School of Science

Computer Science

Loughborough University

# 24COA202: Embedded Systems Programming

**Coursework specification**

Dr Mohamad Saada Semester 1 2024-2025

## 1 Introduction

This coursework forms 100% of the assessment for this module. There are several parts to the coursework. A main exercise (known as BASIC) and several additional parts to allow you to demonstrate more knowledge (EXTENSIONS). These are all detailed below. If anything is not clear as you read, please read through to the end of the document, it may be clear later. If still not, then send me an email.

## 2 Payroll management system

The aim of this exercise is to program the Arduino as a payroll management system. It will read custom input messages over the Serial Interface, process them, and then display the result on the screen. The Arduino's buttons will allow for interaction with the information displayed on the LCD. First, we will introduce the concepts of custom messages and payroll accounts, then how things should be displayed on the Arduino Screen and the protocol that will be implemented across the Serial Monitor, with some examples.

### 2.1 Serial messages and payroll account information information

The payroll management system receives communication over the Serial Interface in a series of custom messages. These are defined as follows:

- After an initialisation phase, each message is a text string sent as a sequence of characters terminated by a "Newline" or "Carriage return" control character. These are sent over the serial interface either through the Serial Monitor or from a separate host program.

  - "`Carriage return`" or "`CR`" programmatically written as "`\r`" is a control character that makes the printing cursor go to the beginning of the line. This is sent from Python with a print('`\r`') or from the Serial monitor by selecting "`Carriage return`" from the drop-down at the bottom of the window.
  - "`Newline`" or "`NL`" or "`Line Feed`" or "`LF`" programmatically written as "`\n`" is a control character that makes the printing cursor go one line down. This is sent from Python with a print('`\n`') or from the Serial monitor by selecting "`Newline`" from the drop-down at the bottom of the window.

- Payroll accounts are represented by an employee id numbers, which are made of 7 numbers in the range `0-9`, spaces are not allowed.
- Each payroll account has a single letter denoting the employee's job grade which can be from 1 to 9.
- Each payroll account has a job title text to indicate the job title of an employee. This is a string between 3-17 characters long from the range A-Z and 0-9, _ and . (period) (e.g. `Software_Engineer`, `HR_Manager`, `Guard`, etc.), spaces are not allowed, and location string cannot be empty.
- Each payroll account has a salary field to show each employees pre_tax salary, which is a number, spaces are not allowed, a salary of 0 will be given upon receipt of account creation message.
- Each payroll account has an indicator for whether the employee has enrolled for workplace pension or not it is `PEN` or `NPEN` (i.e. `On Pension` or `Not On Pension`). These values are provided across the serial monitor as `PEN` or `NPEN`, and special display colours should be used depending on the status of pension enrollment.

The protocol for communication is detailed later in the document, but first look at what is to be displayed.

## 2.2 Arduino Display

The Arduino presents all the information of all the payroll accounts that are managed by the payroll management system.

This information is displayed on its screen and provides an interface via the buttons for the user to examine them. We describe this interface in this section.

- The display has two lines and shows the information of a single payroll account providing an up-down scrolling facility to see others.

- The display must be of the form:

```
+----------------+
|^A BCDE £££££.££|
|v1234567 FGHIJKL|MNOPQRSTUV
+----------------+
```

  **Note** the ^ and v are arrow indicators for up and down. There is one space between job grade (A) and pension status (BCDE), and there is one space between pension status (BCDE) and salary amount (£££££.££), salary should display two numbers after decimal point, even if 00. Also, there is one space between payroll account number (1234567) and job title text (FGHIJKLMNOPQRSTUV),

  Note the exact positioning of the characters.

- A is replaced by the employee's job grade (i.e. 1-9).

- BCDE is replaced by the employee's pension status (i.e. PEN for On Pension or NPEN for Not On Pension). It is worth noting that if the pension status is PEN you would have to make sure that a leading space is added to keep same displayed indicators positioning.

