Embedded Systems
Electronics, Power and Automation

Information technology B final assignments

Jos Onokiewicz

5 November 2016
version 2.0a

Document history

Date	Version	Who	Changes
06-11-2015	1.0	Jos Onokiewicz	Final exercises added. All are related to operating a device.
05-11-2016	2.0	Jos Onokiewicz	Updated guidelines for product document. Added UML statechart and architecture CVM and PlantUML scripts.
10-11-2019	2.0a	van den Hooven	Changed the SVN-Links to GitHub

HAN University of Applied Sciences

Embedded Systems
Electronics, Power and Automation

CONTENTS

PREFACE

This document contains the assignments for INFTb. In the final assigment the program knowledge and skills gained in Block 1 must be applied. The main objective is the realisation of a full text console oriented C program for the operation of an device. It must contain a state machine in a layered architecture that addresses a hardware abstraction layer (HAL) through a number of functions. In addition to the program, a product report must be created.

Arnhem, November 2016
Jos Onokiewicz

1. PREPARATIONS AND EXECUTION OF FINAL ASSIGNMENTS

This chapter examines the set requirements for the approach and development of the final assignment. The relationship with the business operation is explicitly present. Regard the lecturer as a customer.

1.1 Availability of programs

The following programs are used:

- QtCreator by Digia
- PlantUML for state charts: http://plantuml.com/
- Lucidchart for flow charts: https://www.lucidchart.com. Make use of the "Start free account". Lucidchart is an online program for making flow charts. See the sheets on Scholar to use the correct graphical symbols. Flow charts (if necessary) should only be used for describing actions in the statemachine but not he statemachine itself (use plantUML for the state chart).

!	Always make backups of your programs and documents. Consider using, for example, DropBox.

1.2 Requirements for completing final exercise on software

The following requirements apply to completing the final assignment:

- **Proper working program**
  - the program can be started (executable file is available)
  - the program does not crash during execution
  - demonstrably tested (run through all states for this)
  - functionally correct (results in logbook)
  - the program complies with the set specifications
- **User friendliness**
  - informative texts for the user
  - proper screen/display layout
  - tests for incorrect input
  - restore possibilities with wrong input
- **Simplicity, reliability, suitability**
  - lack of unnecessary frills

- lack of non-specific functionality

- **Clear division of software**

  - for the final exercise, use one directory with a suitable name
  - use a modular approach (header files and implementation files)

- **Clear and consistent layout of program code**

  - use the guidelines in “C style and programming guidelines” by Jos Onokiewicz
  - apply a code formatter in QtCreator

- **Good choice of names in C code**

  - suggestive informative names for constants and variables
  - suggestive informative names for functions

- **Correct use of variables and parameters in C code**

  - avoid unnecessary global variables
  - avoid unnecessary parameters
  - proper use of pointers
  - use const

2. INTRODUCTION EXERCISE: STATE MACHINE “COKE VENDING MACHINE”

This chapter contains the introduction exercise to the final exercise. This introduction exercise allows us to become familiar with the C implementation of a state machine for a “Coke Vending Machine” (CVM).

By expanding it into a number of short exercises, you should acquire a good comprehension of how to make the state machine for the final assignment.

2.1 Getting familiar with the state machine

The code of the CVM application is available on OnderwijsOnline in a QtCreator project. This code (CVM1, CVM2 and CVM3) must be examined in detail in order to understand how it works. For each state, work out under which conditions there is progress to another state.

Code repository link: CVM1+CVM2+CVM3

Code repository for the UML diagrams link: CVM System Development

Take note of the use of the applied functions: objective, parameters and naming conventions.

2.2 CVM state machine expansion exercises

The CVM system contains several components (subsystems):

- **CVM** Coke Vending Machine (core code, statemachine code, FSM)
- **CNA** CoiN Acceptor
- **COD** COke Dispenser
- **CHD** CHange Dispenser

The above mentioned abbreviations are used as a necessary prefix for the C function names in the related code for the subsystems.

