UCP Assignment Report

TURTLE GRAPHICS IMPLEMENTATION IN C

KYLE ZRINSKI

CONTENTS

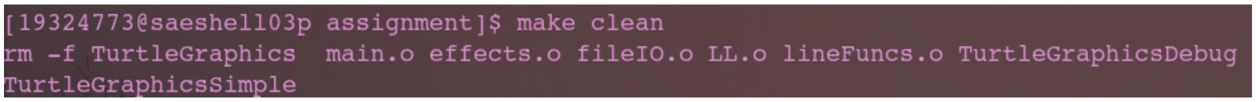
Making the Program 3
  Clean 3
  TurtleGraphics 3
  TurtleGraphicsDebug 3
  TurtleGraphicsSimple 3
Running the Program 4
  Command line arguments and input file selection 4
  Input file 4
  Output 4
    TurtleGraphics(Debug): 4
    graphics.log: 4
Coordinate System 5
  Current Implementation 5
  Alternative Implementation 5
Functions 5
  FileIO.c 5
    readFile 5
    writeFile 6
    readLine 6
    split 6
    strCase 6
    cmdCheck 6
  LineFuncs.c 6
    exCommand 6
    plot 6
    rotate 6
    move 7
    draw 7
    deg2rad 7

round 7
LL.c 7
createList 7
createNode 7
InsertFirst 8
InsertLast 8
removeFirst 8
removeLast 8
freeList 8
Main.c 8
startIO 8
main 8

MAKING THE PROGRAM

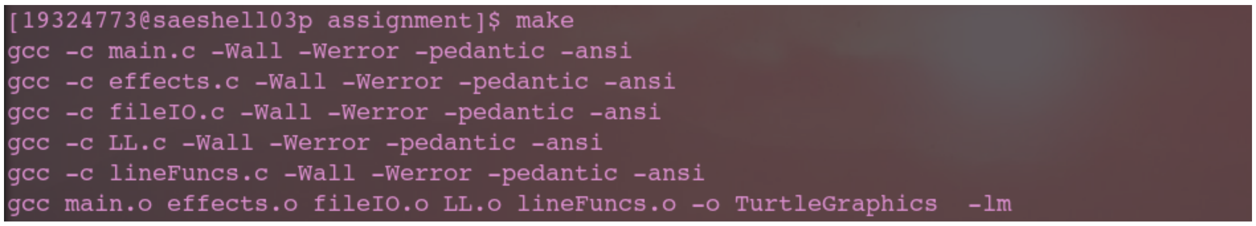
As per the assignment specification there are 3 different versions of the program; TurtleGraphics, TurtleGraphicsDebug and TurtleGraphicsSimple. These are all produce by the makefile with their respective commands as shown below, as well as a clean rule to remove all .o and program files.

CLEAN



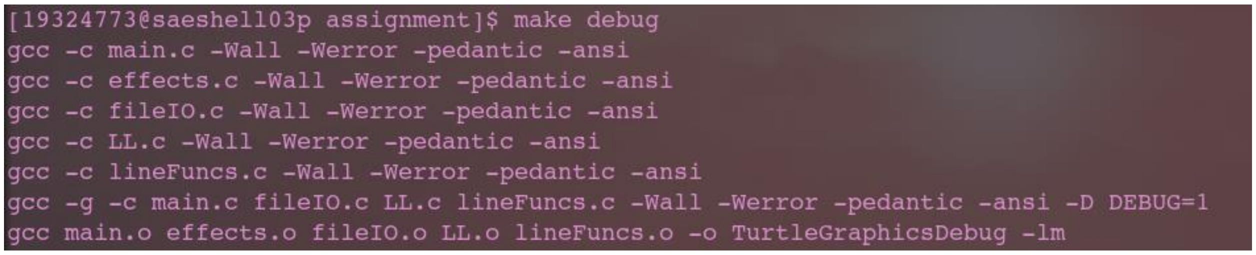
```
[19324773@saeshell03p assignment]$ make clean
rm -f TurtleGraphics  main.o effects.o fileIO.o LL.o lineFuncs.o TurtleGraphicsDebug
TurtleGraphicsSimple
```

