Assessment Brief

COURSEWORK TITLE	Program Development Project
LEARNING OUTCOMES ASSESSED	All module learning outcomes [/d2l/common/dialogs/quickLink/quickLink.d2l?ou=793884&type=content&rcode=ntutest-1681178]
CONTRIBUTION TO MODULE	100%
DATE SET	Monday 4th October
DATE OF SUBMISSION	11pm Monday 10th January
METHOD OF SUBMISSION	NOW Dropbox Folder
DATE OF FEEDBACK	Monday 31st January
METHOD OF FEEDBACK	Written feedback via NOW; Verbal feedback available on request

1. Assessment Requirements

You are required to undertake a program development project, involving developing a software program using C# (see Section 2.1) and producing a reflective report (see Section 2.2). This is an individual project, so the program and accompanying report must be produced by yourself (see Section 7). You will also be required to give a live demonstration of your program to a module tutor (see Section 2.3).

1.1 Deliverables

You are required to:

1. Produce one or more C# code files that together form a working console application.
2. Write a reflective report. The report should include a title page, and, if you have any references, a reference list. The report has a maximum word limit of **2000 words**, excluding the title page, and any references or appendix. Do **not** include a copy of your C# code in the report.
3. Complete a Declaration of Authorship [../../Assessment/Declaration_of_Authorship-Example.txt?_&d2lSessionVal=LE0Hu7D5GhDqjhqBBl88Gttdl&ou=793884] form. This should include references to any pre-existing code that you adapted/reused in your program (see Section 7).
4. Give a 10-minute live demonstration of your program. The demonstration will be held after the project submission deadline. You will be contacted regarding the specific time and date of your demonstration.

1.2 Submission Instructions

You should submit the following to the dropbox folder:

1. A **ZIP archive** containing all of the files needed to build and run your program.
2. A **word-processed document** containing your report.
3. Your Declaration of Authorship [../../Assessment/Declaration_of_Authorship-Example.txt?_&d2lSessionVal=LE0Hu7D5GhDqjhqBBl88Gttdl&ou=793884] form.

If you have used Visual Studio as your development environment, then the simplest way to submit your files is to create a ZIP archive of the entire Visual Studio project folder. You should not submit any machine code, so please 'clean' your Visual Studio project before creating the ZIP archive.

2. Assignment Instructions

2.1 Program Instructions

You are required to produce a C# console application. You are free to choose the nature of the application, and its specific functionality. The application is not expected to be complete or detailed enough for real-world use — the purpose of this assessment is just to exhibit the extent of your software engineering abilities. However, the application must provide coherent functionality to a user, not just perform a series of unrelated actions.

Attempting an ambitious challenging application but then only partially completing the application is fine for this project; you will be assessed on the working functionality achieved, not penalised for any missing parts of the application. See Section 4 for advice and suggestions regarding application ideas.

Your C# program that implements the application should exhibit your skill in decomposing problems, and in selecting and using appropriate C# language features and library components to implement solutions to those problems. The application functionality should

be complex enough that using all of the following programming-language concepts would be appropriate for a high-quality software implementation:

- Console input and output
- Variables for storing data at run-time
- Conditional branching
- Repetition using loops (and/or recursive functions)
- Functions and/or methods
- File input and/or output
- Sequential data structures (arrays or lists) and/or dictionaries
- User-defined structure types and/or classes

To achieve the highest possible grade, the application functionality should also:

- Not have errors for valid inputs.
- Make clear to the user what inputs are expected.
- Be robust against invalid inputs (errors in console or file input, missing input files).
- Not have arbitrary or unnecessary limits on the amount of data (e.g. limited to 10 customers).
- Load input data from text file(s).
- Load configuration settings from text file(s).
- Use serialisation to save and load the program state between program runs.
- If the program is a data-processing application, then allow searching and sorting of the data.
- Solve problems different to those covered in the module programming exercises.
- Require the use of C# language features and standard-library components beyond those taught.

See Section 3 for the details of the assessment criteria. If you have any uncertainty about what is required, or have an application idea but are not sure whether it is of suitable complexity, then please ask a module tutor for guidance.

The substantial majority of your submitted program code is required to be your own original work. Therefore you should not produce the application by adapting an existing program or tutorial. However, a small minority of the program may consist of code reused/adapted from other sources, provided that the source is correctly acknowledged (see Section 7).

2.2 Report Instructions

You are required to produce a reflective report in which you:

1. Describe and evaluate your software-development approach;
2. Explain any problems encountered, and any remaining errors in your program.
3. Evaluate the code quality of your program.

Note that you are not being asked to document the design, functionality, or testing of the program, nor to provide a user manual.