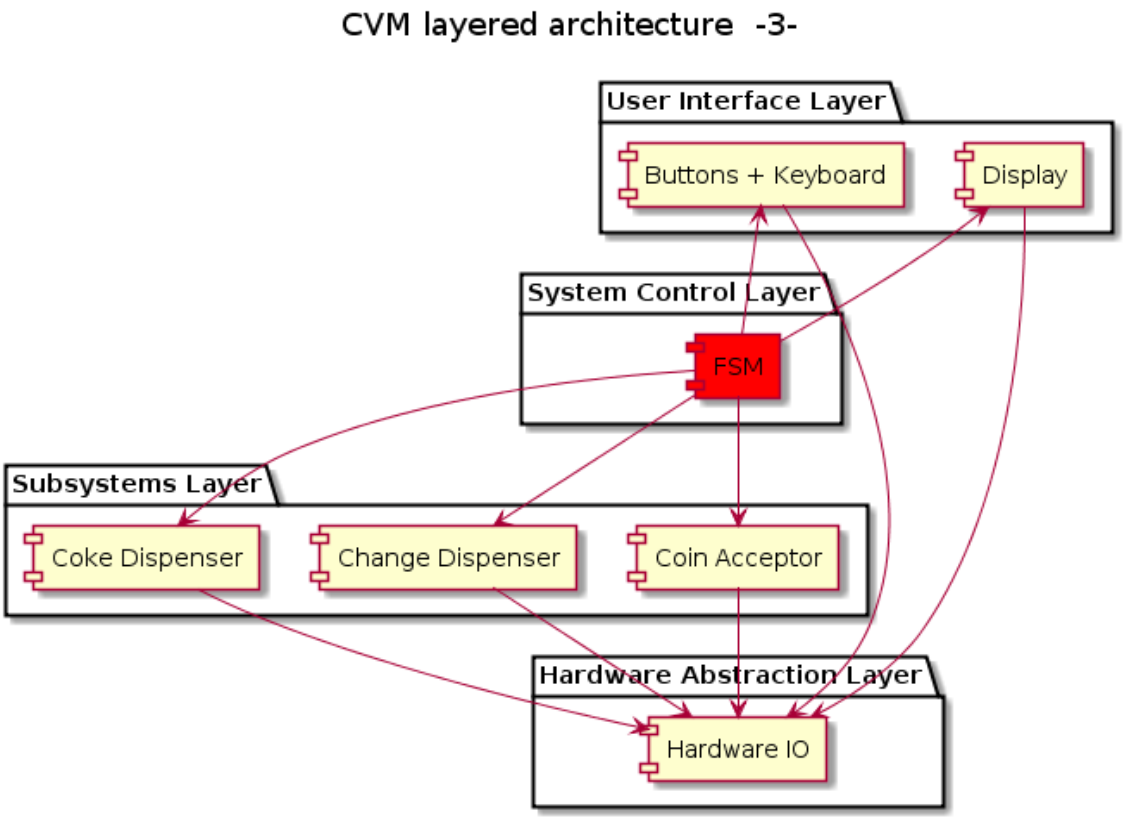

Figure 1 CVM architecture

The CVM architecture diagram shows all **subsystems** and their **dependencies** (arrows).

!	**The direction of the dependency arrows are related to the inclusion of header files.** They are not related to the flow direction of data.

For example the code of the FSM includes the header file of the Coke Dispenser module, but the module Coke Dispenser does not include the header file of the module FSM. The FSM code depends on (uses) the code of the Coke Dispenser but not reverse!

