Assessment Specification

MOD008365 Introduction to Programming

Live Brief presented by Hive Learning

|  |  |
| --- | --- |
| Assessment Format: | Assignment 010/2 |
| Mode of Submission: | Canvas: Online submission via Canvas together with a demonstration of the final solution  We would suggest instead that submission is made on a browser from a computer or mobile device.  It is not recommended that you submit your assignment or take a quiz on a cellular or satellite connection. This is because loss of internet connection during assessment is most commonly associated with these services, although we understand that sometimes this is the only option. Wherever possible, we would recommend that assessments are submitted or undertaken on WiFi or hardwired (ethernet) connections.  Large files are best submitted on a hardwired internet connection (i.e., not WiFi). |
| Weighting: | This assessment is worth 80% of the Module mark |
| Assessment Level: | 4 |
| Element Learning Outcomes: |  |
| Marking Approach: | Fine-graded |
| Feedback Release Date: | N/A |

**Assignment 010/2 (worth 80% of final mark)**

Broken Petrol Ltd own a petrol station on the M25 and you have been tasked to design, implement, test, and document an automated demonstration of a Petrol Station Management application.

The implementation needs to be implemented in a Windows Console application written in C#. Ideally, you will create a .NET Core based project.

***1) Application Synopsis***

Broken Petrol Ltd must ensure that their petrol station is working at full capacity and as efficiently as possible to maximise throughput and therefore profits. The issue of efficient fuelling is paramount long delays in fuelling may result in drivers refuelling at another petrol station

***2) Application Description***

The automated demonstration application refers to a forecourt with 9 fuel pumps arranged over 3 lanes (three pumps in each lane). Vehicles are created and added to a queue. The fuelling starts when a vehicle waiting in the queue is automatically assigned to an available pump.

To add to realism of the demonstration not all drivers completely fill their tank, there is a finite time to fuel a waiting vehicle before it leaves the forecourt. That period is set to a random number of whole seconds. Should the vehicle not be fuelled during that time, it will leave the forecourt. Similarly, once a vehicle is fully fuelled it also leaves the forecourt. The management at Broken Petrol Ltd recognise that a fuel attendant is pivotal to the success of this operation, which is why they have decided to incentivise them by paying a commission of 1% (0.01) of the total sales to their hourly rate of £12.49. This works out as 8 hours shift \* £12.49 + 1% of the total fuel sales during that shift.

The amount of fuel dispensed is to be recorded.

A pump cannot service more than one vehicle at a time and the following values must be kept:

(1) Running total of the number of litres dispensed.

(2) The total fuel cost.

(3) The 1% commission.

(4) The number of vehicles serviced.

(5) The number of vehicles that left before they were fuelled.

***3) Marking* – Please read this section in detail**

The assignment will be marked based on the following three criteria:

* Functionality and Execution, 40%
* Program architecture, 40%
* Documentation and Testing 20% (functional and unit)

Time management is crucial – please make sure you leave enough time to implement and meet all three criteria.

***4) Functionality and execution***

Start by implementing the basic criteria, when all functions have been fully implemented, attempt to implement the next set of functionalities. Again, only once the previous set was accomplished in FULL.

It is recommended to create a new project for each functionality level you have attempted so that you do not end up losing code you might want to reuse later. Submit your best version.

Program architecture and authorship

Use the good programming guidelines and design features (activity diagrams, etc.) throughout the development cycle of the application. Adhering to guidelines retrospectively is time consuming and often leads to bugs.

Submission and presentation (3%) – See details below and please adhere to the presentation and submission instructions.

Application User guide (7%) – See details below.

Testing (10%) – The mark for this section considers bugs that are not documented in your test plan. Please test your program and test it well - reporting all the results!

Total Marks - 100%

The Pump can dispense 1.5 litres / second.

Counters 1 to 4 must be implemented.

***4) Program Functionality and Execution***

Core functionality: (40% +)

* A new vehicle is created every 1.5 seconds. (No need to randomise this timer at this level)
* When a vehicle is sent to a pump, the fuelling process will take 8 seconds. At the end of the 8 seconds the number of litres dispensed is recorded and the vehicle leaves the forecourt, freeing the pump for use.
* At this level, a newly created vehicle can wait until it is sent to a pump without a waiting time limit.
* The attendant presses the number of an available pump to send a car to it.
* One type of fuel (Unleaded)
* One type of vehicle that ever gets serviced by the petrol station.

Acceptable functionality: (50% +)

Additional to and overriding Core functionality:

* A new vehicle is created randomly every 1500 to 2200 milliseconds. And there can only be one car waiting to be serviced.
* When a vehicle is sent to a pump, the fuelling time will be random and take between 3000 and 5000 milliseconds. At the end, the number of litres dispensed is recorded and the vehicle leaves the pump.
* 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.
* There are 3 types of fuels (Unleaded, Diesel and LPG – Classes, ideally) and 3 types of vehicles (Car, Van, and HGV – Classes, ideally). 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. (Implement inheritance)
* All pumps serve all types of fuel.
* All counters must 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).
* A security screen is displayed when the app is first launched which the attendant can use to log in before using the app. The app then displays who is logged in on the main interface with an option to log out at the end of a shift.

