UCP Assignment Report

BY AARON GANGEMI (19447337)

Table of Contents

Functions in file: TurtleGraphics.c

void executeRotate(char* value, double* currentAngle):

The execute rotate function is called when the command in the linked list is “rotate”. The functionality is designed to change and keep track of the current angle. When the graphics are drawn, the rotate function is responsible for the direction the line is drawn in. For example, -90 will rotate the line 90 degrees in an anti-clockwise direction. This function takes the value which is the rotation value and a pointer to the current angle.

void executeMove(char* value, double *XCoordinate, double *YCoordinate, double currentAngle, FILE* appendFile);

The execute move function is called when the command in the linked list is “move”. The functionality of this function moves the printing to a certain position on the terminal. In addition, this function calls getXCoordinate() and getYCoordinate() which gets the values of X2 and Y2, being the new coordinates for X and Y. This is done to keep track of the current movement of the line. Although this function and executeDraw() are similar in many ways, the difference is that this function only moves the cursor and does not draw the line. This function takes the value of move, the current X and Y Coordinates(X1 and Y1), the current angle and the file to append to. The append file parameter is used to print coordinates to file after move is called.

void executeDraw(char* value, double *xCoordinate, double *YCoordinate, char pattern, double currentAngle, FILE* appendFile);

The execute draw function is the most important function in this program. It is solely responsible for drawing the graphics by calling the provided line function. In order to do this, the function first calculates X2 and Y2, being the new X and Y coordinates. From here, X1,Y1,X2 and Y2 are all passed into the line function, and the line is printed to the terminal. This function takes the value of draw, X1, Y1, the current pattern which is the value of the void pointer for the plotter function, the current angle to calculate X2 and Y2 and the file to append the coordinates to.

void executeFg(int *foreground, char* value);

The execute foreground function simply keeps track of the current foreground colour for the terminal. After this function is updated and tracked, it calls setFgColour() which changes the foreground colour. This function takes in a pointer to the current foreground and the value of foreground. The #ifdef simple is used for the makefile to only draw black and white graphics. This function is called if the command in the linked list is “fg”.

```
void executeBg(int* background, char* value);
```

The execute background function simply keeps track of the current background colour for the terminal. After this function is updated using the background pointer, it calls setBgColour() which changes the background colour of the terminal. This function takes a pointer to the current background colour and the value of the background. The #ifdef simple is used for the makefile to only draw black and white graphics. This function is called if the command in the linked list is “bg”.

```
void executePattern(char* value, char* pattern);
```

The execute pattern function keeps track of the current pattern. This value is used for the plotter function which draws the current pattern character. This function takes the pattern character and pattern and is called when the linked list value is “pattern”.

```
int getXCoordinate(double distance, double angle);
```

The getXCoordinate() function calculates X2. The X value is calculated by multiplying the distance by cosine(angle in radians). This value is then rounded to the nearest integer as required for the line function. The rounding functionality was taken by and referenced a user on stackoverflow.com.

```
int getYCoordinate(double distance, double angle);
```

The getYCoordinate() function calculates Y2. The Y value is calculated by multiplying the imported distance by sin(angle in radians). This value is then rounded to the nearest integer as required for the line function. The rounding functionality was taken by and referenced a user on stack overflow.

```
int main(int argc, char* fileName[])
```

The main function takes in the file name which contains the graphics. This concept is validated throughout the program. The main function validates the number of parameters. If the number of parameters is valid and the file is valid, it then calls the read into the linked list function. The Main() contains a big if-else-if statement which iterates through the list and executes through the list. In addition, the main frees the linked list to ensure there are no memory leaks.

Functions in file: TurtleIO.c

void readLines(char* fileName, LinkedList* list);

The read lines function takes in the filename given on command line and the linked list created in main. The read lines function first validates several aspects of the file to check if it is valid. This validation incorporates the following:

- Lowercase and uppercase checking
- The file being empty
- Lines that do not start with one of the specified commands
- Incorrect number of parameters after a command
- Incorrect datatype after a command.

