Curtin University — Computing Discipline

UNIX and C Programming (COMP1000)

Semester 2, 2018

Assignment

Due: Friday 19 October at 11.59 pm (UTC +8)

Weight: 20% of the unit mark.

1 Introduction

Your task for this assignment is to design, code (in C89), test and debug a program to draw terminal-based "turtle" graphics.

In short, your program will:

- Read a series of graphics commands from a file;
- Convert angles and distances to (x, y) coordinates as needed;
- Record these coordinates in a log (output) file;
- Use pre-defined functions to execute the commands; i.e. drawing lines on the screen.

There is a lot of detail here. Read it *carefully* before you start anything.

2 Documentation

You must thoroughly document your code using C comments (/* ... */). For each function you define and each datatype you declare (e.g. using struct or typedef), place a comment immediately above it explaining its purpose, how it works, and how it relates to other functions and types. Collectively, these comments should explain your design. (They are worth substantial marks – see Section 7.)

3 Task Details

Read this section carefully, there is a lot of detail, you may need to read it several times to ensure you do not miss anything.

Input file

Your program should accept a command-line parameter: the name of the file containing graphics commands. Your program should read the contents of this file into a linked list.

The file will consist of a sequence of commands, one on each line. There is no length indicator. Each command may be one of the following:

- `ROTATE` θ
- `MOVE` d
- `DRAW` d
- `FG` i
- `BG` i
- `PATTERN` c

θ is a real-valued angle, d is a real-valued distance, i is an integer-valued colour code and c is a non-space character. Command names may be in lowercase or uppercase (or any combination).

An example file might look like this:

```
rotate -45
move 30
FG 1
Pattern #
DRAW 10
Rotate 90
draw 10
ROTATE 90
dRAW 10
ROTATE 90
DRAW 10
```

(I have deliberately scrambled the casing here; this is still a valid input file.)

Error Handling

You must deal with any invalid file, this includes, but is not limited too:

- The file not existing;
- The file being empty;
- Lines that do not start with one of the specified commands;
- The incorrect number of parameters after a command;
- The incorrect data type after a command.

Turtle graphics

Your program must keep track of several changing pieces of information:

Current position: for each command, you start at a particular position on the screen, described by real-valued (x, y) coordinates, where x begins at zero on the left of the screen, and y begins at zero at the top, increasing downwards. The `MOVE` and

`DRAW` commands will change the current position. The initial position is $(0, 0)$ (the top-left screen corner).

Current angle: for each command, you start at a particular real-valued angle, expressed in degrees, from 0° to 360°. 0° means "right"; 90° means "up". The `ROTATE` command will add or subtract from the current angle; i.e. rotating it anticlockwise or clockwise. The initial angle is 0°.

Current foreground and background colours: for each command, you start with two colour codes representing the foreground and background colours. The terminal has 16 foreground colours (0–15) and 8 background colours (0–7). These are used every time any character is displayed. The `FG` command changes the foreground colour, and `BG` the background. The initial foreground colour is 7 (white) while the initial background colour is 0 (black).

Current pattern: for each command, you start with a character that will be used to make up any lines you plot. The `PATTERN` command changes the current pattern(character). The initial pattern(character)4 is the '`+`' character.

Your program must interpret and execute the sequence of commands, keeping track of the above information. Most importantly, the `DRAW` d command will draw a line d units long, starting at the current position and heading at the current angle. After the line is drawn, the current position will move to the other end of the line. The `MOVE` command works in the same fashion, except it does not actually draw a line.

To do this, your program must calculate real-valued (x, y) coordinates from angles and distances. To this end, you will need to apply basic trigonometry (using `math.h`):

$$\Delta x = d \cos \theta$$
$$\Delta y = d \sin \theta$$

When outputting lines, your program must round its real-valued coordinates to the nearest integer values, as required below.

Existing Functions

You must use the existing functions in the files `effects.c` and `effects.h`. The most important of these is `line()`, declared as follows:

```
void line(int x1, int y1, int x2, int y2,
PlotFunc plotter, void *plotData);
```

This will draw a line from integer-valued coordinates (`x1`, `y1`) to (`x2`, `y2`). It has no idea of the "current position", "current angle", etc.

The `PlotFunc` type is defined as follows:

```
typedef void (*PlotFunc)(void *plotData);
```

For each discrete point on the line, the `line()` will position the cursor at that point, then call the `*plotter` function. You must define this function yourself, and supply it to `line()`. The plotter function must generate the appropriate terminal output (i.e. the current pattern(character)).

The final parameter to `line()` is a `void*`, which is simply passed on to the plotter function. This allows you to pass any required data between your algorithm and the plotter function.

Several other functions are also available:

- `void setFgColour(int)` — changes the foreground colour to a given colour code.
- `void setBgColour(int)` — changes the background colour to a given colour code.
- `void clearScreen()` — blanks the terminal.
- `void penDown()` — moves the cursor to the bottom of the screen (for when drawing is complete).

Log file **(Consider this when developing test cases!)**

Your program must also *append* to (not overwrite) a log file called `graphics.log`.

The first line written to the log file, after its existing contents, should be "`---`". This will serve to separate the new contents from the old.

The log file must show the actual, real-valued (x, y) coordinates calculated for the `DRAW` and `MOVE` commands, in the following format: "*`command`* `(`x_1`,`y_1`)-(`x_2`,`y_2`)`". All coordinate values should be printed such that they line up vertically.

An example `graphics.log` file (generated after only one execution) would be:

```
---
            MOVE (  0.000,  0.000)-( 50.000,  33.333)
            DRAW ( 50.000, 33.333)-(100.000,  50.000)
            DRAW (100.000, 50.000)-( 50.000,  66.667)
            DRAW ( 50.000, 66.667)-( 50.000,  33.333)
```

(This is not related to the previous example.)

