Worksheet 4

COMP110: Principles of Computing

Ed Powley

January 2016

Introduction

In this assignment, you will create two small C++ programs:

A. A console application implementing the "terminal hacking" word-guessing minigame from the Fallout games;

B. A graphical application which generates and displays the Mandelbrot fractal.

This worksheet tests your ability to translate various program notations (pseudocode, flowcharts, mathematics, narrative descriptions) into C++ code.

Submission instructions

The GitHub repository at the following URL contains skeleton projects for the two parts of this worksheet.

`https://github.com/Falmouth-Games-Academy/comp110-worksheet-4`

Fork this repository into your own GitHub account. To submit, create a GitHub pull request.

For the most part, this worksheet requires you to edit C++ code in the skeleton projects provided. **Please do not rename or delete the projects or files provided, and please do not create new projects.** This will ensure that GitHub's "diff" view highlights only the parts of the code that you have edited. Creating new source and/or header files is permitted, if doing so improves the structure of your programs.

Marking

Timely submission: 40%

The deadline for this worksheet is **6pm, Wednesday 10th February 2016**. To obtain the marks for timely submission, you must submit (as a GitHub pull request) your progress towards **both** parts of the worksheet by this time. As with other worksheets, you may resubmit after these deadlines in order to collect extra correctness or quality marks. This 40% is awarded as long as you submit *something* for each part by the deadline, even if your submission has bugs or other issues.

Correctness: 30%

To obtain the marks for correctness, you must submit working solutions for the following:

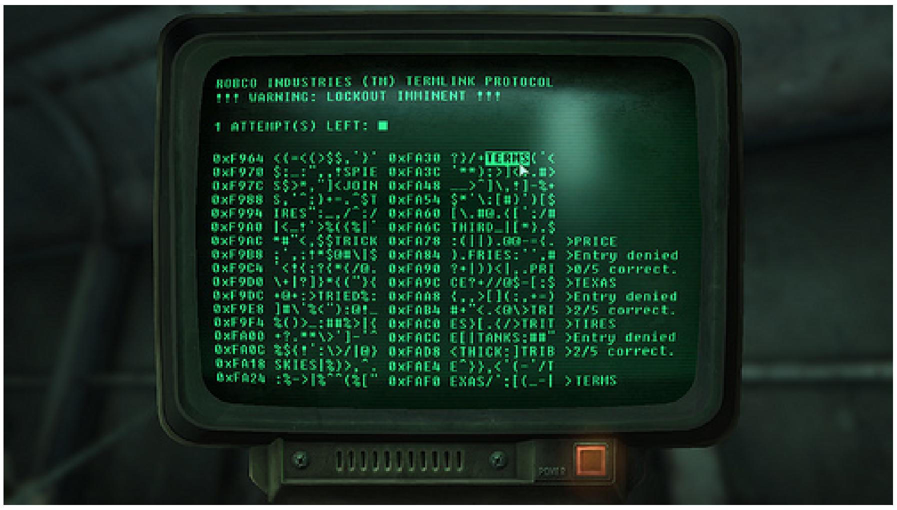
- Part A, Sections 1–2;
- Part B, Sections 1–2.

Note that this is a threshold: the full 30% is awarded for work which is *complete* and contains *no clear errors*. In particular you will not be penalised for trivial errors which do not affect the overall functioning of your programs, nor will you receive extra credit for highly polished solutions.

Quality: 30%

The extra quality criteria for this worksheet are as follows. All are weighted equally, so each is worth 6% of the overall mark for this worksheet.