Good functionality: (60% +)

Additional to and overriding Core functionality:

* 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.
* Each newly created vehicle will have a tank proportionate to the vehicle size.
  + Cars - maximum 50 litres,
  + Vans - maximum 80 litres
  + HGV - maximum 150 litres.
* HGV only use Diesel. Vans on both Diesel and LPG. Cars can use any type of fuel.
* Newly created vehicles will be created with a random amount of fuel already in their tank (which cannot be greater than a half of their total tank capacity). Fuelling time will be based on the above amount.
* A queuing system must 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.
* At the end of a shift the attendant’s wage should be calculated and displayed when they log out.
* All details such as Counter values, transactions and associated attendant details should be logged to file

Excellent functionality: (70% +)

Additional to and overriding Good functionality:

* You will need to exhibit excellent code elegance, noticeable additional functionality that enhances the application user together with an excellent User Guide and Complete testing of functionality.
* Cars will be automatically assigned a pump the screen will indicate their location.

You are encouraged to research and utilise built in C# functionality to aid with the Random aspects of this assignment. You **must** acknowledge and reference all sources of information and code examples used.

You are encouraged to discuss your ideas with your colleagues, however, any code or material you submit should be your own work, produced individually and not in collaboration with others!

ALL levels of functionality must implement a Console based User Interface (UI) 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).

Example User Interface

Queue

Cars: 1 Vans: 2 HGVs: 1

Row A \*\*\*\*\*\*\*\* 1 Busy \*\*\*\*\*\*\*\*\*\*\*\*\* 2 Avail \*\*\*\*\*\*\*\*\*\*\* 3 Avail\*\*\*\*\*\*\*\*  
Row B \*\*\*\*\*\*\*\* 4 Busy \*\*\*\*\*\*\*\*\*\*\*\*\* 5 Avail \*\*\*\*\*\*\*\*\*\*\* 6 Busy \*\*\*\*\*\*\*\*  
Row C \*\*\*\*\*\*\*\* 7 Busy \*\*\*\*\*\*\*\*\*\*\*\*\* 8 Avail \*\*\*\*\*\*\*\*\*\*\* 9 Busy \*\*\*\*\*\*\*\*

Litres sold:

Diesel: 1231

Petrol: 587

LPG: 81

Cost: £6152.12

1%: £61.52

vehicles serviced: 112

Left early: 23

***5) Program Architecture and Authorship***

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

* All classes should be 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, input 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.
* Further Object-Oriented principles, such as Inheritance/Composition and SOLID techniques are not required at this level and will be covered in detail at Level 5 (Object Oriented Programming Development), however, you are free to implement more advanced techniques if you wish.

***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.

Create a zipped file (.zip or .7z) containing:

* The project folder(s) containing the source code for the program.
* The user documentation in electronic form (.doc, .docx, rtf or odt). Excel workbooks (.xls, .xlsx) are acceptable for the testing section. The documentation needs to contain at least:
  + Activity Diagram,
  + Application user guide
  + functional Testing.

Clearly indicate in your documentation which functionality level you are attempting.

Submit the zipped file in Canvas (see Module’s page, Assessment heading for details) before the deadline. Keep a local copy of your work with the file timestamps, you need to provide us with another copy. Check that the zipped file is not corrupted: It is your responsibility to ensure that the zipped file you submit can be unzipped and that the files included can be opened. If the files cannot be read, and you do not possess a copy a mark of zero (0) will be awarded.

***7) Application User Guide***

This should be a clearly written, structured, and presented document describing how the application works and what can be visually seen in the demonstration. It should contain an exciting application synopsis that would entice the reader to buy the full application along with a simple set of instructions of how to run application. The file format should be either DOC, DOCX, RTF or ODT. Please remember that the intended audience for this document is not technical

MARKING

The assignment will be marked based on the following criteria:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Insufficient–Grade 0 – 39** | **Basic – Grade 40+** | **Satisfactory – Grade 50+** | **Good – Grade 60+** | **Excellent – Grade 70+** |
| **Program Functionality and Execution 40%** | Majority of required functionality not working. Confusing user experience | Most basic functionality working. User experience inconsistent. Messages/instructions unclear or not enough of them. Frequent exceptions (crashes) and/or unhelpful error messages | All basic functionality working. User experience generally easy to follow but with some inconsistencies. Some useful instructions. Occasional exceptions (crashes) and/or unhelpful error messages. | All listed intermediate functions implemented working. Usable program. Majority of errors handled without crashing. Clear messages and instructions to the user. | 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. |
| **Program Architecture and Authorship 40%** | 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. | 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 or methods. | Code well laid out and presented though might be somewhat difficult to follow intended behaviour. Some adherence to good coding practices. Attempt at implementing taught material with limited success. Minimal use of classes and methods. | Code easy to follow. Maintainable and well presented. Sound application of taught material with some evidence of researching around the subject to develop skills. Use of classes and methods but poor separation of functionality. | 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. |
| **Documentation and testing 20%** | Little or no adherence documentation No or little evidence of testing. | Basic documentation and testing little or inappropriate commenting | Satisfactory user guide and documentation. Appropriate commenting and variable names. Acceptable functional and unit testing | Good user guide and documentation. With intuitive commenting and variable names. Good functional and unit testing | Professional style documentation with excellent commenting. Very readable code that is easily understood. Full functional testing and unit testing. |

***8) Testing***

Full documentation of the testing carried out throughout the development of the software both unit and functional.

***9) Other Notes***

The intended audience for your documentation (except for the Application User Guide) is to be considered that of your fellow students.

• You should include references to any sources you draw upon that you did not create yourself.

• 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 4500 words essay.

• The deadline for submission is listed in Canvas.