TURTLE GRAPHICS

Owen Frere 19520500

Table of Contents

Functionality

Turtle Graphics provides a method of drawing in the terminal. The instructions for the drawing are provided by premade lists of commands, consisting of any combination of the 6 command types. The drawing is achieved by a coordinate system tracking where a virtual pen tip is and printing characters as the pen draws.

There are two additional modes of Turtle Graphics Supplied;

- Simple mode: Ignores all colour commands and prints black foreground and white background
- Debug mode: prints a log of movement and draw coordinates to stderr as they are calculated. It its highly recommended to redirect this output to a file or another terminal.

Initial State

The pen is initialised at the top left of the terminal window, represented by the (x,y) coordinates of (0,0) with a positive x representing movement to the right and a positive y representing movement towards the bottom of the terminal. The initial angle is 0 degrees, indicating a facing directly right. The initial foreground colour is 7 (white) and the initial background colour is 0 (black). Finally, the initial pattern is '+'.

Available Commands

Turtle Graphics accepts the following commands;

1. FG [x]: Set the foreground colour of the pen to x (0-15)
2. BG [x]: Set the background colour of the pen to x (0-7)
3. MOVE [y]: Move the pen y units in current heading
4. DRAW [y]: Draws a line of y units the current pattern in current heading
5. ROTATE [y]: Rotates current heading anticlockwise y degrees
6. PATTERN [z]: Sets the pen's pattern to z.

Where x is an integer, y is a real number and z is any non-whitespace printable character.

Input File

There is no limit to the number of commands an input file can contain, within the limits of your operating system to assign sufficient memory. The commands may be in uppercase, lowercase or any combination thereof. A single error in the file will cause the program to abort. Errors generally fit into 2 categories; missing information or incorrect information.

Missing information includes;

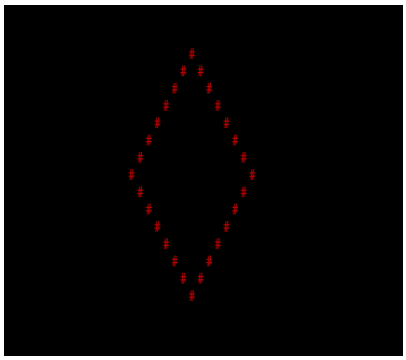
- An empty line
- A missing command name
- A missing parameter for the command

Incorrect information includes;

- A command name not one of the predefined 6
- An incorrect data type in a parameter (Note: x.0 is considered invalid for integer input)
- Any extra characters (including whitespace) after the parameter

Sample Input and Output: (From Assignment sheet)

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



Running Turtle Graphics

Turtle graphics has no UI and is controlled entirely through the command line and the input files. To run Turtle Graphics from the directory it is in use the command;

```
./TurtleGraphics x
```

To run simple mode;

```
./TurtleGraphicsSimple x
```

To run debug mode;

```
./TurtleGraphicsDebug x [2>y]
```

Where x is the file name containing the drawing commands. [2>y] is an optional and recommended argument to send the stderr output to a new location, where y is a location.

Logging

Turtle Graphics keeps a log of all of the coordinates used in draw and move commands. Whenever the program executes, graphics.log is appended with a separator "---" indicating a new execution and then the coordinates in the form

Command type (x_1,y_1)-(x_2,y_2)

Where $_1$ signifies starting coordinate and $_2$ signifies the end coordinate.

Logging Sample

A log provided by the commands in the sample input.

```
---
MOVE (  0.000,  0.000)-( 21.213, 21.213)
DRAW ( 21.213, 21.213)-( 28.284, 28.284)
DRAW ( 28.284, 28.284)-( 35.355, 21.213)
DRAW ( 35.355, 21.213)-( 28.284, 14.142)
DRAW ( 28.284, 14.142)-( 21.213, 21.213)
```

Converting Input to Coordinates

To begin with I created a state struct to hold the current state of the pen, primarily where it was and what direction it was heading when talking about coordinates. Then I created a struct for containing a command from the file. I chose to represent the command types as the integer representing the first letter of the command name, as these are unique, easy to work with and work in switch statements. I chose to use a double to represent the value as it would contain the real values I needed in rotate, move and draw, and would be able to be typecast to int for FG, BG and pattern. I could have used a void pointer to a malloced int or double, but decided against the additional complexity for the small space saving. I went for a fgets method of file reading, grabbing strings of length 128 and processing them as the string, rather than trying to fscanf. I chose a buffer length of 128 as only the reals could go longer than that, and would be pointless instructions if they did. I did error checking line by line as I process instead of doing a file parse first, as it felt more efficient in the average case. One validated the information is extracted from the string, put in a DrawComm struct and appended to the list of commands.