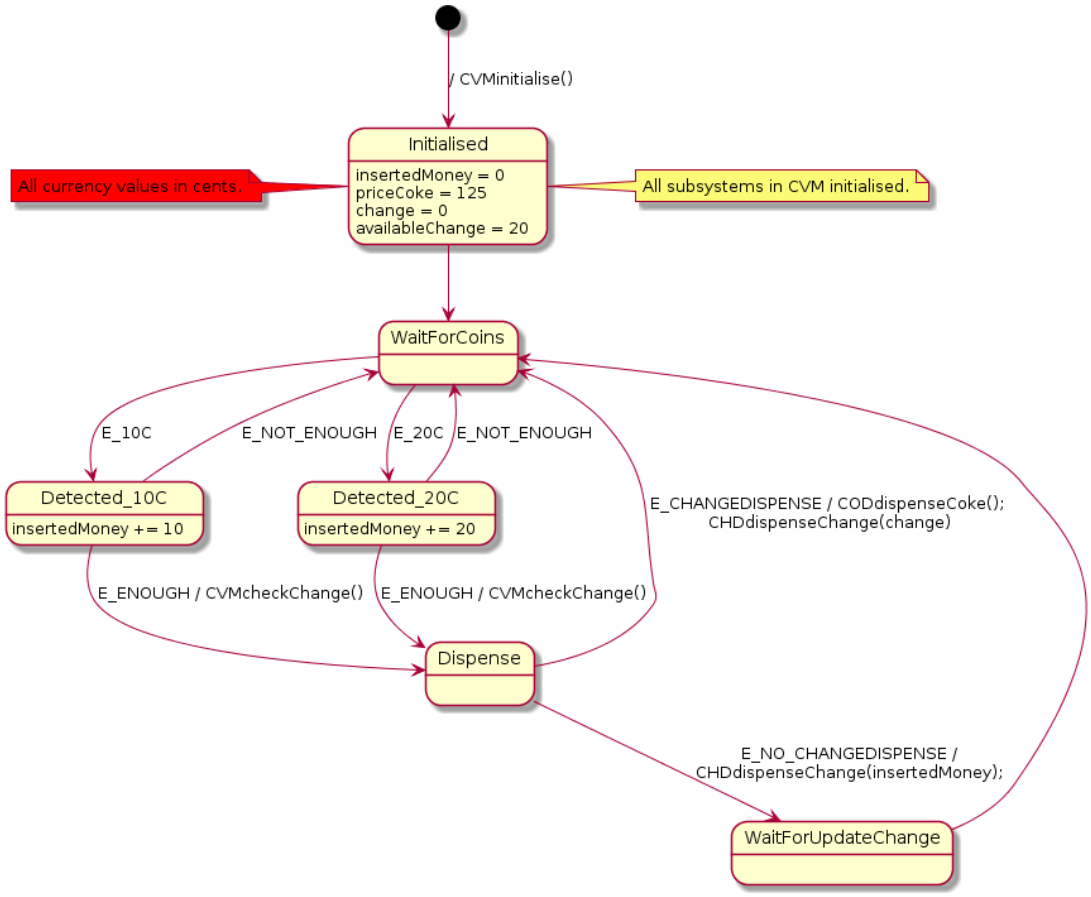

Figure 2 CVM statemachine

Supplement the state machine chart in several steps and code these updated state machines according to the drawn charts with the following functionality:

a. Expand the input of coins with 50 Euro cent and 1 Euro coins.
b. If a coin is not recognised, a suitable message must appear on the display. This coin must be returned to the customer.
c. If, while inserting coins, a customer decides that he or she does not want a Cola after all, he or she must be able to push a button (a key) so that all the money inserted is returned.
d. If the cans of Cola are all gone, a suitable message must appear on the display and the customer will not be able to insert any more money.
e. Test the entire state machine, going through at least all states by entering the appropriate input and then indicating whether the behaviour (functionality) is correct.

For completion, the statechart must be shown and the working of the definitive version must be demonstrated to the lecturer.

!	Use the guidelines in “C style and programming guidelines” by Jos Onokiewicz.

2.3 Scheduling

This introduction exercise must be completed in the first two weeks of Block 2. Also read the final assignment in Chapter 3, so that you get an idea of which exercises there are. For details, see the INFTab course manual.

!	The exercises cannot be done at school during the scheduled time. It will be necessary to work on them outside of these times.

3. FINAL EXERCISES

This chapter contains an overview of the final exercises (cases), one of which must be done. This must be done in consultation with the lecturer.

3.1 Control software for a device without hardware

The final exercise is the realisation of software for controlling a device. But we do not have any hardware yet. By using a HAL, we can nevertheless build the software and (partially) test it. See the details of the preparatory CVM exercise in the previous chapter.

