Assessment Specification

Software Assignment – up to 60% of final mark

MOD003212

Introduction to Programming

1. Table of Contents

2. Software Assignment (up to 60% of final mark)

The owners of Petrol Truly Unlimited Ltd are own a petrol station on the M25 and you have been tasked with the design, implement, test and document a Petrol Station Management app.

The implementation needs to be done in a windows console application written in C# and work on any University Windows PC.

1) App Synopsis

The festive season is around the corner and many families will be travelling to their holiday destinations. This period is a fantastic opportunity for Petrol Truly Unlimited Ltd to make up for the low profits during last December – where travellers were stranded on the M25 due to heavy snow. They therefore must ensure that the petrol station is working at full capacity and as efficiently as possible in order to maximise profits. The issue of efficient fuelling is paramount as drivers are normally very agitated during the summer holiday as they drive long distances with children in the car, which can be somewhat stressful to the parents. Last year, long delays in fuelling at our station resulted in many drivers turning around and fuelling at 'Petrol Nearly Unlimited Ltd', which is only 5 miles down the motorway. The management at Petrol Truly Unlimited Ltd fully recognise that a fuel attendant is pivotal to the success of this operation, which is why they have decided to incentivise them by adding a commission of 1% (0.01) of the total fuel money they sell to their already very generous hourly rate of £2.49. This works out as 8 hours shift * £2.49 + 1% of the total fuel they sell during that shift.

2) App Description

The app refers to a forecourt with *9* fuel pumps arranged over *3* lanes (three pumps in each lane). For the purpose of demonstrating the app to the customer, each *n* number of seconds (where *n* is random) a vehicle is created and awaits to be fuelled. The fuelling takes place by pressing on the pump number you wish to fuel the vehicle from (1-9). To add to realism of the demonstration and to account for drivers' agitation there is a finite period of time to fuel a waiting vehicle before it leaves the forecourt. That period is set to *Z* number of seconds (where *Z* is random). Should the vehicle not be fuelled during that period of time, it will leave the

forecourt. Similarly, a vehicle that was fuelled also leaves the forecourt, only this time the amount of fuel dispensed is recorded. A pump cannot service more than one vehicle at a time and the following **counters** have to be kept: (**1**) running total of the number of litres dispensed during the app's lifetime; (**2**) The amount of money these litres equate to; (**3**) The 1% commission; (**4**) the number of vehicles serviced, (**5**) the number of vehicles that left before they were fuelled and (**6**) keep a detailed list of each fuelling transaction – [Vehicle type, Number of litres and Pump number]. **You are free to choose a sensible cost per litre as well as use any numeric values (vehicle queue generation, vehicle fuel times, fuel capacity, etc.), as these don't change the program's logic**.

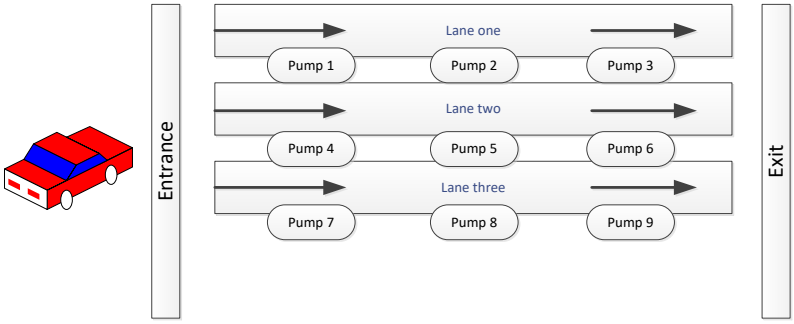

Figure 1: Forecourt Layout (Design only, not the final app UI)

3) Marking – Please read this section in detail

The assignment will be marked based on the following 5 criteria: (1) Functionality and Execution, (2) Program architecture, (3) Presentation and submission guidelines, (4) App User guide and (5) Testing.

- All 5 criteria are important! Please do not neglect any of them.

