# Developing software for a drawing robot

William Alsbury [20466404]

## Software Description:

The software developed for this project is designed to generate G-code from a sentence stored in a .txt file. This G-code is then transmitted to a writing robot, which uses it to 'draw' the specified sentence. For the process to succeed, the software must meet all project requirements, ensuring that the robot produces a cohesive and grammatically correct output.

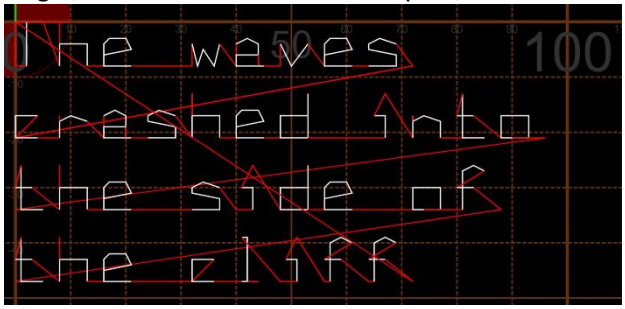
The program begins by defining constants for array sizes and key parameters to ensure proper memory allocation and program logic. For example, #define MAX_CHARACTERS 256 specifies the maximum number of characters the font system can handle, matching the total number of ASCII characters.

Next, data structures are defined for both pen movements and character representations. These structures detail the positional and state data required to render a single character. Using the #define MAX_CHARACTERS 256 directive, an array named fontData is declared to store font information for all supported characters.

With the foundational structures in place, the program constructs its core functions:

1. **openFile()**: This function opens the SingleStrokeFont.txt file and handles any errors encountered during file access.
2. **loadFontData()**: Reads the contents of SingleStrokeFont.txt line by line, extracting movement data for each character and storing it in the fontData array.
3. **scaleFontData()**: Scales the character movement data to ensure the robot draws the text at the correct user-specified height.
4. **processWord()**: Processes individual words from the input text, generating G-code commands to render them on the surface. It uses the font data stored in fontData to determine pen movements, checks if words fit within the drawing area, handles line breaks, and sends G-code commands for the pen's actions. This function is called by generateGCode().
5. **generateGCode()**: Orchestrates the text rendering process. It first scales the font data using scaleFontData() and then calls processWord() to convert input text into G-code commands. It handles text layout, manages word and line breaks, and sends the necessary G-code to the robot to control movement and pen state.
6. **SendCommands()**: Sends G-code commands to the robot, ensuring proper synchronisation. This function is called by processWord(), generateGCode(), and main() whenever commands need to be transmitted.

The **main()** function coordinates the program's workflow, starting by initializing the RS232 port and establishing communication with the robot. It then wakes up the robot with initialization G-code commands and loads font data from the SingleStrokeFont.txt file. The user is prompted to specify the text height (4–10 mm) and the name of the .txt file containing the desired sentence. After validating inputs, the program calls generateGCode() to scale the font data and convert the text into G-code, which is transmitted to the robot for drawing. Finally, the program resets the robot to its origin, closes the communication port, and exits.




**This is the custom shape generated by the program when the height is set to 8mm. The image is from the G-code simulator.**

# Project Files:

**'main.c' –** The 'main.c' file contains all the code to convert the font file data into processible G-code that is specific to text within the desired .txt file. It initialises the connection with the RS232 port which then allows the G-code to be pulled to the robot.

**'rs232.c' –** This code file facilitates serial (RS232) communication management across multiple operating systems. It provides functions for opening, configuring, and handling interactions with serial ports.

**'rs232.h' –** This header file code defines functions and macros to facilitate the use of serial ports, such as: opening and closing serial ports, configuring port settings (e.g., baud rate), sending and receiving data over the serial interface.

**'serial.c' –** This file opens, configures, and interacts with a serial port by pushing the data calculated in the 'main.c' file through to the port. The file also provides the ability to test main function code without actual RS232 communication by either commenting or uncommenting a placeholder function.

**'serial.h' –** This file is there for the same reason as the 'rs232.h' file. The declaration of the functions within this file provides information about the functions that will be implemented elsewhere in the corresponding .c file.