3.2 Final assignment

A choice must be made from the following cases. In order to make the realisation as feasible as possible, the functions to be realised have been restricted. The descriptions need not be complete. Submit your own necessary supplements and/or changes. Consult with the lecturer in the role of the customer in order to get a more complete list of the requirements. The requirements must be reflected in the product document (see Chapter 4).

!	In week 3, the draft requirements must already be in the draft of the product document so that the lecturer can review them.

Design a state machine, draw it with PlantUML and consult with the lecturer in the role of technical specialist whether it appears fitting for the solution of the operation of the instrument.

Important: first draw the full state chart, build the state machine step by step. Every step must result in compilable and testable code.

On the basis of the input (buttons) and the output, determine how it must be given in text form. In a number of cases, this has already been given in outline. Suitable functions must be created in the HAL. Therefore, the HAL code from the CVM project cannot simply be used as it is! Always use the display functions to show messages to the user (bare printf calls in the FSM code are not allowed).

Case 1 Coffee machine control software

Assume the basic functions of the coffee machine that we saw at school. Use the introduction exercise: the CVM project as an example of a vending machine.

Requirements:

- The input must consist of coins.
- The user must be able to select from three types of coffee.
- If one of the types of coffee is finished, a suitable message must be displayed.
- If there is no more change, a suitable message must be displayed.
- The administrator can insert coffee and change into the machine.
- Supplement the software with functions in a separate module that adds an amount paid to a total amount and saves this in a text file. This file also contains the amount of coffee still available and the change that is available. These amounts must be saved in one file. When restarting the coffee vending machine, these amounts must be read.

Describe for yourself the missing functional requirements. Always begin with the requirement for the main function that a coffee vending machine must execute. Number these requirements. Add to this a new numbered list with technical requirements.

Case 2 3D LED cube control software

A 3D LED cube consists of a three-dimensional structure of LEDs with which we can show light patterns by switching specific LEDs on or off.

Requirements:

- The size of the LED cube is 4x4x4.
- Each of the LEDs must be able to display 2 colours: red and green.
- The LED cube must begin with a standard pattern (you choose which one).
- The LED cube software must give the light patterns in text format as follows:

```
RGGR  G..G  G..G  RGGR
G..G  ....  ....  G..G
G..G  ....  ....  G..G
RGGR  G..G  G..G  RGGR
```

G LED is on: green
R LED is on: red
. LED is off

- The user must be able to activate the following items with several buttons:
  - switch on all LEDs in one colour (testing function), both red and green.
  - reset, restart the LED cube.
  - supplement the LED cube with at least two patterns in addition to the starting pattern from which the user can choose.
  - write and read the patterns of the LED cube to and from a file.

- move the pattern to the left, right, up and down in one step. The maximum number of steps in one direction is equal to the size of the side of the cube. The software must remember the LED colours that can no longer be given because of the shift. If a shift back occurs, the old pattern must be able to be given. The user must be able to indicate with a button when the results of a following step must be given.
- circular shift (like the previous function), but now the LEDs that disappeared appear on the other side of the cube.

Implementation tip: use a three-dimensional char array in C (see Chapter 5).

Describe for yourself the functional requirements that may still be missing. Always begin the description with the requirement for the main function that must be executed. Number these requirements. Add to this a numbered list with technical requirements.

Case 3 2D LED gadget control software

A 2D LED gadget consists of a two-dimensional structure of LEDs with which we can show light patterns by switching specific LEDs on or off. The objective is to be able to play a retro 2D game. Think of games such as Pong, Snake and Maze. One of these games or a game of your choice must be implemented.

Requirements:
- The 2D LED gadget must be at least 10x10.
- Each of the LEDs must be able to display 2 colours: red and green.
- The player must be able to operate the game with buttons.
- At any time, the gamer must be able to reset the game in order to start over again.
- The gamer can switch all LEDs on to test whether they all work or not.
- The LED gadget software must give the light patterns in text format as follows (for example, for a 10x10 gadget):

```
RGGGGGGGGR
G..G......
G..G.....G
RGGGGGGGGG
RGGGGGGGGR
G..G......
G..G.....G
RGGGGGGGGG
```

G LED is on: green
R LED is on: red
. LED is off

Implementation tip: use a two-dimensional char array in C (see Chapter 5).