TURTLEGRAPHICS



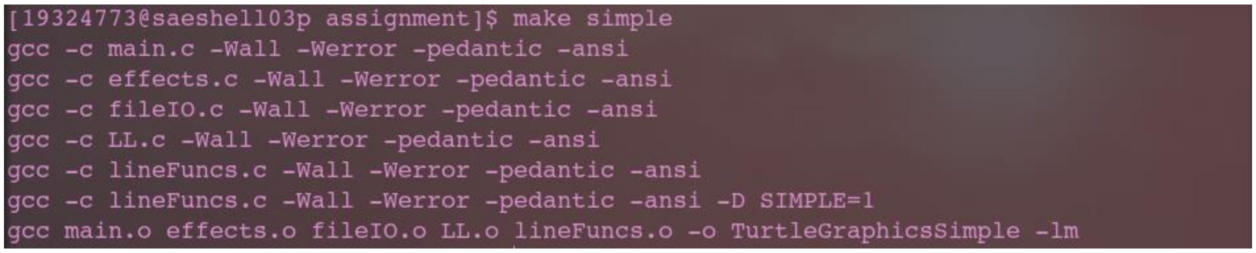
```
[19324773@saeshell03p assignment]$ make
gcc -c main.c -Wall -Werror -pedantic -ansi
gcc -c effects.c -Wall -Werror -pedantic -ansi
gcc -c fileIO.c -Wall -Werror -pedantic -ansi
gcc -c LL.c -Wall -Werror -pedantic -ansi
gcc -c lineFuncs.c -Wall -Werror -pedantic -ansi
gcc main.o effects.o fileIO.o LL.o lineFuncs.o -o TurtleGraphics  -lm
```

TURTLEGRAPHICSDEBUG



```
[19324773@saeshell03p assignment]$ make debug
gcc -c main.c -Wall -Werror -pedantic -ansi
gcc -c effects.c -Wall -Werror -pedantic -ansi
gcc -c fileIO.c -Wall -Werror -pedantic -ansi
gcc -c LL.c -Wall -Werror -pedantic -ansi
gcc -c lineFuncs.c -Wall -Werror -pedantic -ansi
gcc -g -c main.c fileIO.c LL.c lineFuncs.c -Wall -Werror -pedantic -ansi -D DEBUG=1
gcc main.o effects.o fileIO.o LL.o lineFuncs.o -o TurtleGraphicsDebug -lm
```

TURTLEGRAPHICSSIMPLE

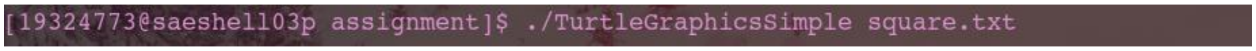


```
[19324773@saeshell03p assignment]$ make simple
gcc -c main.c -Wall -Werror -pedantic -ansi
gcc -c effects.c -Wall -Werror -pedantic -ansi
gcc -c fileIO.c -Wall -Werror -pedantic -ansi
gcc -c LL.c -Wall -Werror -pedantic -ansi
gcc -c lineFuncs.c -Wall -Werror -pedantic -ansi
gcc -c lineFuncs.c -Wall -Werror -pedantic -ansi -D SIMPLE=1
gcc main.o effects.o fileIO.o LL.o lineFuncs.o -o TurtleGraphicsSimple -lm
```

RUNNING THE PROGRAM

COMMAND LINE ARGUMENTS AND INPUT FILE SELECTION

The program is run via the command line in the following format "./Program inputfile" for example to run the program in simple mode with the input file square.txt the user would write "./TurtleGraphicsSimple square.txt"