## Key Data Items:

| Name | Data type | Rationale |
|---|---|---|
| Movement | Struct | Keeps all the coordinate related data together making it easy to access later in the program. This reduces complexity and reduces the risk of inconsistency and errors. |
| Character | Struct | Like the 'Movement' rationale, keeps related character data together for easy access later in the program, whilst also containing an array for the movements of each character. |
| Buffer[] | Char | A char array is appropriate because char is designed for storing characters. The size of 256 is chosen as a reasonable upper limit for storing a line of text or a command. It's large enough to handle most G-code commands while remaining efficient. |
| Word[] | Char | This buffer accumulates characters in a word when generating G-code. The size of 128 is chosen because it is large enough for typical words while not wasting too much memory. |
| Height | Float | Heights often need to represent fractional values. Using a float data type allows you to store non-integer values, which is necessary when the measurement isn't a whole number. |
| scaleFactor | Float | The scale factor incorporates the height value, which necessitates using a float to avoid warnings related to implicit type conversion from float to int. |
| x_pos, y_pos | Int | An int is used because X and Y coordinate values are usually represented as whole numbers and int provides sufficient range and efficiency for this purpose. |
| CharWidth | Int | This stores the width of each character. It is an integer because width is measured in discrete units (movement units), and int is ideal for such measurements. |
| maxLineWidth | Int | This was a requirement set in the brief, so setting this as an int is perfectly acceptable to prevent words from crossing the line. |
| PenState | Int | Tracks the pen's state (up or down) as an int value for simplicity and efficiency. |
| openFile | FILE* | By using FILE*, the program ensures efficient, flexible, and reliable file access while keeping the code straightforward and reusable. |
| textFileName[] | Char | It efficiently stores and manipulates the file name as a sequence of characters. Also allows for straightforward handling of user input and ensures memory efficiency compared to alternatives. |

# Functions:

**FILE *openFile(const char *filename, const char *mode)**

This function opens the SingleStrokeFont.txt file and handles any errors encountered during file access.

**Parameters:**

- **Filename –** The name of the file to open.
- **Mode** – The file access mode
- **Return Value** – Returns a pointer to the opened file. If the file cannot be opened, the program terminates and displays an error message.

**void loadFontData(const char *filename)**

Reads the contents of SingleStrokeFont.txt line by line, extracting movement data for each character and storing it in the fontData array.

**Parameters:**

- **Filename -** The name of the file containing font data.
- **Return Value –** As it is a void function there is no return value.

**void scaleFontData(float height)**

Scales the character movement data to ensure the robot draws the text at the correct user-specified height.

**Parameters:**

- **Height –** The height inputted by the user.
- **Return Value -** As it is a void function there is no return value.

**void processWord(const char *word, int *x_pos, int *y_pos, int *penState, int charWidth, int maxLineWidth, int *lowestY, int lineGap, int minY)**

Processes individual words from the input text, generating G-code commands to render them on the surface. It uses the font data stored in fontData to determine pen movements, checks if words fit within the drawing area, handles line breaks, and sends G-code commands for the pen's actions.

**Parameters:**

- **Word -** The word to process into G-code commands.
- **X_pos -** Pointer to the current X-coordinate position.
- **Y_pos -** Pointer to the current Y-coordinate position.
- **penState -** Pointer to the current pen state.
- **charWidth -** The width of each character.
- **maxLineWidth -** The maximum width of the drawing area.
- **lowestY -** Pointer to track the lowest Y-coordinate reached.
- **lineGap -** The gap between lines in the drawing.
- **minY -** The minimum allowed Y-coordinate.
- **Return Value -** As it is a void function there is no return value.

**void generateGCode(const char *text, float height)**

It first scales the font data using scaleFontData() and then calls processWord() to convert input text into G-code commands. It handles text layout, manages word and line breaks, and sends the necessary G-code to the robot to control movement and pen state.

**Parameters:**

- **Text –** The input text to convert to G-code.
- **Height –** The height inputted by the user.
- **Return Value -** As it is a void function there is no return value.

# Testing Information:

| Function | Test Case | Test Data | Expected Output |
|---|---|---|---|
| Main() | Execution of code with valid data | Valid serial connection, input of 'SingleStrokeFont.txt', input height of 8mm, input of valid written text file containing ‘Green Grass’. | ‘Green Grass’ is drawn with correct spacing, and pen returns to (0,0). |
| openFile() | Attempt to open a valid file. | 'SingleStrokeFont.txt' | File handle returned if successful. |
| loadFontData() | Read and store valid font data from a file. | A valid path to ‘SingleStrokeFont.txt’. | fontData is populated with character movement data from the file. |
| scaleFontData() | Scale the font data to a specific height. | Input height of 5mm. | Movements in fontData are scaled proportionally to a 5mm height. |
| processWord() | Generate G-code for a word that fits within the line width. | Word = ‘Orange’, fontData with data for each letter. | G-code commands for drawing "Orange" are generated within line width. |
| generateGCode() | Generate G-code for a valid string of text. | Text = “The sky is blue”, input height of 7mm. | Complete G-code commands for the text are output, respecting layout and line breaks. The text is also scaled to the correct height. |
| SendCommands() | Send a valid G-code command. | Command = “G0 X0 Y0” | The pen returns to the origin (0,0) |

## Flowchart(s):

Located in PDF in ZIP file.