Case 4 ATM machine control software

Assume a countertop ATM machine that can be found in many shops. For the structure of the display and keyboard, see the following source:

http://www.flexipin.nl/pin-huren/toonbank-pinautomaat-huren/

Describe the functional requirements. Always begin with the requirement for the main function that a countertop ATM machine must execute. Number these requirements. Add to this a new numbered list with technical requirements.

Supplement the software with functions in a separate module that adds a requested amount to the account number to which the payment terminal belongs. This amount must be saved in one file. Each time that an amount is withdrawn, this amount must be raised.

Case 5 Gym lockers control software

A fitness company uses lockers that can be hired by a user at no cost by entering an individual code. To open it, this code must be used again. The lockers are used to store items such as clothing, soap, watches and cell phones. There is a console between the lockers with buttons and a display.

Requirements:

- There are 10 lockers, each with its own number (1 through 10).
- There is a keyboard and display between the lockers available to operate the lockers. The display provides informative messages that are to assist the user in operating the lockers.
- An operating unit asks the user if he/she wants to open his/her locker or to open a new locker.
- If a locker is still available, a user can get one by submitting an individual code.
- A user must be able to open his/her locker if the locker number entered agrees with the code entered earlier.
- After opening, a user must always be able to hire his/her locker again. The code then remains the same.
- If someone forgets his/her code or locker number, it is possible for the administrator to open any locker with a special code and a special key. If the locker turns out to belong to someone else, the locker is closed again, retaining the code that was entered. If the locker belongs to the person, it remains open and can be used again by a person who wishes to use a locker.
- After being closed and after a certain time, the door of a locker must automatically be locked (use pushing a button to indicate that time has passed).

Describe the functional requirements that are missing. Always begin with the requirement for the main function that a locker must execute. Number these requirements. Add to this a numbered list with technical requirements.

Case 6 Personal weight scale control software

For a personal weight scale, operating software must be created that records your height and your weight. At the same time, the Quetelet index (BMI) must be calculated with each weighing. The Quetelet index, named after a Belgian nutritionist, can be calculated for a person by dividing his/her weight by the square of the height in meters: **Q = g / l ²**. If this value (for adults) is between 18.5 and 25, it is a healthy weight, under 18.5 is a sign of being underweight, and between 25 and 30 of being slightly overweight, and above 30 of being seriously overweight. For example, if you are 1.94 meters tall and weigh 83 kilograms, your Quetelet index is 83/1.942 ≈ 22.1. This means that you are at a healthy weight.

Requirements:

- The personal weight scale must be able to store the data of a maximum of 4 people. You must be able to indicate with a button which number you are (1 through 4). If the number submitted (= person) has not yet been assigned a height, a suitable message must appear on the display.
- For each person, the height must be able to be submitted with a maximum of two numbers after the decimal and must be between 1.00 and 2.50. If the height submitted is outside of this range, a suitable message must appear on the display, as well as another request for the height.
- The user must set the weight scale to the right person with a button prior to stepping on the scale for the weighing (1 through 4).
- The weight must be between 40 and 160 kilograms. If the weighing is outside of this range, a suitable message must appear on the display. The weight must be accurate within 100 grams.
- The Quetelet index must be given with a maximum of one number behind the decimal point, supplemented with the appropriate statement “ underweight”, healthy weight”, “slightly overweight”, or “seriously overweight”.
- The information about the people must be saved in a file.

Describe for yourself the functional requirements that may still be missing. Always begin with the requirement for the main function that a personal weight scale must execute. Number these requirements. Add to this a numbered list with technical requirements.

Implementation tip: create a separate module for calculating the Quetelet index (.h and .c file).

Case 7 <Own selection of a device> operating software

It is possible to select your own device provided it is sufficiently “simple” and includes a keyboard and display and/or LEDs. Look on internet for photographs and manuals. Always do this in consultation with the lecturer.

Describe for yourself the missing functional requirements. Always begin with the requirement for the main function that the selected device must execute for the user. Number these requirements. Add to this a new numbered list with technical requirements.

