RMIT University

EEET2490 – Embedded System: OS and Interfacing, Semester 2025-1

GROUP ASSIGNMENT

OBJECTIVES

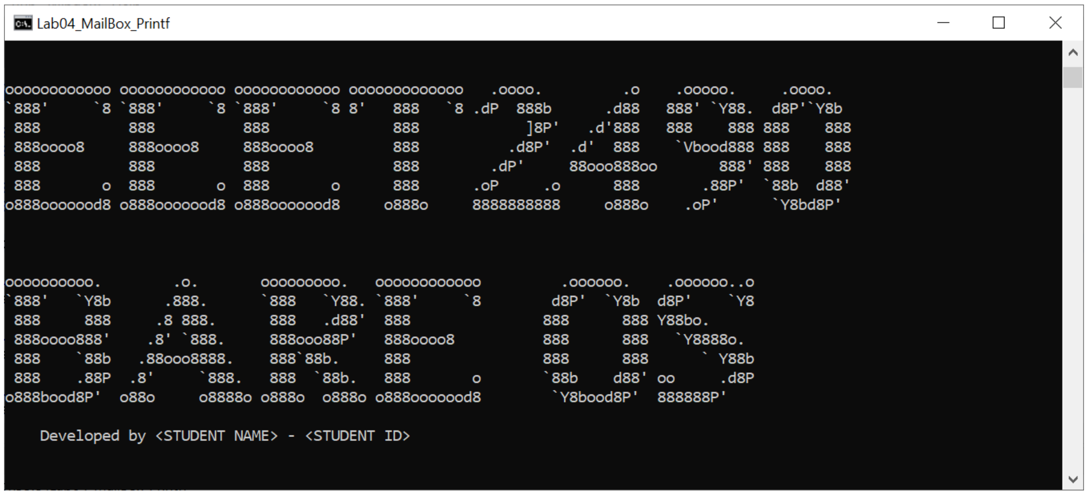
In this group project, students as a team work together to further sharpen their skills and enrich knowlege in OS and peripheral driver development.

DESCRIPTION

Students should read all of the tasks below before starting of implementation, since one task may support another task.

1. Welcome Message and Command Line Interpreter (CLI) (~30%)

Normally when an OS successfully boot up, it will show a **Welcome Message** (text/ screen/ sound). For your Bare Metal OS, you should also have a welcome text like below.

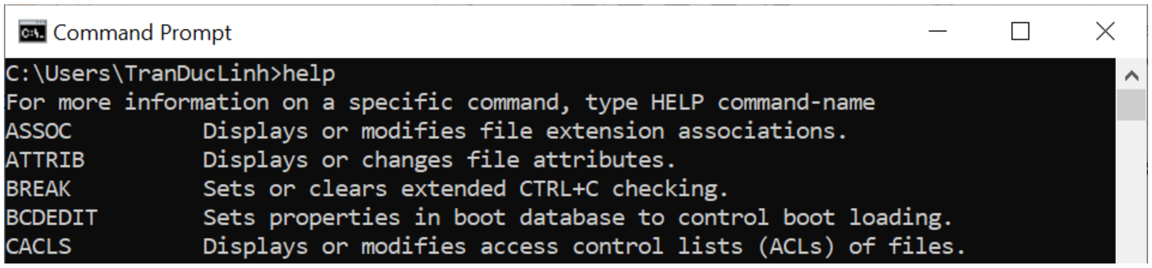


```
oooooooooooo oooooooooooo oooooooooooo ooooooooooooo   .oooo.         .o    .ooooo.     .oooo.
`888'     `8 `888'     `8 `888'     `8 8'   888   `8 .dP  888b      .d88   888' `Y88.  d8P'`Y8b
 888          888          888              888            ]8P'   .d'888   888    888 888    888
 888oooo8     888oooo8     888oooo8         888          .d8P'  .d'  888    `Vbood888 888    888
 888          888          888              888        .dP'     88ooo888oo       888' 888    888
 888       o  888       o  888       o      888      .oP     .o      888       .88P'  `88b  d88'
o888ooooood8 o888ooooood8 o888ooooood8     o888o     8888888888     o888o    .oP'      `Y8bd8P'



oooooooooo.        .o.       ooooooooo.   oooooooooooo         .oooooo.    .oooooo..o
`888'   `Y8b      .888.      `888   `Y88. `888'     `8        d8P'  `Y8b  d8P'    `Y8
 888     888     .8 888.      888   .d88'  888               888      888 Y88bo.
 888oooo888'    .8' `888.     888ooo88P'   888oooo8          888      888   `Y8888o.
 888    `88b   .88ooo8888.    888`88b.     888               888      888      ` Y88b
 888    .88P  .8'     `888.   888  `88b.   888       o       `88b    d88' oo     .d8P
o888bood8P'  o88o     o8888o o888o  o888o o888ooooood8        `Y8bood8P'  888888P'

    Developed by <STUDENT NAME> - <STUDENT ID>
```

<u>Supporting Resource</u>: You may use this tool https://onlineasciitools.com/convert-text-to-ascii-art to help convert any text into an ASCII art string, then just print it out to the console. You are free to select any font and customize the welcome message.