Time management is crucial – please make sure you leave enough time to implement and write all **five** criteria.

1. ***Functionality and execution*** (**24%**) – Bottom up approach! Start by implementing the set of functionality [**Basic**/**Intermediate**/**Advanced**] which you are most likely to achieve within the time available to you. When all functions have been fully implemented, attempt to implement the next set of functions. Again, only once the previous set was accomplished in FULL.

2. ***Program architecture and authorship*** (**24%**) – Use the programming guidelines throughout the development cycle of the app. Adhering to guidelines retrospectively is time consuming and often leads to bugs.
3. ***Submission and presentation*** (**2%**) – See details below and **please** adhere to the presentation and submission instructions.
4. ***App User guide*** (**5%**) – See details below.
5. ***Testing*** (**5%**) – The mark for this section take into account bugs that are **not** documented in your test plan. Please **test** your program, and **test it well -** reporting all the results!

4) Program Functionality and Execution

Before deciding which level of functionality to implement, please read carefully what is required in each level. You need to assess whether or not you are able to implement the level of functionality you chose within the time available to you. It is better to scale up the implementation should you have time. Please make sure that you leave enough time for carrying out the testing phase as well as writing the user guide.

Basic functionality: (for a maximum mark of 24% to 30%)

A. A new vehicle is created every 1.5 seconds. (No need to randomise this timer at this level)
B. When a vehicle is sent to a pump, the fuelling process will take 18 seconds. At the end of the 18 seconds the number of litres dispensed is recorded and the vehicle leaves the forecourt, freeing the pump for use.
C. The Pump is capable of dispensing 1.5 litres / second.
D. At this level, a newly created vehicle can wait until it is sent to a pump without a time limit.
E. Counters *1* to *4* and 6 have to be implemented (See App Description for detail, look for **counters** keyword).
F. At this level, there is only one type of fuel (Unleaded) and one type of vehicle that ever gets serviced by the petrol station.

Intermediate functionality: (for a maximum mark of 30% to 37%)

A. Same as above.
B. A new vehicle is created randomly every 1500 to 2200 milliseconds. And there can only be one car waiting to be serviced.
C. When a vehicle is sent to a pump, the fuelling time will be random and take between 17000 and 19000 milliseconds. At the end, the number of litres dispensed is recorded and the vehicle leaves the pump.
D. Newly created vehicles can only be fuelled within 1500 milliseconds of their creation. Failure to service them within this time frame will remove the vehicle from the forecourt.
E. There are 3 types of fuels (Unleaded, Diesel and LPG) and 3 types of vehicles (Car, Van and HGV). Both the type of vehicle and fuel of newly created vehicles will be selected randomly. At this level HGVs and Vans can run on any fuel type.
F. All pumps contain all types of fuel.
G. As above, all counters have to be shown, but with the addition of two extra counters totalling the number of litres dispensed for the other two fuel types (Diesel and LPG).

High level functionality: (for a maximum mark of 38% to 45%)

A. Same as above.
B. A new vehicle is created randomly every 1500 to 2200 milliseconds. And the queue of vehicles waiting to be serviced can reach 5. At this level, the time frame to service a vehicle is random between 1000 and 2000 milliseconds
C. Newly created vehicles will be created with a random amount of fuel already in their tank (which cannot be greater than a quarter of their total tank capacity). Fuelling time will be based on the above amount.
D. The random amount of fuel in a newly created vehicle will be proportionate to the vehicle size. Each vehicle type will have the following maximum amount in their tank: Cars - maximum 40 litres, Vans - maximum 80 litres and

HGV - maximum 150 litres. HGV can only run on Diesel. Vans on both Diesel and LPG and finally cars on all three types of fuel.

E. A queuing system has to be implemented in the forecourt as each lane has three pumps. As per figure 1, a vehicle which uses pump 1 blocks the way for new vehicles to get to pumps 2 or 3. The situation is similar for lanes 2 and 3.