1. **Presentation.** Your solutions are appropriately presented in GitHub, with descriptive commit messages and appropriate documentation in `readme.md` files. You have edited the provided skeleton projects, and refrained from renaming the provided files or creating new projects. You have checked code into GitHub regularly whilst working on the worksheet.
2. **Code quality.** Your code is well formatted. Variable and function names are clear and descriptive. Comments are used where appropriate, and are well written. Your formatting, naming and commenting follow consistent conventions.
3. **Part A stretch goal.** You have submitted a working solution for Part A, Section 3 (an improved word selection algorithm for the terminal hacking game).
4. **Part B stretch goal.** You have submitted a working solution for Part B, Section 3 (a generator for a fractal other than the Mandelbrot set).
5. **Sophistication.** Your solution for **either** of the stretch goals is particularly sophisticated, demonstrating mastery of C++ programming concepts appropriate to the task at hand. (N.B.: the emphasis here is on *appropriate*; you will not receive extra credit for shoehorning advanced constructs into your code where a simpler solution would have sufficed.)



```
ROBCO INDUSTRIES (TM) TERMLINK PROTOCOL
!!! WARNING: LOCKOUT IMMINENT !!!

1 ATTEMPT(S) LEFT: ■

0xF964 <(=<{)$$,`}`  0xFA30 ?}/+TERMS(`<
0xF970 $:_:",;,!SPIE 0xFA3C '"*):>|<*.#>
0xF97C $$>*,"]<JOIN  0xFA48 __>`]\,!]-%+
0xF988 $,'^;)+=.^$T  0xFA54 $*'\:[#)`)[$
0xF994 IRES":_,/^:/  0xFA60 [\.#@.{[`:/#
0xF9A0 |<_!`>%{{%|`  0xFA6C THIRD_|[*}.$
0xF9AC *#"<,$$TRICK  0xFA78 :(||).@@-={.  >PRICE
0xF9B8 ;`,;!*$@#\|$  0xFA84 ).FRIES:``.#  >Entry denied
0xF9C4 '<!{;?{*{/@.  0xFA90 ?+|}}<|,.PRI  >0/5 correct.
0xF9D0 \+|?]}*{("}{  0xFA9C CE?+//@$-[:$  >TEXAS
0xF9DC +@+;>TRIED%:  0xFAA8 {,,>[](:,+=)  >Entry denied
0xF9E8 ]#\`%<"):@!_  0xFAB4 #+"<.<@\>TRI  >2/5 correct.
0xF9F4 %()>_:##%>|{  0xFAC0 ES}[.{/>TRIT  >TIRES
0xFA00 +?.**\>']-`^  0xFACC E[|TANKS:##"  >Entry denied
0xFA0C %${!`:\>/|@}  0xFAD8 <THICK:]TRIB  >2/5 correct.
0xFA18 SKIES|%})>,^. 0xFAE4 E^}},<`{-`/T
0xFA24 :%->|%^^(%["  0xFAF0 EXAS/`:[(_-|  >TERMS
```

Figure 1: A screenshot of the hacking minigame in *Fallout 4*. Image from `http://fallout.wikia.com/wiki/Terminal`.

Part A. Terminal Hacking

In this part, you will implement a version of the "terminal hacking" minigame from *Fallout 4* (Bethesda, 2015); see Figure 1. In this minigame you must guess a secret n-letter word, one of several options presented to you. On choosing an option, you are told the *likeness*: the number of letters which match the secret word (i.e. the same letter in the same position). For example if the secret word is `HOUSE` and your guess is `MOUSE`, the likeness is 4 out of 5. If your guess is `HOPES`, the likeness is 2 out of 5 (the letters `S` and `E` do not count as they are in the wrong positions).

The GitHub repository contains a project named `PartA_TerminalHacking` for you to build upon. This contains code to read words from a file, and choose the secret word and other words. You will implement the rest of the game.

1.

Algorithm 1 takes a guessed word and the secret word, and returns the likeness score as described above.

Implement the algorithm as a C++ function in `TerminalHacking.cpp`, choosing appropriate data types for the parameters, return value, and any variables.

Algorithm 1 An algorithm for calculating the likeness score for the terminal hacking minigame.

```
procedure GetLikeness(guessedWord, secretWord)
    result ← 0
    for $i = 0, 1, \ldots, \text{secretWord.length} - 1$ do
        if secretWord[$i$] = guessedWord[$i$] then
            increment result
        end if
    end for
    return result
end procedure
```

2.

Implement the main loop of the game, structured according to the flowchart shown in Figure 2.

3. Stretch goal

In the skeleton project, the words are chosen at random. This may lead to instances of the game which are unsatisfying, for example where all words have a low likeness score with respect to the secret word.

Design an improved word choosing algorithm. Present your algorithm as pseudocode and/or a flowchart in your `readme.md` file on GitHub.

Implement your algorithm within your C++ project.