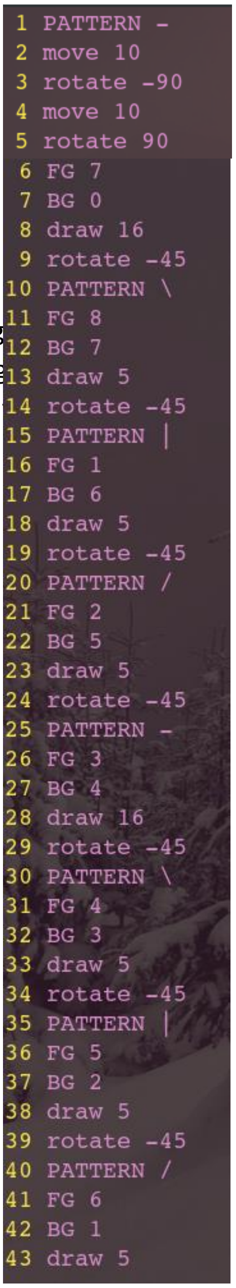


```
[19324773@saeshell03p assignment]$ ./TurtleGraphicsSimple square.txt
```

The program will accept any file type as an input so long as it is plain text and follows the format shown below.

INPUT FILE

The Following input file initially changes the pattern and moves the cursor away from units on both the x and y axis. It then prints an octagon to the screen with alternating text/foreground colours.

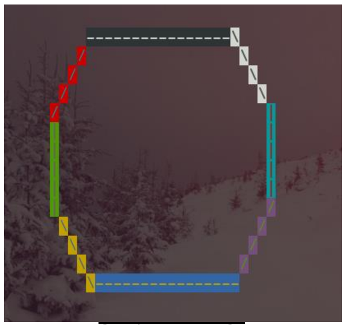


```
PATTERN -
move 10
rotate -90
move 10
rotate 90
FG 7
BG 0
draw 16
rotate -45
PATTERN \
FG 8
BG 7
draw 5
rotate -45
PATTERN |
FG 1
BG 6
draw 5
rotate -45
PATTERN /
FG 2
BG 5
draw 5
rotate -45
PATTERN -
FG 3
BG 4
draw 16
rotate -45
PATTERN \
FG 4
BG 3
draw 5
rotate -45
PATTERN |
FG 5
BG 2
draw 5
rotate -45
PATTERN /
FG 6
BG 1
draw 5
```

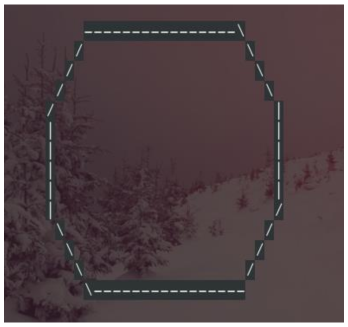
OUTPUT

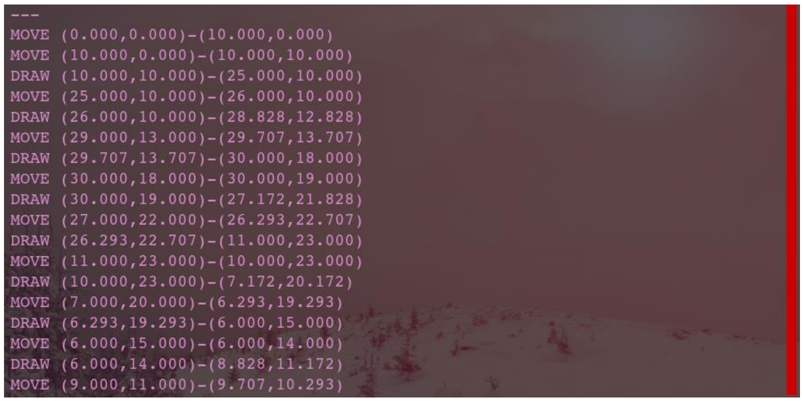
Once the file is validated it will draw the commands to the terminal as well as printing draw commands to a log file called "graphics.log". The output of TurtleGraphicsDebug TurtleGraphics will be the same once ***stderr*** has been piped to a separate terminal or