If any of these validations return false, then the program will either finish printing at the current point and not read in the rest of the file to the list, or go back to the terminal and ask the user to correct the file. Assuming these are valid, the read lines function will then read in the command and use the struct to pass it into a linked list.

void convertToLowercase(char command[]);

The convertToLowercase() function takes in the current command. The specification states that the line is valid if a command is either uppercase, lowercase or both. This command switches the command to uppercase, this ensuring the command is valid.

void checkEmptyLine(int noOfLines);

The checkEmptyLine() function checks the file is not empty. It takes the number of elements read in by the fscanf() function. If this function returns -1, then the file is invalid and the program will return to the terminal and ask the user to fix the file.

int checkNoOfElements(int noOfElements, char* command, char* value, FILE* inputFile);

The checkNoOfElements() function validates the number of parameters on command line. For example, if a line was

“ROTATE 45 64”

This function would return the line as invalid and stop drawing the graphics once the line is read in. To do this, checkNoOfElements returns EOF.

int validateDataType(char* command,char* value);

The validateDataType() function validates that the datatype after a command is valid. To do this it uses the return of atoi() and atol(). If these functions return 0, then the value could not be converted, hence making it invalid.

```
double convertDouble(char* value);
```
The convertDouble() function takes in the value after a command and converts it to a double.

```
int convertInt(char* value);
```
The convertInt() function takes in the value after a command and converts it to an integer.

Functions in file: LinkedList.c

Structs:

- LinkedListNode:
  - Void* data: pointer to the data of the node
  - LinkedListNode* next: pointer to next node
- LinkedList:
  - Head: Pointer to first element of list
  - Tail: Pointer to last element of list
- newString:
  - newCommand: stores command in file
  - newValue: stores value in file

LinkedList* createList();

The createList() function simply creates a new Linked List by initializing the head and tail to NULL.

void insertLast(newString line, LinkedList* list);

The insertLast function inserts the struct line into the end of the created linked list.

void freeLinkedList(LinkedList* list);

The freeLinkedList() function frees the data and the nodes in the linked list to ensure that there are no memory leaks.

int isEmpty(LinkedList* list);

The isEmpty function is used to check whether or not the list is empty. This is used for insertion of the list. The function returns an int which is an indicator in C for the Boolean value.

void printList(LinkedList* list);

The printList() method is solely used for the testing of the list. For example, I know that insert list last is working if the list correctly prints out.

How I converted the input file to a coordinate system:

After reading in the file, I created a coordinate system by keeping X and Y as separate variables. X knows nothing about Y and Y knows nothing about X. This was purposely done as I found it easier to keep track of the coordinates for passing. The only time they come together is when they are called for the line function.

Furthermore, the X and Y Coordinates are created in the getXCoordinate() and getYCoordinate() functions. These functions assisted me in distinguishing between X1, X2, Y1 and Y2. The implementation of the coordinates for my program were more about simplicity and ease of access for the line() function. In order to keep track of the coordinates, I pass the reference to the memory address when iterating through the linked list.

Alternative approach:

An alternative approach to the coordinates system would be to store the coordinates in a struct. By storing the coordinates in a struct, you could easily pass the entire struct rather than passing each individual coordinate. This would achieve the same outcome.

Examples of the program working:

Command line used to execute the program and user input provided:

```
[19447337@saeshell01p Assignment]$ ./TurtleGraphics Test.csv
```

Contents of the input file: Should draw a square (See below)

```
1 Draw 10
2 Rotate -90
3 Draw 10
4 Rotate -90
5 Draw 10
6 roTate -90
7 drAw 10
```

Contents of the output file

```
1 ---
2 draw(0.000000,0.000000)-(9.000000,0.000000)
3 draw(10.000000,0.000000)-(10.000000,9.000000)
4 draw(10.000000,10.000000)-(1.000000,10.000000)
5 draw(0.000000,10.000000)-(0.000000,1.000000)
```