The program then progresses to processing commands. The linked list has DrawComms taken off in a queue order, and the state of the pen is updated by each command, and the line function is called where necessary. The x and y coordinates of the pen are calculated from the angle of movement and distance using basic trigonometric functions (sine and cosine) but of note is the inversion of the y axis. This results in having to subtract the cosine result from the y coordinate instead of adding it. While there are error checks for unknown commands here it is unexpected for any to make it that far. The major error checking here is for coordinates to go into the negative. There is a small tolerance around the negatives, I use a larger tolerance than would be required for Pi inaccuracy or floating point error. If the values are greater than -1 the printing is fine. I can only posit this is an artefact of the way the line function works.

Alternate Method

An alternate method of achieving this while meeting all the constraints of the assignment would be to use a line struct. Processing all of the commands at file read, updating the state of the pen with the FG, BG, Pattern, Angle and move commands, but upon finding a draw command creating a struct containing the beginning x and y, the end x and y, and the FG, BG, and pattern. A linked list of these lines could then by fed into the line command in place of draw commands being fed into my processing functions. This way far fewer items would need to be malloced and the linked list would be shorter.

Testing

The first issue I had to test and debug was the double printing or wonky diamond choice. As it was noted that the removal of the double printing fix from blackboard resulted in the diamond (elDiamondo.txt) going back to normal I isolated that decrement and move as the cause. I ran a test on a simpler shape, a square (beSquare.txt), and noticed it printed fine with the decrement fix still in place. This made me think the issue was angles, and I noticed that the lines going bottom left to top right were both broken. I thought maybe the angles 0-90 and 180-270 may need to be excluded from the fix. This did fix my diamond. I was concerned that was not the actual conditions and drew the diamond backwards (elDiamondoBackwards.txt) and saw a failed triangle again. I Decided to look through charizard's source file (charizard.txt) and I noticed it is only every printed along a 0 degree line. This made me think that the decrement fix should not be applied to angles, so I added a mod 90 to the angle as a check before the decrement. This has fixed the issue in all my tests, but I lack the understanding of the line function to work out if this will fix it in all cases or why it is necessitated.

Then I moved to more generalised error testing. Printing off screen (badInstruction.txt), opening non-existent files, opening a 0b file (voidFile.txt), values too big (tooBig.txt), incorrect command names and missing parameters (errorizard.txt), and providing no file name as input. I fixed several issues including an infinite loop when a massive angle was supplied; the original normalise function was a while based subtraction loop that would run into overflow issues and never reduce the angle. It was replaced with a division and truncation subtraction method while solved the issue.

Function Purpose

turtleGraphics.c

main

The main function of the turtle graphics. This function's purpose is to be the initial point of the program, to store the store the linked list header and pen state struct. It also calls the file input to fill the linked list and then step through the linked list passing commands to processCommand. Finally it is used to free memory used by draw command structs (DrawComm) as they are processed or free the draw command structs and linked list node structs in the case of an early exit.

processCommand

This function provides the flow control and command processing required by the program. It takes a DrawComm from main and performs the correct action based on the command type. Its responsibilities include calling trigonometry functions, calling functions for effects.c and updating the pen's state struct (state). Aborts and returns an error code if a DrawComm of unknown type is encountered. Also aborts if the integer truncation of the newly calculated x or y coordinates is negative. While any negative at all should theoretically be invalid this method provides a bit of leeway around the inaccuracy of Pi calculations and floating point inaccuracy. I also note that while these could be covered by a much smaller tolerance, coordinates greater than negative one do not result in incorrect terminal printing, which I can only posit is an artefact of the functionality of the provided line command's calculations.

Imports: state (State*), comm (DrawComm*)

Exports: errCode (int)

printLogging

The printLogging function's purpose is to print a log file of the move and draw commands including their start and end coordinates. It is also responsible for printing to stderr in the debug compilation of the program.

Imports: state (State*), type (Char*), newX (int), newY (int), log (FILE*)

Exports: none

intialiseStorage

While not entirely necessary this function was used to keep the main function clear of setting default values while also providing an easy mechanism of changing the defaults. Combined with a pre-processor defined default values this function sets the state to all its default values.