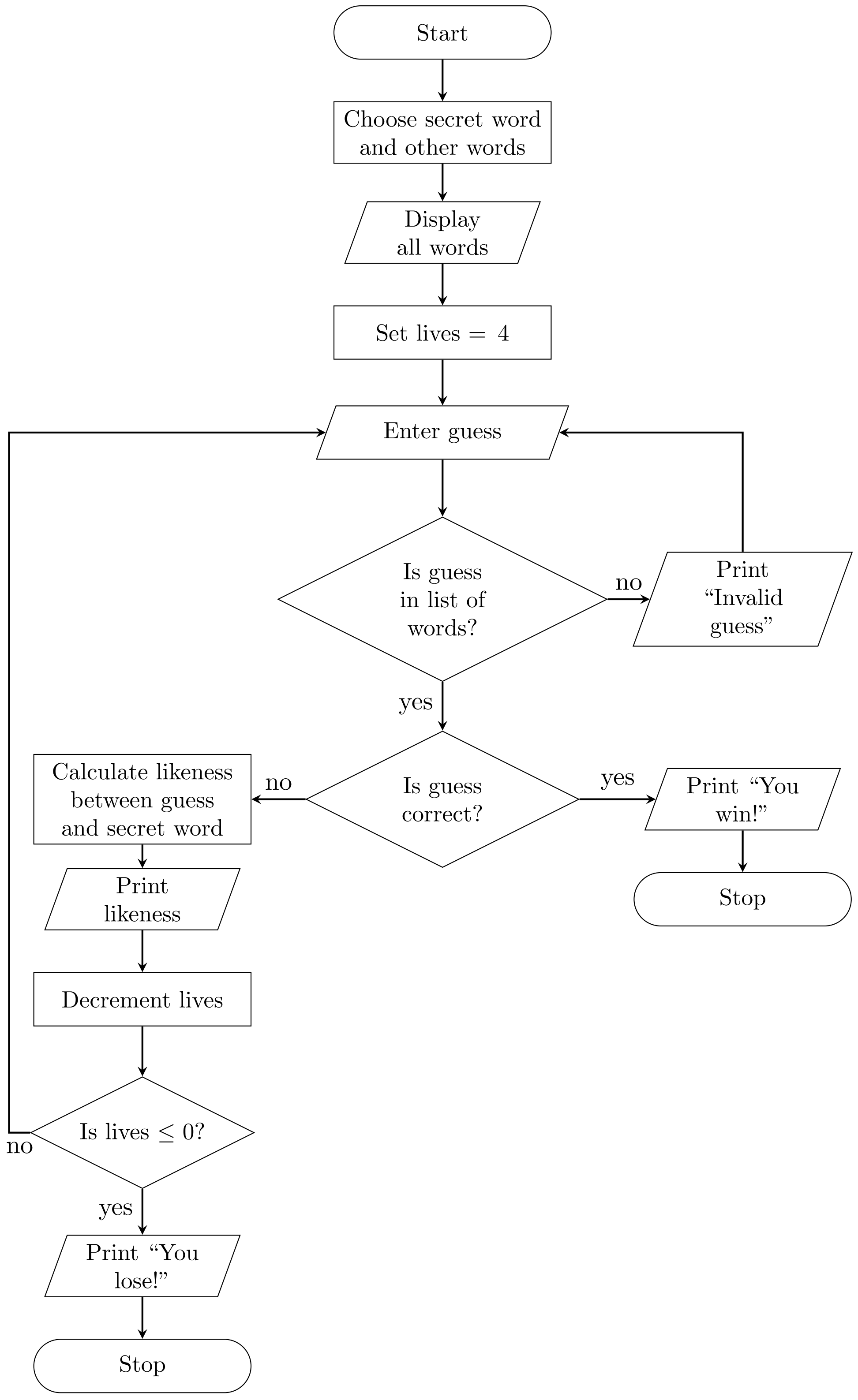


Figure 2: Flowchart for the Terminal Hacking game

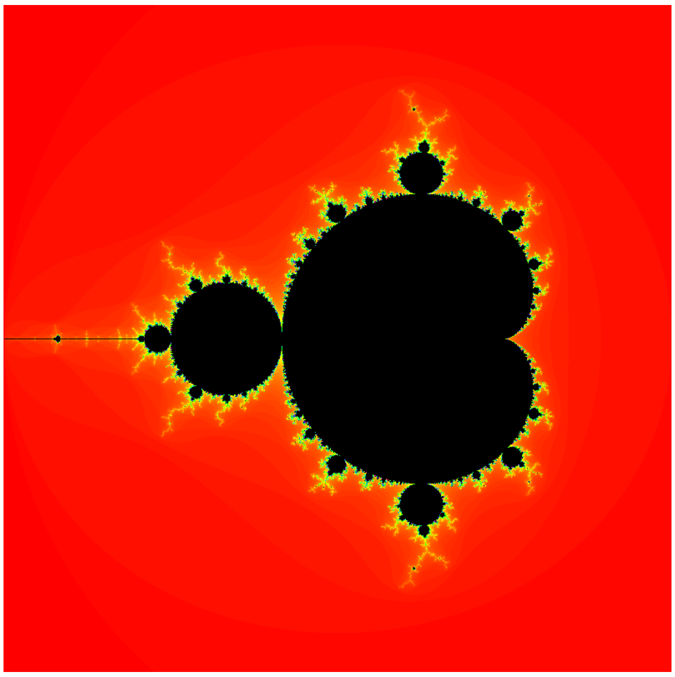


Figure 3: The Mandelbrot set fractal.

Part B. Mandelbrot

In this part, you will use the CImg library (`http://cimg.eu/`) to write a program to generate and display the *Mandelbrot set* fractal; see Figure 3. This fractal colours each pixel of the image according to an iterated mathematical formula, as described below. The GitHub repository contains a project named `PartB_Mandelbrot` for you to build upon. This contains code to create and display a blank image; you will implement the calculations to generate the Mandelbrot set fractal.

1.

To generate an interesting fractal, the on-screen x and y coordinates must first be rescaled. In the skeleton project the pixel coordinates range from 0 to 800, whereas the Mandelbrot set fractal is most interesting in the region $-2 \leq x \leq 1$ and $-1.5 \leq y \leq 1.5$.

Let p_x be the x coordinate of the pixel. This can be remapped into the range x_{min} to x_{max} using the following formula:

$$x_0 = \frac{p_x}{\text{image.width}} \times (x_{\text{max}} - x_{\text{min}}) + x_{\text{min}}$$

The y coordinate can be remapped using a similar formula.

Implement the above calculations for the x and y coordinates, at the indicated parts of `Mandelbrot.cpp`.

2.

The Mandelbrot set is based on the following sequence of numbers. Let x_0 and y_0 be the coordinates of a point in the image. Then the sequence $x_1, y_1, x_2, y_2, x_3, y_3, \ldots$ is defined[1] recursively for $i = 0, 1, 2, 3, \ldots$ by:

$$\begin{aligned} x_{i+1} &= (x_i)^2 - (y_i)^2 + x_0 \\ y_{i+1} &= (2 \times x_i \times y_i) + y_0 \end{aligned}$$

The points are coloured according to the *smallest* value of i for which $(x_i)^2 + (y_i)^2 \geq 4$. If such a value of i is not found after a large number of iterations (for example $i = 200$), the pixel is coloured black.

Implement an algorithm which performs the above computation, determining the smallest value of i for which $(x_i)^2 + (y_i)^2 \geq 4$ and selecting the appropriate pixel colour. Implement the algorithm in `Mandelbrot.cpp` so that the program generates the Mandelbrot set fractal (Figure 3) when it is run.

3. Stretch goal

Mathematicians have discovered many other fractals besides the Mandelbrot set.

Research a different type of fractal, using online resources and/or academic literature. Briefly summarise your findings (with URLs and/or citations) in your `readme.md` file.

Implement a generator for the fractal you have researched. Allow the user to choose (by command line entry) which fractal they want to generate when the program starts up: the Mandelbrot set or the new fractal. You may wish to restructure the skeleton program to accommodate the new fractal generation algorithm.

[1] If you are familiar with complex numbers, you may notice that this is equivalent to $z_{j+1} = z_j^2 + z_0$, where $z_j = x_j + y_j i$.