Furthermore, **Command Line Interpreter (CLI)** is an indispensable feature of every operating system. For example, Windows 10 provides two CLIs for their users - Command Prompt and PowerShell, whereas Linux users use Shell on a regular basis.

Example of some commands in Windows' Command Prompt



```
C:\Users\TranDucLinh>help
For more information on a specific command, type HELP command-name
ASSOC          Displays or modifies file extension associations.
ATTRIB         Displays or changes file attributes.
BREAK          Sets or clears extended CTRL+C checking.
BCDEDIT        Sets properties in boot database to control boot loading.
CACLS          Displays or modifies access control lists (ACLs) of files.
```

For your OS, implement a **command line interpreter** that reads keypresses into a buffer and then executes the commands when the key "Enter" is pressed. Implement some feature below for your CLI:

- An **OS name (e.g. MyOS)** as *initial text that is always displayed in the console while waiting for user to type commands*
- **Auto-completion:** user can use TAB for auto-completion of the command name and options.
- **Command history:** to browse within command history. Typically the user use UP and DOWN arrow to do so. However, these special keys are not read by the UART, thus, please use **_** key as UP arrow and **+** key as DOWN arrow for this feature.

Implement **some commands as below** to test your CLI:

No.	**Comand Name**	**Usage**
1	`help`	- Show brief information of all commands - Example: **`MyOS> help`**
	`help <command_name>`	- Show full information of a specific command - Example: **`MyOS> help showinfo`**
2	`clear`	- Clear screen *(in our terminal it will scroll down to current position of the cursor).* - Example: **`MyOS> clear`**
3	`showinfo`	- Show **board revision (value and information)** and **board MAC address** in correct format - Example: **`MyOS> showinfo`**
4	`baudrate`	- Allow the user to **change the baudrate of current UART being used**. The command must support various baud rates, including but not limited to 9600, 19200, 38400, 57600, and 115200 bits per second.
5	`handshake`	- Allow the user to **turn on/off CTS/RTS handsharking on current UART if possible**

Note: Your CLI should allow users to **delete characters while typing** (but does not over delete the OS name), and should be developed in a user-friendly maner. In addition, there should be **commands to run for tasks 2 and 3** of the assignment.

Supporting Resource:

- **Information for Board Revision** can be found here: https://www.raspberrypi-spy.co.uk/2012/09/checking-your-raspberry-pi-board-version/
- **Information for MAC address:** https://www.javatpoint.com/what-is-mac-address

2. Image, Video, and Text Display (~20%)

So far, you have learnt all the basics about screen display, and can work well with it to draw with specific position and color. It is quite easy to draw a line, rectangle, circle on the screen. However, how about displaying images and videos?

In this task,

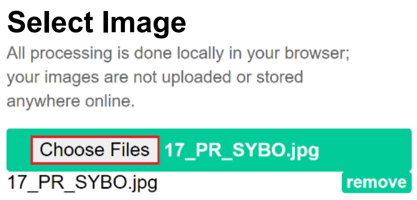
i. Display **names of all team members** on top of a background image on the screen. You should create your own font data and use several <u>different colors</u> for the text.

ii. We can display images and text, how can we display a **video** on the screen? Think of it and find the answer for yourself. **Record any video by yourself (e.g. RMIT campus scene) and display it on the screen**. For the purpose of demonstrating, you only need to display a very-short video (just several seconds).

<u>Supporting Resource</u>:

- This tool can help you to **convert an image into ARGB32 data** (select RGB888 option), which is compatible with our Raspberry Pi: https://javl.github.io/image2cpp/

  Note that, the tool convert an image into an array of data, in which each element is 32-bit data for each pixel. Follow the steps below

1. 