- If the pension status of an employee payroll account that is currently displayed on the LCD screen is PEN (i.e. On Pension), then the backlight must change to the colour: green, pension status of an employee payroll account that is currently displayed on the LCD screen is NPEN (i.e. Not On Pension), then the backlight must change to the colour; red.

- The £££££.££ is replaced by the employee's salary. Salary should display two numbers after decimal point, even if 00. Note that salary should not accept values over £99999.99

- The 1234567 is replaced by the employee's ID number which is unique to each employee. The ID number is a 7 digit number in the range 0-9.

- The FGHIJKLMNOPQRSTUV is replaced by the employee's job title. A job title is a string between 3-17 characters long from the range A-Z and 0-9, _ and . (period) (e.g. Software_Engineer, HR_Manager, Guard, etc.).

- Program the UP and DOWN buttons to move up or down through the accounts list in **numerical order**. On reaching the first or last account remove the ^ or v, to indicate there are no more accounts to display.

- Pressing and holding the SELECT button for longer than one second clears the screen, sets the backlight to `purple` and simply displays your student ID number and nothing else. Releasing it returns the display to normal. During this time, your program must continue to read lines from the serial monitor and process them.

## 2.3 The protocol in more detail

- Synchronisation phase:
  - After your Arduino has booted it must set the backlight to `yellow` and repeatedly send the character `R` with no following "`Newline`" or "`Carriage return`" control characters to the Serial interface at a frequency of once every 2 seconds.
  - After sending, the Arduino must monitor for an incoming String and when an `BEGIN` is received synchronisation completes and the main program loop can start up.
  - No "`Newline`" or "`Carriage return`" characters should be sent. If a "`Newline`" or a "`Carriage return`" character is received it must be treated as an error.
- After synchronisation happens your program must send the string `BASIC` followed by a "`Newline`", then set the backlight to `white` and start the main phase.
- Main phase:
  - The host sends messages to the Arduino over the serial interface. The messages will start with a string that denotes the type of requested operation, this is followed by other information, these operation characters are as follows:
    - **ADD** — this indicates adding a new payroll account. It will be followed by '-' then a numbers made up of exactly 7 digits that represent the employee id number, then '-' and then followed by a number for the employee's job grade `(i.e. 1-9)`. This is followed by another '-' which is followed by a text for the job title. The payroll account number digits are from the 0-9 range (e.g. `1234567`, `7654321`, `0817263`, etc.), the job title is a string between 3-17 characters long from the range A-Z and 0-9, _ and . (period) (e.g. `Software_Engineer`, `HR_Manager`, `Guard`, etc.). Any extra characters after the payroll account number (i.e. employee ID number) and job title will be treated as an error. If the host sends an "`ADD`" message for an existing payroll account number, this will give an error message should indicate can't create an account with existing account number.

* **PST** — this sets the employee's pension status (i.e. `PEN` or `NPEN`). It will be followed by a '-' that is followed by either an `PEN` or `NPEN` text. Any pension status values received for accounts that have not been added must be treated as an error. It is worth noting that the default pension status of an account when added is `PEN`. If an `PST` message is received which does not change the pension status (i.e. `PST-1234567-PEN` when status is `PEN` or `PST-1234567-NPEN` when status is `NPEN`), it should raise an error to indicate that account pension status not changed. If a `PST` message is received for an account with a salary of `£0.00` this should raise an error to inform that this employee's salary is still `£0.00`.
* **GRD** — this indicates changing the employee's job grade. It will be followed by '-' then a number of exactly 7 digits form the 0-9 range, these are for the employee's id number, then '-' and then followed by a number for the account's new job grade `(i.e. between 1-9)`. If the host sends a "GRD" message with the same job grade as the account's existing job grade or lower, an error message should indicate can't be modified because of the same job grade or lower.
* **SAL** — this indicates changing the employee's salary. It will be followed by '-' then a number made up of exactly 7 digits form the 0-9 range, these are for the employee's id number, then '-' and then followed by a number that can have a decimal point, this number should range anywhere between `£0.00` and `£99999.99`. If the numbers after the decimal point are more than 2 then they should be rounded to 2. if the number sent is `100000.00` or bigger, or less that `£0.00` then it must be treated as an error.
* **CJT** — this indicates changing a job title. It will be followed by '-' then the 7 digit payroll account number (i.e. employee's id number), then '-' and then followed by a new job title string between 3-17 characters long from the range A-Z and 0-9, _ and . (period) (e.g. `Software_Engineer`, `HR_Manager`, `Guard`, etc.). If the string is lass than 3 or linger than 17 characters long it will be treated as an error.
* **DEL** — this indicates deleting an employee's payroll account. It will be followed by '-' then by the 7 digit payroll account number (i.e. employee's id number). If the payroll account number does not exist, then it should be treated as an error. Otherwise, the payroll account with the matching number, should be deleted with all its data.
* 
  - Any lines not conforming to the protocol must output an error message of the form "ERROR: followed by the error message" should be written to the Serial Monitor. This will aid debugging. This should have no other effect on the system.
  - Lines for any operation `(i.e. ADD, PST, GRD, SAL, CJT or DEL)` with no `1234567` (i.e. employee payroll account number) are invalid and treated as an error.