F. Same as above.

Very High level functionality: (for a mark of 45% to 60%)

For a mark of 45% **PLUS** (up to the maximum of 60%) you will need to exhibit exceptional code elegance, noticeable additional functionality (e.g. Windows Forms) that enhances the app user experience as well as the realistic 'feel' of the demo, together with exemplary User guide and testing.

Students are encouraged to research and utilise built in methods in C# to aid with the Random and Timer aspects of this assignment (e.g. Random and Timer classes). This, however, is not necessary for all levels of implementation. Please note that you must acknowledge and reference all sources of information and code examples you use.

ALL of the above levels of functionalities have to implement a **Console** based Graphical User Interface (GUI) which displays the forecourt, denoting (in any way you want) newly created cars, free and busy pumps, counters, etc. The idea is to make the screen simple, clear and easy for the user to use (using simple characters - **See example below).** It is up to you whether you wish to create a Windows Form solution, thus adding another level of complexity to the app – it would have to be part of the Very High level of functionality (see above paragraphs for more info). Look at clearing the Console window and redrawing the UI every time a change happens to give the impression of a dynamic app.

```
                ********1 BUSY*************2 AVAIL***********3 Avail********
CAR→            ************************************************************
Counter A       ********4 BUSY*************5 AVAIL***********6 BUSY********
Counter B       ************************************************************
Counter C       ********7 BUSY*************8 AVAIL***********9 BUSY********

                Press the pump number to service the awaiting vehicle
```

5) Program Architecture and Authorship

The program architecture must be based on Object Oriented Design Methodology.

- There must be a minimum of 5 classes, complete with their member fields, Constructors and Methods.
- Where possible, fields are required to be private, only accessible via accessors.
- The structure needs to be easily maintainable and extendable to accommodate more pumps/lanes/petrol types/ etc.
- Clearly separate User Interface functionality objects (e.g. draw, click etc.) from the data manipulation objects (e.g. add/subtract fuel amounts, calculate costs etc.).
- Name classes, variables, methods etc. with meaningful, clear names using consistent, appropriate capitalisation.
- Be well documented with both normal // comments within functions and detailed /// comments on methods, classes and other definitions.
- Have a consistent, appropriate layout including good use of indentation, white-space and individual files for each class.

6) Submission & Presentation

The work must be submitted according to the following instructions. This ensures reduced risk for you and maximum convenience for the marking staff.

[Staff at associate colleges may vary these submission instructions, with the exception of item 2, and will mark according to the instructions given by them. Any variations will be posted to Canvas.]

1. ***Print a copy*** of the cover sheet and receipts from e-vision and a copy of the self-assessment form from Canvas. Do **not** print anything else (code, documentation etc.). There is a checklist as part of the self-assessment form, check this off as you go along.
2. ***Create two copies of a CD*/USB** containing:
    a. The app compiled as an executable (.exe) file
    b. The complete project folder of the source code for the program

c. The documentation in electronic form as a Word document (.doc, .docx) or Adobe (.pdf) file. Excel workbooks (.xls, .xlsx) are acceptable for the testing section.
d. Clearly Mark which functionality level you have implemented [**Basic** | **Intermediate** | **High** | **V.High**].

3. ***Check that the disks work***: it is your responsibility to ensure that the disks you submit are readable and that the files on them can be copied and opened. Staff will attempt to use the first disk and the second disk on two University-owned computers. If after this point the disks still cannot be read, a mark of zero (0) will be awarded.
4. Write your SID and the module code (MOD003212) using a permanent marker such as a Sharpie on the label side of both disks.
5. Fill in the self-assessment form, double check the checklist, and attach the Disk wallet sleeve containing your disks to the spaces indicated on the form. Make sure that no glue or tape is applied directly onto any part of the disks.
6. Fill in any relevant tick boxes on the cover sheet then secure the self-assessment form to it with a staple, top left. Please **do not use** A4 wallets, covers, binders, Arch folders.
7. Hand this in to the iCentre before the deadline. Keep a copy of your work and the receipt in a safe place until the end of the academic year.