In more detail, your report should consist of three sections, covering the following topics:

1. **Software Development Approach**. Describe how you carried out this program development project. For example, did you plan everything in advance, or did you develop it incrementally? Did you make a prototype? How did you plan? For example, did you draw any diagrams or write any pseudo-code? How did you test the program? For example, did you test regularly as you went along, or do all the testing at the end? (If you have any diagrams or pseudo-code, these can be included as an appendix, and will not count towards the word limit.)

Evaluate what went well and what went badly with this development approach. Explain and justify what of this approach you would change, and what you would keep the same, if you were to develop another C# console application.

2. **Problems Encountered**. Explain any significant problems encountered during the project, and how you dealt with them. Also identify any erroneous behaviour in your final program (i.e. anything that can cause your program to crash, or functionality that does not behave as it should).
3. **Code Quality**. Evaluate the quality of your program code. You should identify the main strengths and weaknesses of the code quality, and the main ways in which the code quality could be improved.

2.3 Demonstration Instructions

You are required to give a live demonstration of your program running, during which you should demonstrate the program's functionality (including how the program copes with erroneous input, if applicable). You should be prepared to answer questions about your C# code, and how it provides the program functionality.

3. Assessment Criteria

You will be assessed on:

- the complexity of the problems that have been decomposed;
- the specific functionality that the application provides;
- the quality of your program code;
- the awareness and insight conveyed by your report.

The detailed criteria for each of these aspects are listed in the grade grids below. (For conciseness, the grids use some technical terminology in places. Refer to the Technical Glossary [/d2l/common/dialogs/quickLink/quickLink.d2l?ou=793884&type=content&rcode=ntu-2063654] for any unfamiliar terminology.)

There is no grade for the demonstration itself, but attending the demonstration is compulsory, and a Zero grade will be awarded if you do not attend. **You will not receive credit for any code that that you are unable to explain during the demonstration.**

3.1 Problem Decomposition (20% of project grade)

Note: If any code in your program was reused/adapted from pre-existing code, then it should have an accompanying reference to the original code (see Section 7). Reused/adapted code will not earn credit for any problem decomposition that code already achieves, but will earn credit for the skills you have shown in adapting that code to decomposing the problems in your application.

Criteria	Distinction	Commendation	Pass	Fail	
Complexity	Challenging domain-specific problems decomposed, of greater complexity than the programming exercises	Problems decomposed of similar complexity to the 'Advanced' programming exercises.	Problems decomposed of similar complexity to the 'Core' programming exercises.	Problems decomposed of less complexity than many of the 'Core' programming exercises.	I c

exercises.

Criteria	Distinction	Commendation	Pass	Fail

3.2 Program Functionality (30% of project grade)

Note:

- The program functionality must have a coherent purpose. No credit will be awarded just for having C# language features or library components present in your code, or for code that does not contribute to this purpose.
- If errors in the program code prevent parts of the code from working or being reached, then there will be no credit awarded for the functionality those parts of the code are intended to provide.
- If the majority of the program code is not your own individual work, then you will only receive credit for the functionality provided by the remainder of the code. You may also receive a penalty for Academic Misconduct (see Section 7).

Criteria	Distinction	Commendation	Pass	Fail
Control-flow	Nested definite and indefinite repetition, and branching.	Definite and indefinite repetition, and branching.	Repetition and branching.	Branching onl

Criteria	Distinction	Commendation	Pass	Fail
Input/output	Loading of configuration settings from user-friendly configuration files. Loading and processing of input data from text files and console,	Loading of configuration settings or input data from text files. Processing input data from console input, with user-	Writing output to text files and console.	No file usage Some consol input or

Criteria	Distinction	Commendation	Pass	Fail
	with user-friendly console output. Saving and loading complete program state between runs using serialisation.	friendly console output. Saving and loading some information between program runs using serialisation.	Some processing of console input.	output, but n processing o console input
Data Handling	Arbitrarily sized homogeneous collections of compound heterogeneous data **OR** two-dimensional data structures.	Unnecessarily size-restricted homogeneous collections of compound heterogeneous data. Some searching **OR**	Homogeneous collections of primitive data.	A small finite amount of data.

Criteria	Distinction	Commendation	Pass	Fail
	Sorting, searching and updating of data.	updating of data.		
Robustness	No run-time errors for valid and invalid inputs, including console input and missing / corrupt input files.	No run-time errors for valid inputs. Almost all invalid console input is handled without run-time errors.	Mostly free of run-time errors for valid inputs, except for unusual inputs / circumstances. Invalid console input may cause	Run-time errors occur for typical valid inputs / circumstance

Criteria	Distinction	Commendation	Pass	Fail
			numerous run-time errors.	