        · Lines for `ADD` missing the job grade or/and job title fields are invalid and treated as an error, with an error message.
        · Lines for `PST` missing the pension status field are invalid and treated as an error, with an error message.
        · Lines for `GRD` missing the job grade are invalid and treated as an error, with an error message.
        · Lines for `SAL` missing the salary number field are invalid and treated as an error, with an error message.
        · Lines for `CJT` missing the job title string are invalid and treated as an error, with an error message.
        · Lines with the wrong number and/or wrong position of field separators '-' are invalid and treated as an error.
    * In addition, you may send any text from your program beginning with **DEBUG:** to the Serial Monitor; it will be ignored by the assessment testing program but will help your debugging.
    * You can send a message from your program with **DONE!** every time the program finishes executing a serial command.
    * There must be no space characters in the serial command messages.
    * Your Arduino program should not output any serial messages that don't start with **ERROR:** or **DEBUG:** or the message **DONE!** This will help the testing program understand what you are sending.
- Unless you implement the EEPROM extension (see below), then on initialisation no payroll accounts information should be assumed added and consequently after synchronisation all received messages not starting with `ADD` must be treated as an error until a suitable first `ADD` message arrives.
- Given that the payroll account numbers are made of $7$ digits, this means that theoretically you can have $10^7$ number of possible payroll account numbers. Since the Arduino's SRAM memory is limited, please choose a reasonable enforced limit on the number of payroll accounts that can be added (i.e. once reached you can't add more accounts until you delete some of the existing ones). The main goal is that your program doesn't run out of memory, yet you maximise the number of accounts and their information that you can add. Being able to add the most amount of accounts possible without running out of memory will be looked at advantageously.

## 2.4 Protocol Examples

Example uses of the protocol: The example below adds payroll accounts and manipulates them through different serial commands.

Example messages with teal comments are valid and get processed. Whereas messages with gold comments are not valid and provoke an error response from your program. Assume the example messages are run in this order (i.e. if an account is added in one command then assume it exists for consecutive commands).

|  |  |
|---|---|
| **ADD-6359870-6-Software_Engineer** | ← Adds a payroll account with the account number “6359870”, grade 6 and job title “Software_Engineer” |
| **ADD-3356872-4-Guard** | ← Adds a payroll account with the account number “3356872”, grade 4 and job title “Guard” |
| **ADD-1425466-8-HR_Manager** | ← Adds a payroll account with the account number “1425466”, grade 8 and job title “HR_Manager” |
| **SAL-6359870-45000** | ← Sets the salary of the account with the account number “6359870” to £45000. |
| **PST-6359870-PEN** | ← Sets the pension status of the account with the account number “6359870” to PEN (i.e. On Pension). |
| **PST-1425466-PEN** | ← Produces an error as trying to change pension status of an employee account when the salary is still set to `£0.00`. |
| **GRD-1425466-50** | ← Produces an error as trying to change the job grade of this payroll account “1425466” to 50 which is not a valid grade |