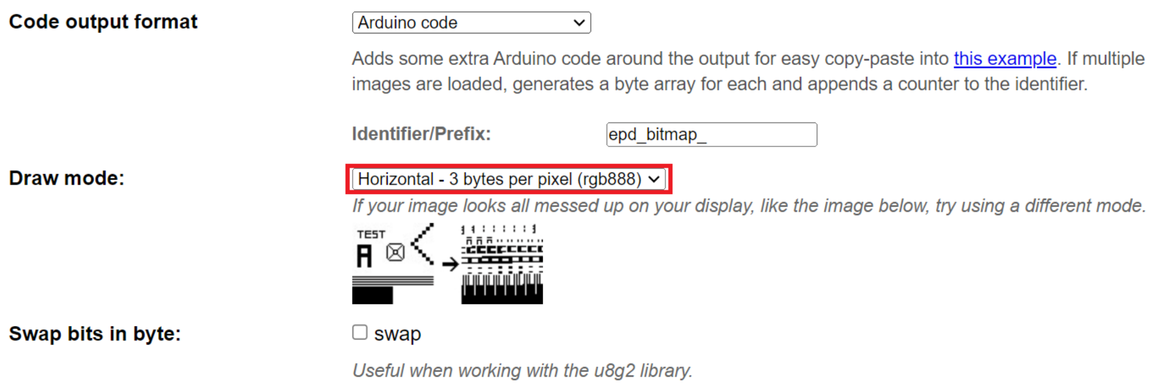
   Select Image

   All processing is done locally in your browser; your images are not uploaded or stored anywhere online.

   Choose Files 17_PR_SYBO.jpg

   17_PR_SYBO.jpg remove

2. **Setting for Output: select 3 bytes per pixel (rgb888)**



Code output format Arduino code

Adds some extra Arduino code around the output for easy copy-paste into this example. If multiple images are loaded, generates a byte array for each and appends a counter to the identifier.

Identifier/Prefix: epd_bitmap_

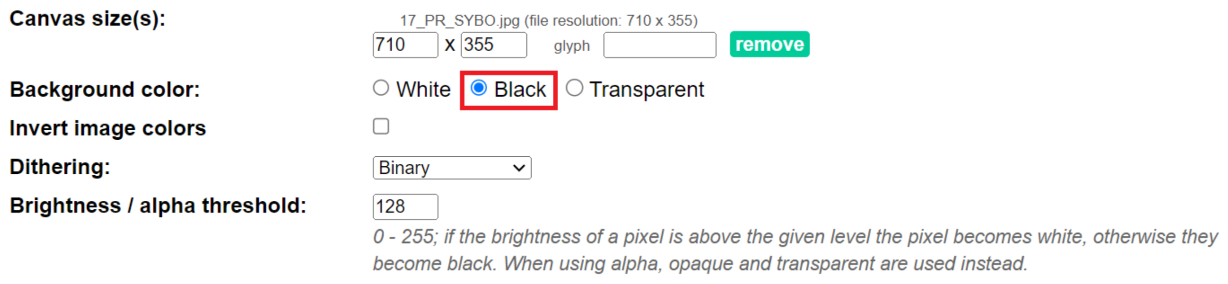
Draw mode: Horizontal - 3 bytes per pixel (rgb888)

If your image looks all messed up on your display, like the image below, try using a different mode.

Swap bits in byte: ☐ swap

Useful when working with the u8g2 library.

3. **Image Settings: select Black for Background color**



Canvas size(s): 17_PR_SYBO.jpg (file resolution: 710 x 355)
710 x 355 glyph [ ] remove

Background color: ○ White [✓ Black] ○ Transparent

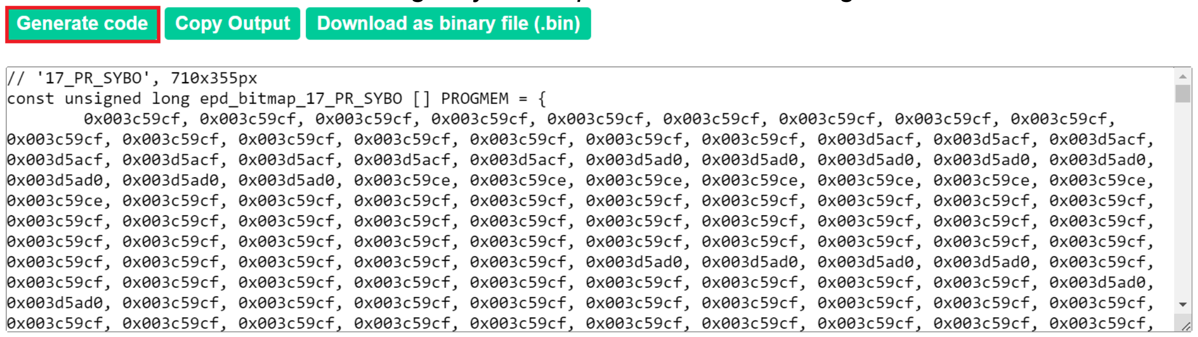
Invert image colors ☐

Dithering: Binary

Brightness / alpha threshold: 128
0 - 255; if the brightness of a pixel is above the given level the pixel becomes white, otherwise they become black. When using alpha, opaque and transparent are used instead.