---

4. REQUIREMENTS PRODUCT DOCUMENT

The product report must provide information for the customer and developers who, in the following step, will adjust the program to new requirements, supplement or improve it and/or remove errors (bugs). In practice, much time is devoted to further development of programs, usually by people other than those who created the first version! This means that the quality of product documents must be such that they are complete and clear. If they are not, the development costs for subsequent versions of the program could run unnecessarily high.

4.1 Content of product report

The product report should contain the following parts (if a chapter becomes too large, introduce a division in section with informative names):

Title page
Summary
Preface
Table of contents

1 Introduction
2 Definition phase
3 Design phase
4 Realisation and test phase
5 Test plan
6 Final results and recommendations

Appendix 1 Program listings
Appendix 2 User manual

- **The title page** contains the project title (name of the product developed) and subtitle (name of the concerned unit of INFTb study), author with student number, class, name of supervising lecturer, date and possibly an appropriate illustration.

- **The summary** entails a maximum of three quarters of a page **and must be able to be read completely separate from the report** (the reader is not yet familiar with the project). This summary must cover the entire report. There must be a short description of the subject: formulation of a problem, what has been solved for the customer, the customer, the most important requirements and what results were achieved and not achieved and, finally, any advice for improving and/or expanding the program. A short conclusion as to the results of the tests carried out must also be included. **There may not be any references to the rest of the document.**

- **The foreword** is meant for **personal comments**. It must relate what experience you have, the experience that was gained during the project (what did you learn in this field?), the problems that occurred or causes of delays and such. Finally, indicate what you will improve on next time. The foreword is written in the first person.

- **The table of contents** lists all numbered chapters and sections of the report with the number of the page at the end of the line.

1. **The introduction** gives a concise description of the topic of the report and how it is built up. By reading the introduction, the reader can familiarise him/herself with the contents of the report and determine if he/she wishes to continue to read. He/she can decide to read only certain chapters or certain sections in them. This means a brief discussion of the matters and problems that will be addressed and resolved and what will be discussed in the subsequent chapters. The latter means that somewhat more than just the titles of the chapters must be given.

2. **The definition phase** describes the background of the project, the main function and the requirements and desires that must be set for the product that is to be developed. The definition phase results in a functional specification and a technical specification.

   The functional specification concerns the exterior of the system, the what (describe the behaviour, what the system does for its users and/or other systems). Number the specifications. Define (outline) user interface (choice of keys and display) as it will be used for the actual instrument.

   The boundaries are also marked (what is done and what not) and the acceptance test is described (show the customer whether or not the system complies with the specifications). For example, the technical specifications contain the choice of the programming language C.

3. **The design phase** describes the interior of the system (the processing side, the how). A rough indication is given of how the functional specification is converted into a technical solution.

   Give an outline of the layered software architecture containing subsystems in packages (user interface, state machine, system functions and the HAL). To that end, add an outline of the state machine created, the pseudo code/flow charts for the system functions and a description of the HAL.

   The names of the functions, events and states used in the diagrams must correspond with the code created.

4. **The realisation phase and tests** indicate in which modules and related files is being worked. Create a user manual. Appendices are used for this. Extensive test cases are also generated in the realisation phase for in-depth testing of the product. The test plan is discussed in Chapter 5.

5. **The test plan** shows the test cases of the different C functions used and the acceptance test with its results. For the reproduction of the test, use the table format as shown in Figure 9.3.

6. **The final results and recommendations** contain
   - an overview of what has and what has not been realised of the exercise (in relation to the set requirements.
   - a summarising conclusion about the test performed
   - a proposal for any expansions and amendments
   - recommendations for improvements and/or supplements.

- **Appendix 1 Program listings** contains all listings belonging to the application, and therefore, also the header file(s). Order: main program, one or more combinations of a header file with the related implementation file.

- **Appendix 2 User manual** gives the user manual that is appropriate for the selected user interface. This is in English. Make sure that all states in the state machine that take care of the input and output for the user are covered. Please note that this should not be a technical description of the contents, since it is a user manual.