7) App User Guide

This should be a clearly written, structured and presented document describing how to use the app. It should contain an exciting app synopsis that would entice the reader to buy the app along with a simple set of instructions of how to use the app. The file format should be either PDF or DOCX.

8) Testing

Full documentation of the tests carried out throughout the development of the software.

A proposed test plan layout can be seen in the module notes. You need not copy this exactly, but whatever plan you choose to use, it must be easily read, accurate,

and thorough i.e. all operations in your code are expected to have been exhaustively tested and recorded with enough detail to repeat the tests.

You may choose to compile this documentation using an Excel spreadsheet if you prefer.

9) Other Notes

- The intended audience for your documentation is to be considered that of your fellow students.
- You should include references to any sources you draw upon.
- A program that does not perform all the requirements will not necessarily be considered a failure – it can still earn marks for what it can do and how it does it. Clearly document any shortcomings.
- This assignment is ‘equivalent to’ a 5000 word essay.
- The deadline can be found on Canvas, please make sure you check it.

10) Marking Scheme	**Excellent – Grade 45 (+)**	**Good – Grade 35 (+)**	**Adequate – Grade 24 (+)**	**Insufficient–Grade 0 – 24**
Program Functionality and Execution 24%	All listed advance functions implemented and working. Simple and intuitive to use. Clear meaningful instructions and error messages. Robust and reliable, with all exceptions handled without crashing.	All listed intermediate functions implemented working. Usable program. Majority of errors handled without crashing. Clear messages and instructions to the user.	Most basic functionality working. User experience complex or inconsistent. Messages/instructions unclear or not enough of them. Frequent exceptions (crashes) and/or unhelpful error messages.	Majority of required functionality not working. Confusing user experience
Program Architecture and Authorship 24%	Excellent adherence to coding standards i.e. indentation, use of comments, naming and capitalisation. Code easily maintainable and easy to follow. Appropriate use of 'new', 'untaught' features. Clear and sensible use of classes and methods. Good separation of UI and data functionality and strong use of OO principles of encapsulation and inheritance.	Code easy to follow. Maintainable and well presented. Sound application of taught material. Use of classes and methods but little evidence of OO principles. Weak separation of functionality.	Code confused and difficult to follow intended behaviour. Limited adherence to good coding practices or code which is otherwise difficult to maintain. Attempt at implementing taught material with limited success. Limited or confusing use of classes and methods.	Little or no attempt at solving the problem or most code submitted is auto-generated. Lacking in basic topics as covered in module. Classes poorly used if at all. Methods used sparingly or not at all.
Submission Presentation 2%	All submission requirements followed. Self-Assessment thoughtfully completed.	Majority of submission requirements followed. Self-assessment and checklist completed.	Self-Assessment or checklist only partly completed. Disks difficult to safely extract. Excess printed material or unnecessary binding/wallet. Used USB not disk.	Little or no adherence to submission requirements. No self-assessment form, missing backup or loose CD.

User Guide **5%**	Clear and easily read/watched. Effectively instructs the user on the use of the program. Extra value such as troubleshooting, future development plans, screenshots.	Effectively instructs the user on the use of the program. Sound structure, good presentation. Some added value.	Covers the main features of using the program. Structure is hard to follow or not logical, or presentation is haphazard.	Poorly set out or inadequate guide to even the basic use of the program.
Testing **5%**	Significant number of tests recorded in detail. Test data and procedure clearly evidenced. Good evidence of boundary testing and repeated testing where prior tests have failed. Evidence of testing throughout development process. Clear layout of information, with additional analysis	Good number of meaningful tests recorded with few details missing, or smaller number of tests at exceptional detail. Evidence of failed tests. Evidence of range of inputs being used. Clear, structured layout of information.	Reasonable number of meaningful tests recorded but with many details missing (e.g. test data, steps taken, date & result information). Small number of tests at a higher level of detail with few details missing. Functional layout of information.	Little or no evidence of testing, or trivial/similar tests repeated. Confused layout of results.