| | |
|---|---|
| **PST-6359870-PEN** | ← Produces an error as trying to change pension status of a payroll account with employee id number “1425466” but its status is already `PEN` (i.e. On Pension) |
| **PST-6359870-1** | ← Produces an error as 1 is not a valid option for pension status |
| **GRD-3499234-6** | ← Produces an error as an account with the account number “3499234” does not exist |
| **SAL-1425466-71250** | ← Changes salary of payroll account with employee id number “1425466” to “£71250” |
| **PST-1425466-PEN** | ← Sets the pension status of payroll account with the account number “1425466” to PEN (i.e. On Pension). |
| **CJT-3356872-Security-Guard** | ← Produces an error as ”-” is not a valid character to be included in job title. |
| **SAL-3356872-31300** | ← Changes the salary of payroll account with employee id number ”3356872” to “£31300”. |
| **ADD-23577172-5-Care_Taker** | ← Produces an error as adding a payroll account with an account number longer than 7 digits |

| | |
|---|---|
| **ADD-2357712-Care_Taker** | ← Produces an error adding a payroll account with no job grade. |
| **ADD-2357712-5-Care@Taker** | ← Produces an error as adding a payroll account with a job title containing characters not in the correct range. |
| **ADD-2357712-5-CT** | ← Produces an error as adding a payroll account with a job title less than 3 characters long. |
| **ADD-2357712-5-** | ← Produces an error because adding a payroll account with no job title. |
| **ADD-2357712-5-Care_Taker** | ← Adds a payroll account with the account number “2357712”, grade 5 and job title “Care_Taker” |
| **ADD-3356872-7-Swim_Instructor** | ← Produces an error as there is a payroll account with the same account number. |
| **ADD9872334-7-Swim_Instructor** | ← Produces an error as it does not represent a known message format |
| **ADD9872334-7- Swim_Instructor** | ← Produces an error as it does not represent a known message format because of the spaces |
| **DEL-3356872** | ← Deletes payroll account with this employee id ”3356872” |

A python program for running on a host and conforming to the specification is provided with this assignment. You are welcome to modify this for your testing purposes. A version of this program will be used to test your implementations, although the messages might change.

### 2.5 Requirements

- You must implement your code using a Finite State Machine (FSM). Include a picture of this machine, listing the states and the state transitions in your documentation.
- Make sure you comment your code appropriately. Ideally code should be mostly self-explanatory through sensible choice of variable and function names and use of macros. However, there will be times when things are less comprehensive, so be sure to highlight theses.
- In your documentation:
  - Write a description of your states, what is being waited for, and the actions taken during transition between states.
  - Write a description of the process that you have used for debugging in your code.
  - Write 200–300 words of reflection on your code. Include those things that don't work as well as you would like and how you would fix them.

### 2.6 Assessment

Achieving the above implementation is worth up to 70 marks, if properly documented (25 marks) and coded (45 marks).

## 3 Extension features

The remaining marks are distributed across the extension features. You may implement as many as you wish. Assume about 35% of each extension's marks will be based on any documentation.

The following extensions allow you to demonstrate more knowledge and in return are worth additional marks as indicated. If you have implemented any of these then the string BASIC sent after synchronisation must be replaced by a list of the extension names separated by commas and followed by a "Newline". For example:

**UDCHARS,FREERAM,EEPROM,SCROLL,HCI**

## 3.1 UDCHARS

This shows your ability to define your own character forms. [5 marks]

- Define some characters to replace the v and ^ with nicer looking arrows.
- Add to your documentation a section to discuss your UDCHARS implementation.

## 3.2 FREERAM

This shows your ability to read the free SRAM in an Arduino. [5 marks]

- Modify what happens when pressing the SELECT button. In addition to the ID number, display the amount of free SRAM.
- Add to your documentation a section to discuss your FREERAM implementation.