4.2 Product report's layout and style requirements

Requirements for layout and style:

- Number the pages, chapters and sections.
- Begin each chapter on a new page.
- Give the sections an informative name so that we can find a specific subject in the table of contents easily.
- Use a business style, and not the "I" or "we" form, with the exception of the foreword.
- Use correct spelling and sentence structure. Always use the spelling and grammar check of the word processing programme.
- Use summaries to prevent long sentences, so that it is easier for the reader to find the different topics.
- Put code in a non-proportional font: New Courier makes it much easier to read.

4.3 Reviews: have the product report read by others

Practice has proven that writing is difficult. This applies to people with lots of experience. Having others read your document (review it) quickly reveals the quality of your document.

4.4 Scheduling

You are expected to begin drawing up a draft of the product report in week 2 already. It will consist largely of empty chapters, with the exception of the chapter with the functional and technical requirements. Feedback on this by the lecturer is needed in week 3. Take into account that the lecturer's time is restricted. A draft with chapters that are more complete must be ready in week 5 already.

5. IMPLEMENTATION TIPS

This chapter gives a number of implementation tips.

5.1 conio.h for Windows

In Windows we can use the include file: conio.h. For example, with the function `_kbhit, we can wait for a key to pressed.` We no longer have to confirm this with an <Enter>. We also have to use the function `_getch`.

```
while (!_kbhit()){
  ;
}
printf("\nThe key pressed was (%c)\n", _getch());
```

5.2 Multi-dimensional arrays

In C we can make multi-dimensional arrays by adding pairs of brackets to the declaration of arrays.

Two-dimensional:

```
char leds2D[2][3];

leds2D[0][2] = 'R';
```

Three-dimensional:

```
char leds3D[4][4][4];

leds3D[0][2][1] = 'G';
```

6. PLANTUML SCRIPT CVM STATE MACHINE AND ARCHITECTURE

CVM-3 PlantUML scripts:

```
@startuml

'------------------------------------------------------ CVM - 3

[*] --> Initialised : / CVMinitialise()
Initialised : insertedMoney = 0
Initialised : priceCoke = 125
Initialised : change = 0
Initialised : availableChange = 20
Initialised --> WaitForCoins

WaitForCoins --> Detected_10C : E_10C
Detected_10C : insertedMoney += 10
Detected_10C --> WaitForCoins : E_NOT_ENOUGH
Detected_10C --> Dispense : E_ENOUGH / CVMcheckChange()

WaitForCoins --> Detected_20C : E_20C
Detected_20C : insertedMoney += 20
Detected_20C --> WaitForCoins : E_NOT_ENOUGH
Detected_20C --> Dispense :  E_ENOUGH / CVMcheckChange()

Dispense --> WaitForUpdateChange: E_NO_CHANGEDISPENSE /
     CHDdispenseChange(insertedMoney);
WaitForUpdateChange --> WaitForCoins

Dispense --> WaitForCoins :  E_CHANGEDISPENSE /
CODdispenseCoke(); \
     CHDdispenseChange(change)

note right of Initialised
 All subsystems in CVM initialised.
end note

note left of Initialised #red
 All currency values in cents.
end note

@enduml
```

```
@startuml

'------------------------------------------------------ CVM - 3

title CVM layered architecture\t-3-

package "User Interface Layer" {
  [Display]
  [Buttons + Keyboard]
}

package "System Control Layer" {
  [FSM] #red
}

package "Subsystems Layer" {
  [Coke Dispenser]
  [Coin Acceptor]
  [Change Dispenser]
}

package "Hardware Abstraction Layer" {
  [Hardware IO]
}

[FSM] -down-> [Coin Acceptor]
[FSM] -down-> [Coke Dispenser]
[FSM] -down-> [Change Dispenser]
[FSM] -up-> [Display]
[FSM] -up-> [Buttons + Keyboard]
[Coin Acceptor] --> [Hardware IO]
[Coke Dispenser] --> [Hardware IO]
[Change Dispenser] --> [Hardware IO]
[Display] --> [Hardware IO]
[Buttons + Keyboard] --> [Hardware IO]

@enduml
```