If the program were run a second time with the same input, `graphics.log` would become:

```
            ---
            MOVE (  0.000,  0.000)-( 50.000,  33.333)
            DRAW ( 50.000, 33.333)-(100.000,  50.000)
            DRAW (100.000, 50.000)-( 50.000,  66.667)
            DRAW ( 50.000, 66.667)-( 50.000,  33.333)
            ---
            MOVE (  0.000,  0.000)-( 50.000,  33.333)
            DRAW ( 50.000, 33.333)-(100.000,  50.000)
```

```
DRAW (100.000, 50.000)-( 50.000,  66.667)
DRAW ( 50.000, 66.667)-( 50.000,  33.333)
```

The second half of the log file would be added on the second execution.

Makefile Targets

You must provide a makefile for your assignment. Failure to provide a makefile will result in zero (0) marks being awarded for compilation/execution/test cases. Your makefile must provide the following targets.

TurtleGraphics – The compiled assignment as described above.

TurtleGraphicsSimple – This version of the assignment should generate all line art with a white background and black foreground. That is, any `FG` or `BG` commands should be ignored.

TurtleGraphicsDebug – This version of the assignment should print the log file entries to the terminal as well as the file. The log file entries should be interspersed with the turtle graphics, that is, each call to `line(...)` should be preceded by any relevant log file entries.

4 Report

You must prepare a report that outlines your design and testing. Specifically:

1. For each function you write, describe its purpose (in a paragraph).
2. On how you converted the input file to a coordinate system:
   - Describe (in two or three paragraphs) how you implemented this.
   - Describe (in one or two paragraphs) an alternative approach that would achieve the same outcome.
3. Demonstrate that your program works by showing sample input and output, including:
   - The command-line used to execute your program.
   - The user input provided via the terminal.
   - The contents of the input file.
   - The output, as shown on the screen (and written to the output file).

Your report should be professionally presented, with appropriate headings, page numbers and a contents page.

5 Submission

Submit a single .tar.gz file containing all your files (projects, report, etc.).

Submit your entire assignment electronically, via Blackboard (lms.curtin.edu.au), before the deadline.

Submit one .tar.gz file containing:

A declaration of originality – whether scanned or photographed or filled-in electronically, but in any case *complete*.
Your implementation – including all your source code (.c files, .h files, makefile, etc…).
Your report – a single PDF file containing everything mentioned in Section 4.

Note: do not use .zip, .zipx, .rar, .7z, etc, they will be taken as **non-submissions** and instantly receive zero (0) for the whole assignment.

You are responsible for ensuring that your submission is correct and not corrupted. You may make multiple submissions, but only your latest submission will be marked.

No extensions will be granted. If exceptional circumstances prevent you from submitting an assignment on time, contact the Unit Coordinator ASAP with relevant documentation. After the due date and time is too late.

Late submission policy, as per the Unit Outline, will be applied.

6 Demonstration

You will be required to demonstrate your assignment in your registered practical session, in the week starting the 22nd of October. If you cannot attend your registered practical session you must make arrangements with the lecturer prior to the 1$_{st}$ of October. You may be asked several questions about your assignment in this demonstration.

Failure to demonstrate your assignment during your registered practical will result in a mark of ZERO (0) for the entire assignment.

7 Mark Allocation

Every valid assignment will be initially awarded 100 marks. Marks will then be deducted for the following:

up to -50% Commenting.
You will not lose any marks if you have provided good, meaningful explanations of all the files, functions and data structures needed for your implementation.

up to -50% Coding practices and coding standard.
You will not lose any marks if you have followed the coding standard and good cod-

ing practices (see Blackboard, resources section), and your code is well-structured, including being separated into various, appropriate .c and .h files.

up to -100% Functionality.
You will not lose any marks if you have correctly implemented the required functionality, according to a visual inspection of your code by the marker.

up to -100% Working Product. You will not lose any marks if your program compiles, runs and performs the required tasks without unexpected error. The marker will use test data, representative of all likely scenarios, to verify this. You will not have access to the marker's test data. You may lose this entire component of your mark by either:

1. not having a working makefile,
2. failing to solve memory issues.
3. if your program must exit due to any reason then you must clean up the memory before allowing the program to exit.

up to -50% Report. You will not lose any marks if your report is complete and professionally presented.

Signoffs:

Once your mark has been calculated, the practical signoff result will be applied as follows:

Your total practical signoffs will be awarded a mark out of 10, which will have the following effect (multiplier) on the assignment mark.

Total	**Effect**
8 - 10	100%
7 - 7.9	90%
6 - 6.9	80%
5 - 5.9	70%
3 - 4.9	50%
0 - 2.9	30%

8 Academic Misconduct – Plagiarism and Collusion

This is an assessable task, and as such there are strict rules. You must not ask for or accept help from *anyone* else on completing the tasks. You must not show your work to another student enrolled in this unit who might gain unfair advantage from it.

These things are considered **plagiarism** or **collusion**.

Staff can provide assistance in helping you understand the unit material in general, but nobody is allowed to help you solve the specific problems posed in this document. The purpose of the assignment is for *you* to solve them *on your own*.

Please see Curtin's Academic Integrity website for information on academic misconduct (which includes plagiarism and collusion).

The unit coordinator may require you to provide an oral justification of, or to answer questions about, any piece of written work submitted in this unit. Your response(s) may be referred to as evidence in an Academic Misconduct inquiry. In addition, your assignment submission may be analysed by Turnitin and/or other systems to detect plagiarism and/or collusion.

End of Assignment