Imports: state (State*)

Exports: none

calcCoords

calcCoords provides the trigonometric functions required for calculating changes in 2 dimensional coordinates from just an angle and a distance. Of note is the inversion of the y axis used in the Turtle Graphics system necessitating a change in the normal method of calculating coordinates, with the sin value being subtracted rather than added.

Imports: state(State*), distance (double), newX (double*), newY (double*)

Exports: none

Plotter

This function's purpose is purely to meet the requirements of the line function provided in effects.c. It takes prints a single character (supplied by a void pointer) when called.

Imports: plotData (void*)

Exports: none

printErr

Error codes are not generally person readable. This function is to take any of the error codes from the program and print out a meaningful message related to the code prior to exit. Every error code in Turtle Graphics corresponds to the same number in this function, with the exception of the error codes from the extraction functions, which just indicate an error occurred. The extraction errors are then replaced with the line number of the file the error was on, which is used as the real error code.

Imports: errCode (int)

Exports: none

fileIO.c

loadCommandList

loadCommandList purpose is to open the input file, extract it line by line and pass the lines to the main flow control function of fileIO for processing, and then close the file. It is the initial point of the fileIO process. Aborts and returns an error if the file cannot be opened, is empty, or any call to processLine returns an error.

Imports: fileName (char*), list (LinkedList*)

Exports: errCode (int)

processLine

This function provides the main flow control of the fileIO process. The function cleans the commands names by using strUpr to turn it to all uppercase to reduce complexity of handling the commands for the rest of the program, which only needs to function on uppercase. It then uses the first letter of a command to define its type, and from this calls the relevant extraction function to determine the value. The command type and the value are used to fill a newly malloced DrawComm, which is added to the linked list of commands. If the line does not start with an expected character, or an extraction function returns an error the function returns an error code of the offending line number.

Imports: line (char*), list (LinkedList*), ii (int*)

Exports: errCode (int)

processDouble

The processDouble function is used to verify the command name by matching to an expected name, extract a real value, and ensure no extra information is present. If any of these fail the function returns an error.

Imports: line (char*), expected (char*), value (double*)

Exports: errCode (int)

processInt

The processInt function is used to verify the command name by matching to an expected name, extract an integer value, the values are within expected ranges for FG and BG, and ensure no extra information is present. Of note is the strict classification of integer here. If a real number is detected, but equal to an integer (i.e. 12.00 or 0.00), an error is thrown as this is the incorrect data type. I chose a strict interpretation as this suggests there may have been an error in file creation as the command may have been intended to be a different type. Used will be notified of line, making it easier to troubleshoot. If any of these fail the function returns an error.

Imports: line (char*), expected (char*), value (int*)

Exports: errCode (int)

processChar

The processChar function is used to verify the command name by matching to an expected name, extract an integer value representing a printable character, and ensure no extra information is present. Of note is that the more than one character will return an error, including whitespace. If any of these fail the function returns an error.

Imports: line (char*), expected (char*), value (int*)

Exports: errCode (int)

linkedList.c

insertLast

This function adds a node containing a DrawComm to the end of a linked list storing the queue of DrawComms to process. It mallocs the space required by the node, updates the LinkedList's head (if the list is empty) and tail, and the next of the previous node (if extant).

Imports: list (LinkedList*), comm (DrawComm*)

Exports: none

removeFirst

This function removes the first node from the linked list, returns the DrawComm stored within, and frees the memory used by the now redundant node. If the list only has a single node, it sets the LinkedList's head and tail to null as well, otherwise updates the head to the next node.

Imports: list (LinkedList*), comm (DrawComm*)

Exports: errCode (int)

tools.c

strUpr

strUpr is used to convert all lower case characters in a string to uppercase characters. It was included as makes the all future comparisons within the program simplified, this is a safe optimisation as case is irrelevant in command names. As the compilation is C89, the strupr built in function is not available and so this function is made in replacement. The functionality is based on a lecture by Mark Upston.

Imports: line (char*)

Exports: none

round

The line function provided by effects.c requires integer coordinates to be supplied and the coordinates are actually stored as real numbers, necessitating a rounding function as truncation would be inaccurate. As C89 does not have a rounding function one had to be made. Tiebreaker for the x.5 case is to go to the adjacent even number.

Imports: numIn (double)

Exports: outNum (int)

normaliseAngle

This function normalises a supplied angle to within 0.0 and 360.0. Not completely necessary but normalised angles are more easily human readable, making trouble shooting simpler. Has no effect on the drawing. Has a slight chance of protecting against an integer overflow in the case of millions of rotations in the same direction, although this would be unexpected.

Imports: state(state*)

Exports: none