4. **Check Preview:** *it should show your uploaded image with correct color. If not, reload the webpage and start again*

5. **Generate Code:** *it should now give you the pixel data of the image*



```
// '17_PR_SYBO', 710x355px
const unsigned long epd_bitmap_17_PR_SYBO [] PROGMEM = {
	0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 
0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003d5acf, 0x003d5acf, 0x003d5acf, 
0x003d5acf, 0x003d5acf, 0x003d5acf, 0x003d5acf, 0x003d5acf, 0x003d5ad0, 0x003d5ad0, 0x003d5ad0, 0x003d5ad0, 0x003d5ad0, 
0x003d5ad0, 0x003d5ad0, 0x003d5ad0, 0x003c59ce, 0x003c59ce, 0x003c59ce, 0x003c59ce, 0x003c59ce, 0x003c59ce, 0x003c59ce, 
0x003c59ce, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 
0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 
0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 
0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003d5ad0, 0x003d5ad0, 0x003d5ad0, 0x003d5ad0, 0x003c59cf, 
0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003d5ad0, 
0x003d5ad0, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 
0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 0x003c59cf, 
```

- **Method to extract frames from video**

  To display a video, it is simply just to display all frame images of it. Thus, use a tool to extract frame images from your video. Search for the keyword "**extract frames from video**" and there is a plenty of tools and ways to do so.

- **Method to create font**
  Besides images and videos, displaying a **text** on screen is a conventional need. Basically, it is similar to the way that we draw any thing on the screen – we need to draw at corresponding pixels. However, to make it simple, we should have **FONT** which defines pattern of each single character glygh.

  The pixel data for the character glyphs is usually stored as a bitmap. In a bitmap, each pixel is represented by a single bit. If the bit is 'ON' (value 1), the pixel is to be drawn in the foreground color; if 'OFF' (value 0), the pixel is set to the background color (hidden).

Example of letter “A” in an 8x8 FONT:

```
0 0 0 0 1 1 0 0 = 0x0C
0 0 0 1 1 1 1 0 = 0x1E
0 0 1 1 0 0 1 1 = 0x33
0 0 1 1 0 0 1 1 = 0x33
0 0 1 1 1 1 1 1 = 0x3F
0 0 1 1 0 0 1 1 = 0x33
0 0 1 1 0 0 1 1 = 0x33
0 0 0 0 0 0 0 0 = 0x00
```

The letter A above can be represented by just an array of 8 bytes : {0x0C, 0x1E, 0x33, 0x33, 0x3F, 0x33, 0x33, 0x00}.

Thus, to have a font, you can get an existing font image, such as at https://fontmeme.com/fonts/static/235021/autolova-font-character-map.png then convert the image of each character into pixel data and implement it for your OS.

References: those tutorials listed below could be good references to look at:

[1] https://github.com/bztsrc/raspi3-tutorial/tree/master/0A_pcscreenfont

[2] https://github.com/babbleberry/rpi4-osdev/tree/master/part5-framebuffer

3. App Development For Bare Metal OS (~50%)

Develop a small game that player(s) use screen and keyboards to play the game.

Your game should have colors (rather than just black and white). Basic requirement is to only have one fixed stage/level, excellent goods should have at least two game stages/levels.

For observation and troubleshooting purposes, the communication between the game application and the CLI terminal should be as follow:

- **Terminal Input**: The terminal can send characters to control the game; the game should respond with **ACK** (acknowledgment) or **NAK** (negative acknowledgment, meaning something's gone wrong) depending on the situation.
- **Data Logging**: The game should log information to the terminal. This technique is useful for troubleshooting issues over the long run. Logs should include:
    - The number of commands received.
    - Any other useful information that students like to send

Criteria for evaluation:

- Complexity of the game application
- How well and user-friendly it is developed (variables, resources usage, coding style, UI…)
- The communication between the game application and the CLI terminal.
- Creativity: a little of creativity is welcome, but not compulsory. It is okay to get the idea from existing game (but don’t completely copy and paste source code of existing games or works of other students).

*Note that it is required to **test with the real board** for all tasks (after you complete the development with QEMU).*

Advice: Developing a small game is actually not difficult. If you haven't developed any game before, my advice is to try making a simple game on an drag-and-drop platform like Scratch https://scratch.mit.edu/

Try with **Make an Apple Catcher Game:** www.youtube.com/watch?v=jFVJdRLZoQ4

Or **How to Make a Shooter Game**: www.youtube.com/watch?v=QXru0rSV2ZQ

Implement a game with the guide above in Scratch to understand how we can make a game with Background image, sprites (player, opponent, obstacles or other objects in your game), and their interactions. Then, you can think of your own game and implement it in C in a similar way.

Report and Presentation

Complete your **report** with discussion of all parts. You should provide some background information, explanation for your work, and also include screenshots of output as evidence of a successful outcome. *A discussion on success and limitation of the outcome would be also meaningful.*

For presentation, please record a **video** of maximum 20 mins that present, explain and demonstrate your work. All members should present in the demo video.