3.3 Code Quality (30% of project grade)

Criteria	Distinction	Commendation	Pass	Fail
Readability	Well-chosen indicative names that clearly indicate their role.	Most names are fairly indicative of their role.	Roughly half of the names are somewhat indicative of their role. Small amounts of redundant code.	Layout is po with substantial scope for improvemen Most variabl names do nc indicate thei role. Substantial amounts of redundant code.

Criteria	Distinction	Commendation	Pass	Fail
Named Constants (Maintainability)	All program constants named, with no magic numbers / strings.	Most program constants are named, but a few magic numbers / strings.	A fair attempt at naming program constants, but at least half are magic numbers / strings.	Almost all program constants a unnamed.
Code Repetition (Maintainability)	No significant missed opportunities to introduce loops or procedures / functions / methods.	No significant missed opportunities to introduce loops or procedures / void methods. Some missed opportunities to introduce and reuse functions / non-void methods.	No significant missed opportunities to introduce loops. Some missed opportunities to introduce and reuse procedures / void methods.	Some misse opportunitie to introduce loops to combine repetitive code.

Criteria	Distinction	Commendation	Pass	Fail
Modularity	Consistent organisation of code into multiple classes and small functions / methods.	Fairly good organisation of code into functions / methods, though some may be very large and could be further broken down.	Several useful functions / methods defined and used, but the majority of the program logic is in a single large block. Structure types / classes with fields defined and used.	The executable program coc is organised into one or more procedures, with no functions / methods defined. Inappropriat reliance on global variab for communicat

Criteria	Distinction	Commendation	Pass	Fail
Reusability	Classes provide an interface to objects, encapsulating more complex internal logic. Functions / methods created that typically have 1-3 / 0-2 parameters, and do not use reference parameters for value types.	Classes defined that provide objects with private fields and public constructors and methods. Functions / methods created.	Classes / structure types defined that provide constructors. Procedures (but not functions or methods) with parameters defined.	Procedures defined but without parameters, and no functions or methods defined.

Criteria	Distinction	Commendation	Pass	Fail
Language and Library Use	Code consistently solves problems directly with well-chosen language features and library components. Numerous uses of features / components beyond those taught.	Code mostly solves problems directly using appropriate language features and/or library components. Some use of features / components beyond those taught.	Code often solves problems indirectly, and/or is unhelpfully verbose. May use some workarounds and/or placeholder values to compensate for poor structure or inappropriate features / components.	Code is convoluted, often doing substantial unnecessary work. May often use inappropriate language features and library components accomplish results 'by accident'.

3.4 Report (20% of project grade)

The report is a piece of reflective writing. Credit is earned for conveying what you have learned about software development, and for showing awareness of the quality of the program code that you have produced. No credit is awarded for describing the design or functionality of the application.

Criteria Criteria	Distinction Distinction	Commendation Commendation	Pass Pass	Fail Fail
Software Development Approach	Clear critical evaluation of the software-development approach. Well justified explanation of lessons learned.	Mostly clearly conveys the main aspects of the software-development approach. Some positives **AND** negatives of the approach identified. Lessons learned explained, but justification may be tenuous.	Mostly clearly conveys some aspects of the software-development approach. Some positives **OR** negatives of the approach identified. Reasonable lessons learned stated, but without (valid) iustification.	An attempt has been made to convey the development approach, but it is very limited and/or unclear. No identification of positives or negatives of the approach. Lessons learned do not pertain to the development approach, or are factuallv

Criteria	Distinction	Commendation	Pass	Fail
Problems Encountered	Clear description of problems encountered and how they were dealt with. All possible erroneous behaviour is identified (if any).	Mostly clear description of problems encountered and how they were dealt with. The most significant erroneous behaviour(s) are identified.	Mostly clear description of problems encountered. Some erroneous behaviour is identified.	Mostly unclear description of problems encountered. Some discussion of erroneous behaviour, but mostly unclear or incorrect.
Awareness of Code Quality	Thorough well-justified critical evaluation of code quality.	Correctly identifies some of the strengths **AND** weaknesses of the code quality.	Correctly identifies some of the strengths **OR** weaknesses of the code quality.	Evaluation of code quality is mostly unclear or incorrect.

4. Advice on Getting Started

Once you have an idea for an application, you should think about who the (hypothetical) intended user is. This will help you clarify the required functionality of the application. For example, if your idea is to make a 'Shop' application, you should consider whether it is intended to be used by a customer, a manager, or a front-line sales staff member — each of these users would require significantly different functionality.

Once you have determined a user (or users), then you should think about the specific requirements of your application. That is, what functionality should it provide, what inputs will it take, and what outputs will it produce?