## 3.3 EEPROM

This shows your ability read/write from EEPROM. [5 marks]

- Store the payroll account numbers, grades, job titles, pension status and salary in the EEPROM so that they can survive power resets.
- Read from the EEPROM on start-up and find a mechanism for determining whether the values were written by you or simply left from before.
- in order to activate this function, a user should send a special message to the Arduino, the message structure as follows. The message starts with `ROM` followed by '-' followed by a string representing a new sync phrase, this should be between `1-10` characters long in the range of `a-z` and `A-Z`. Once the Arduino is rebooted it, if `EEPROM` function is active then it should use the new sync phrase for starting instead of `BEGIN`.
- sending the same command with the current sync phrase again to the Arduino should deactivate the `EEPROM` function. If a wrong sync phrase is sent then this should be treated as an error.
- Add to your documentation a section to discuss your EEPROM implementation. Also indicate if you foresee any problems with this approach regarding RAM size and EEPROM size and how these problems can be avoided.

## 3.4 SCROLL

This shows your ability to implement a more complicated FSM that can take time into account as well as listening to the Serial Monitor and button presses. [5 marks]

- If a job title text is too long (i.e. longer than 7 digits available on screen) then add code to scroll it left at 2 digits per second, returning to the start when the full balance has been displayed.
- All other functionalities should remain implemented as described.
- Add to your documentation a section to discuss your SCROLL implementation.

## 3.5 HCI

This shows your ability to select particular values from an array of values, and process these values. [10 marks]

- When the user presses and releases the RIGHT button, the display must only display payroll accounts that have status `PEN` (i.e. `On Pension`). If none match this criterion, then display "`NO MATCHING ACC`".
  - While in the first right menu, if a user presses and releases the RIGHT button again, then the system enters the second right menu (monthly salary menu) and the payroll accounts are displayed exactly the same but with salary showing as monthly salaries (i.e. salary divided by 12). Display `0` for accounts with `0` salary.
  - While in the second right menu, if a user presses and releases the RIGHT button again, then the system enters the third right menu (tax menu), where employee's monthly tax is displayed instead of salary. Display `0` for accounts with `0` salary.

    Tax is calculated on salary after deducting pension contribution which is 6.1%, the tax is then calculated according to the table 3.5 below.

| Band | Taxable income | Tax rate |
|---|---|---|
| **Personal Allowance** | Up to £12,570 | 0% |
| **Basic rate** | £12,571 to £50,270 | 20% |
| **Higher rate** | £50,271 to £125,140 | 40% |
| **Additional rate** | over £125,140 | 45% |

Table 1: UK Tax Bands

  - While in the third right menu, if there are accounts that are on Pension, if a user presses and releases the RIGHT button again, then the system enters the fourth right menu, where employee's monthly pension contribution is displayed instead of salary (i.e. 6.1% of monthly salary). If there are no payroll accounts that have status `PEN` (i.e. `On Pension`). If none match this criterion, then display "`NO MATCHING ACC`".
  - while in any of the right menus, if a user presses and releases the LEFT button, then this changes the display to the menu before, until the main default menu of the program is reached.

- While displaying the main default menu, if a user presses and releases the LEFT button, the display must only display payroll accounts that have status `NPEN` (i.e. `Not On Pension`). If none match this criterion, then display "`NO MATCHING ACCOUNTS`".
- While displaying any of the side menus, the only buttons that should change the display are LEFT and RIGHT, as well as SELECT which should still do its function in any menu.
- The v and ^ arrows should only appear based on the subset of accounts to display.
- Add to your documentation a section to discuss your HCI implementation.

## 4 Hints

- SRAM is limited. You want to keep use of SRAM to only what really needs the SRAM. Make use of the FLASH memory for fixed strings.
- Be careful using the String class—this tends to fragment the memory.
- On writing to EEPROM remember to use the correct calls to avoid writing when you don't need to.
- In your design of the FSM, consider what events your code will be waiting for. Think about button presses, button releases, something from the serial, timeouts.
- You can choose to interact with your program simply through the Serial Monitor, but it may help you to write some python host code to automate this. An example of this code is provided along with the coursework brief.

