036145 3
'v' 0.000000 0
'w' 0.036145 3
```

```
'x' 0.000000 0
'y' 0.084337 7
'z' 0.000000 0
```

Hint: hint use `getchar` to read characters (don't use `scanf` or `fgets`).
Hint: make sure you understand this example program which reads characters until end of input.

Hint: use an array to store counts of each letter.

Hint: make sure you understand this example program which counts integers from the range 0..99.

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

```
$ 1511 autotest frequency_analysis
```

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

```
$ give cs1511 wk05_frequency_analysis frequency_analysis.c
```

Note, even though this is a pair exercise, you both must run **give** from your own account before **Sunday 26 August 23:59:59** to obtain the marks for this lab exercise.

Exercise-04: Cracking A Caesar Cipher (individual) [Challenge]

This is an individual exercise to complete by yourself.

Write a C program **crack_caesar.c** which decrypts text encrypted by an unknown Caesar cipher.
Your program should make no assumptions about the language of the original text - don't assume its English. However, you can assume the English alphabet ('a'..'z').

Your program will be given as a command-line argument the name of a file containing a large amount of unencrypted text in the same language as the encrypted text.

For example your program might be given this file containing 188k characters of English text (wikipedia sentences from here).

Your program will be given the encrypted text on standard input. It should print its decryption.

For example, here is some English text encrypted with a Caesar cipher with an unknown shift:

```
Kyzj zj fli crjk xffuspv
Z yrkv kf wvvc kyv cfmv svknvve lj uzv
Slk zk'j fmvi
Aljk yvri kyzj reu kyve Z'cc xf
Pfl xrmv dv dfiv kf czmv wfi
Dfiv kyre pfl'cc vmvi befn
```

So, for example:

```
$ ./crack_caesar wiki_sentences.txt
Kyzj zj fli crjk xffuspv
Z yrkv kf wvvc kyv cfmv svknvve lj uzv
Slk zk'j fmvi
Aljk yvri kyzj reu kyve Z'cc xf
Pfl xrmv dv dfiv kf czmv wfi
Dfiv kyre pfl'cc vmvi befn
Ctrl-d
This is our last goodbye
I hate to feel the love between us die
But it's over
Just hear this and then I'll go
You gave me more to live for
More than you'll ever know
```

You may assume the encrypted text of stdin contains at most 10000 characters.
You may assume the unencrypted example text in the file contains at most 250000 characters.

Hint: use `fopen` to open the file and `fgetc` to read the file. If you haven't seen them in lectures yet, read this example program to see how to use this functions to read a file.

Hint: read all the encrypted text into an array, then decrypt it.

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

```
$ 1511 autotest crack_caesar
```

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

```
$ give cs1511 wk05_crack_caesar crack_caesar.c
```

You must run give before **Sunday 26 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 with each exercise 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 26 August 23:59:59** to submit your work.