<u>TurtleGraphics(Debug):</u>



<u>TurtleGraphicsSimple:</u>





```
---
MOVE (0.000,0.000)-(10.000,0.000)
MOVE (10.000,0.000)-(10.000,10.000)
DRAW (10.000,10.000)-(25.000,10.000)
MOVE (25.000,10.000)-(26.000,10.000)
DRAW (26.000,10.000)-(28.828,12.828)
MOVE (29.000,13.000)-(29.707,13.707)
DRAW (29.707,13.707)-(30.000,18.000)
MOVE (30.000,18.000)-(30.000,19.000)
DRAW (30.000,19.000)-(27.172,21.828)
MOVE (27.000,22.000)-(26.293,22.707)
DRAW (26.293,22.707)-(11.000,23.000)
MOVE (11.000,23.000)-(10.000,23.000)
DRAW (10.000,23.000)-(7.172,20.172)
MOVE (7.000,20.000)-(6.293,19.293)
DRAW (6.293,19.293)-(6.000,15.000)
MOVE (6.000,15.000)-(6.000,14.000)
DRAW (6.000,14.000)-(8.828,11.172)
MOVE (9.000,11.000)-(9.707,10.293)
```

COORDINATE SYSTEM

CURRENT IMPLEMENTATION

The terminal is divided into coordinates starting at 0,0 (top left of the terminal) the draw function in effects.c draws lines in reference to this point and taking in the starting x,y values and the ending x,y values. The draw and move commands need to be converted to x,y coordinates from the current coordinates in order for the function to properly draw.

The program starts by reading in the input file checking that each command is from the set of valid commands, passing the command and value into a ***dict struct*** which links the command-value pair into a single datatype. This dict is then passed into the data field of a linked list node inside "inList" which holds all of the commands from the input file once validated. These commands are then read again and values validated based on the type of command as follows.

ROTATE is validated as a float and is implemented as a cyclic float so that any float value can be added or subtracted from the current angle but will stay within 360 degrees. The **MOVE** and **DRAW** functions act in largely the same way using cosine and sine trigonometric functions to calculate the end x,y coordinates based off the current angle and distance value given by the input file and moving the current x,y coordinates to these end values at the end of the function. In the case of the **DRAW** function the starting coordinates and ending coordinates are passed to the ***line*** function as well as the plotter function pointer and the plot data, to print the line to the terminal between those coordinates.

ALTERNATIVE IMPLEMENTATION

With the use of an enum for each of the commands, a switch statement could be used to compare the input file and valid commands, commands could be verified as exactly the type of command not just as a valid command. Value could also be validated within the switch statement and converted to the correct type. With this validation to minimize further iteration through the linked list the dict could have a 3rd variable which acts as a function pointer to the relevant command function. Allowing the list to only be checked once for filling and then iterated through to call the function pointer attached to each node minimizing the computation and iteration of the loop.

FUNCTIONS

FILEIO.C

readFile

This function is intended to take in a filename as a string and a LinkedList to be filled with all of the commands from the following fileIO functions. The function checks that the file exists and is not empty before calling the readLine function. If the above checks or readLine function fails/encounters an error the program will return an error flag as well as printing an appropriate error message.

writeFile

writeFile takes in a filename and a linked list, the filename being the graphics.log file and the linked list being full of strings from the draw and move command. This function opens graphics.log in append mode and inserts each of the strings to the file for debugging purposes.

readLine

readLine takes in a file pointer and a linked list to be filled with all of the commands from the input file. It iterates through the file ensuring that for each line to be inserted in the linked list it is not an empty line or that any file error occurs. It then passes this read in line to the split function to be tokenized and inserted into the linked list.

split

split takes in a string and linked list to be filled with the tokenized halves of the string. It uses sscanf to ensure that only 2 parameters have been read in, using the strCase function to convert first half (command) to entirely uppercase for comparisons. It then validates the command section of the string and inserts the command-value pair into a dict then into the linked list inserting at the tail of the list so it can be read in order later in the program.

strCase

strCase converts the case of a string of alphabetic characters to entirely uppercase it does this by shifting the ascii values of lowercase characters by 32 (the difference between lower case letters and their corresponding uppercase value) and leaving uppercase letters and non-alphabetic characters alone.

cmdCheck

Checks a string against a list of valid commands and simply returns whether the string matches 1 of the valid strings. In debug mode it also prints out the name of the command and whether or not it matches to the stderr stream.