## 5 Implementation

1. You must use (at least two) Finite State Machines to implement this specification and these must be described in your final deliverable.
2. You must implement the protocols exactly as described. If you feel anything is not clear, then please ask. We may use automated testing of your code.

## 6 Deliverables

This is an individual coursework. Similarity-detection software will be used on both the documentation and the source code as an initial check to see where collusion or copying have taken place. Code may be checked against other submissions this year, previous years and other sources as appropriate.

Your submission needs to include the following:

- Source code of your implementation: a single .ino file submitted to LEARN.
- A written report—submitted as a PDF along your code submission to LEARN.

### 6.1 Source code

Submit this as the actual .ino file, not copy-pasted into the report.

### 6.2 Written report

A template for the report is available on the Learn page in markdown, Word and LaTeX, your choice. But remember the final submission must be as a PDF. You must use the provided template—see LEARN.

Please note that other file formats will not be accepted, and simply renaming the report's extension to a PDF extension will be penalised and might be treated as a non-submission.

## 7 Further information

- You may only use the following libraries (as identified by their header files). You might not need to use all of these.
  - `Wire.h`

  - `Adafruit\_RGBLCDShield.h`
  - `utility/Adafruit\_MCP23017.h`
  - `EEPROM.h`
  - `avr/eeprom.h`
  - `TimeLib.h`
  - `TimerOne.h`
  - `MemoryFree.h`

- Your code must consist of a single .ino file, which needs to compile using the Arduino IDE. If the code fails to compile in this environment, then you risk failing the module.
- Your code must work with the Arduino Uno and LCD shield that we issued to you. In particular, if you develop your code on a different type of Arduino or with a different type of shield, you risk failing this module.
- We will use various types of software for code similarity detection to compare each submission to all others, to submissions from previous years, and to code from other sources. Please be reminded that this is an individual assessment, which means that group work is not permitted.

## 8 Grading scheme

Most marks will fall into a scale between 40 (scraped pass) and 80 (impressive). These are explained in the grading descriptors below.

Significantly fewer marks will be awarded if you do not follow the compulsory instructions, for example, not using the template, using other libraries that those specified above, using standard C++ code not designed for Arduino.

As a rough guide the following sections indicate what the expected mark ranges.

### 8.1 Code

Code covers the code itself and includes readability, sensible function names, use of functions and language features as well as whether the functionality is there.

- A (80+) - The project significantly exceeds the minimal specification in the number of features and in the technical challenge behind these features. All features have been realised at an excellent level. The project implementation carried out at a high-quality level.
- B (70) - The project significantly exceeds the minimal specification in the number of features or in the technical challenge behind these features. The project implementation has been executed well with some aspects at an excellent level.
- C (60) — The minimal feature set and some non-trivial extra features have been realised at an adequate level. The project implementation and testing is at an adequate level. The interface is easy to understand after some explanation.
- D (50) — The minimal feature set, and some minor extra features have been realised a mostly adequate level. Enough has been done the overall achievement is mediocre.
- E (40) — The minimal feature set has been realised at a somewhat adequate level. The overall implementation and testing of the project is rudimentary.
- F (¡40) - Not even the minimum feature set working and nothing to make up for this. The higher end of this range would be used to reward partial attempts with some merit.

## 8.2 Documentation

Documentation will cover pictures and text in the report and comments in the code.

- A (80+) The documentation is beautiful and describes all aspects of the implementation well.
- B(70) The documentation is very good, including all details, but lacks in presentation and understanding.
- C(60) The documentation includes most of what is necessary, but misses some vital meaning.
- D(50) The documentation describes some aspects well, but these aspects only form some of what is required.
- E(40) The documentation is barely adequate.
- F(¡40) Something worse. No documentation will lead to a mark of zero.