LINEFUNCS.C

exCommand

Iterates through inList of all commands and values validates the values then passes them to the relevant functions with other supporting pointers to be executed. Returns an error Boolean so that main can exit the program if an error occurs.

plot

Plot takes in void pointer which WILL be a char it is simply defined as taking in a void pointer to match the function pointer declaration. The void pointer is then converted to a char pointer and dereferenced before printing the single character. This function will be called multiple times by the line function from effects.c and is used to actually print the pattern to screen.

rotate

Takes in a pointer to the current angle as well as a double new angle to be added/subtracted from the current angle. The function also makes the angle cyclic about 0 and 360 degrees and as such any new angle can be given and it will be converted to its identical value at 360 degrees. The function will also return an error code if the new angle given is invalid.

move
move moves the current cursor (coordinates) using the current angle and distance it calculates the shift in coordinates of the move based on trigonometric functions and then sets the current coordinates to these values. It calculates the end x coordinate through the use of the cosine function and the y coordinate using the sine function. Adding the corresponding value to the x coordinate and subtracting it from the y as for standard trigonometric functions 0,0 is the bottom left and as such needs to be inverted to get the correct coordinates for y when 0,0 is top left.

draw
draw draws a line between the current coordinates and the end coordinates. Using the current angle and distance it calculates the shift in coordinates of the move based on trigonometric functions and then sets the current coordinates to these values. It calculates the end x coordinate through the use of the cosine function and the y coordinate using the sine function. Adding the corresponding value to the x coordinate and subtracting it from the y as for standard trigonometric functions 0,0 is the bottom left and as such needs to be inverted to get the correct coordinates for y when 0,0 is top left. It rounds both the starting values and end values to produce straight lines at 45 degree angles as without round there is a slight inaccuracy due to the floating point nature of the coordinates and as such lines in shapes become skewed. It also calls the line function from effects.c to print the current pattern to the screen between the coordinates calculated.

deg2rad
deg2rad converts degrees given as a double into radian units as a double. This conversion is as per the formula (degrees * pi)/180 this conversion is necessary as the math.h library defines cosine and sine functions in radians rather than degrees.

round
takes in a double and returns a rounded int. Used since there is no standard rounding method for converting floating point numbers to integers in c. it makes use of the ceil and floor functions from the math library, to accurately determine whether a double should be rounded up or down then returns the rounded value as an integer.

LL.c

createList
Creates a new linked list and allocates it the required memory on the heap. Initializes the head and tail to NULL and the size to zero. Used for the initial creation of new lists, simplifying the initialization process.

createNode
Creates a new LLNode and allocates it the required memory on the heap. Initializing it with a NULL next,prev and data. Used for adding new nodes to a list, simplifying the initialization

process. It is not used for temporary LLNode's used to iterate through a list as the function would double assign memory to the heap and cause memory to be leaked when freeing.

InsertFirst
Takes in a linked list and a new node inserts the new node at the head of the list and joins it to the second node in the list if one or more nodes already in the list.

InsertLast
Takes in a linked list and a new node inserts the new node at the tail of the list and joins it to the second last node if one or more nodes already in the list.

removeFirst
Takes in a linked list and removes the node at the head. If the list has more than one node in it, the node will be made to point to the second node. The removed node is then freed and the size of the list is decremented.

removeLast
Takes in a linked list and removes the node at the tail. If the list has more than one node in it, the node will be made to point to the second last node. The removed node is then freed and the size of the list is decremented.

freeList
Takes in a linked list using a temporary current node and next node, it increments its way through the list freeing the current node and moving onto the next node until all of the nodes are freed. If the is only one element in the list it is simply freed and the head and tail set to null. If there is no elements in the list a warning is given. The list is then freed regardless of if there was previously elements in it or an empty list was passed to it.

MAIN.C

startIO
A function purely to kick off the IO process and ensure no errors occur in the reading of the file. In debug mode it prints that it is beginning read to stderr and whether it encounters an error.

main
Initializes all of the relevant pointers, variables and lists and assigns them memory, accepts the accepts only 2 command line arguments the first of which being the program name and the second being the input file. It then runs through the read functions of fileIO the execution of each command from the input file and the printing to the graphics.log file. In debug mode it informs the user of which section an error occurs in and quits the program. In all versions regardless of errors the program frees all memory before quitting and returns the terminal to the default colour scheme.