If you are struggling to think of an idea for a program, I recommend that you look through the programming exercises and textbook examples for inspiration. These contain a variety of different types of programs, and it would be fine for you to create something similar. If you are still stuck, here are some additional suggestions:

- A mortgage-assessment calculator. This could be used by either a staff member of the mortgage provider, or the customer. It would take customer details (such as property price, deposit, customer age), and then offer a range of mortgage deals. For a high grade, this should involve compound interest calculations, and should load available mortgage options from files.
- An appointment-booking system for a hair-dressing salon. This would be intended to be used by a staff member at the salon, and could allow for storing customer details, booking and cancelling appointments, and tracking staff availability.

- A computer version of a card game or simple board-game such as 'Connect 4'. This would be used by one or two human players.
- A text-based adventure game. This would be intended to be used by a player of the game, and would allow them to move North/South/East/West around a map. At each location, the player would be given a description, and options as to what to do. For a high grade, this should allow a player to save their game and resume later.
- A text parser that does some analysis on written text, such as computing the typical sentence length or linguistical complexity, or identifying sentences that end with prepositions.
- A simulation of some simple discrete process, such as Conway's 'Game of Life' [4].
- A children's maths quiz. This would be intended to be used by a child taking the quiz, and would randomly generate mathematical problems. For a high grade, this should record a score for each user, save all past scores in a file, and be able to display a high-score table in the console.

5. Resources that may be Useful

The module reading list contains several introductory C# textbooks [1, 6, 9]. These are all available from the NTU library, either electronically or as hard copies. For advanced code-quality advice, the textbook by Martin [5] is an excellent resource. See the module reading list for reading notes on each book, and to access the electronic copies.

An exemplar report for this assignment is available from this link [../../Assessment/Exemplar_Report.pdf?_&d2lSessionVal=LE0Hu7D5GhDqjhqBBl88Gttdl&ou=793884]. Refer to this if you're not sure what style to use when writing the report.

NTU library offers support with academic writing, and with mathematics and statistics skills. Individual appointments with librarians or student mentors can be booked online [2]. The library has also published a guide for the correct use of citations and references [3]. NTU also provides online advice for avoiding plagiarism [7].

Note that failure of personal IT equipment is not considered to be extenuating circumstances, and will not provide grounds for an extension to assessment deadlines. **You are therefore strongly advised to maintain a backup copy of your project work at all times.** The University provides the NTU OneDrive [8] remote-storage drive for your use, though you could also use a personal cloud service, or a personal memory stick or portable hard drive.

6. Feedback Opportunities

You may request verbal feedback on your work-in-progress project during the lab sessions. You will receive written feedback regarding both your program and report within three weeks of the submission deadline.

7. Referencing, Plagiarism and Collusion

This is an individual project. This means that your program should be developed individually, and the report describing your program should be individually authored. However, discussing ideas, programming techniques and reference sources with other students is encouraged, provided that you then individually apply those techniques and ideas within your project.

In the field of Software Engineering, it is standard practice to reuse and adapt pre-existing code (subject to licensing restrictions). However, for this project, there is a requirement that ***the significant majority of submitted program code is your own original work***. You may reuse or adapt a small amount of pre-existing program code, but such code must be identified and acknowledged as follows:

- You should give the full reference details as a comment in the file containing the code, above or below the reused/adapted piece of code. The comment should also clearly identify the extent code that has been reused/adapted (e.g. the next 5 lines, or the entire function, or the entire class).
- You should also give the reference details in the reference list in your Declaration of Authorship [../../Assessment/Declaration_of_Authorship-Example.txt?_&d2lSessionVal=LE0Hu7D5GhDqjhqBBl88GttdI&ou=793884] form.

Note that the use of standard-library components is encouraged, and individual library components do not need to be referenced.

Any reference must identify precisely where the original code is to be found so that it can be examined when assessing your work. For example, a reference to a website must identify the specific webpage on which the code is found, whereas a reference to a textbook must identify the specific page(s) on which the code is found. If the precise location of the original code is not specified, then your work will be assessed under the assumption that the code was copied verbatim with no adaptations. For guidance on how to reference a webpage or textbook, see the NTU Guide to Citing References [3].

Failure to identify and correctly reference reused or adapted code is **plagiarism**. This includes code written by another student — such code, if not publicly available, can be referenced as a 'personal communication', and a copy of the original code submitted as an additional file with your submission. Furthermore, you should not jointly develop a program with another student — doing so is **collusion**.

If a substantial amount of your program code was reused or adapted without being correctly referenced, then the entire program will be assumed to have been copied verbatim with no adaptations, and a Zero grade awarded for the project. (It is not the assessor's responsibility to attempt to determine what parts of the submitted code are your own original work.)

In the report, any text that was not written by yourself must be enclosed in quotation marks, and the source must be acknowledged by providing an accompanying citation referring to an entry in your reference list. Any diagrams that you did not create should also be cited and referenced. Failure to do so is **plagiarism**. If you summarise or paraphrase another person's work without a citation and reference to acknowledge the source, then that is also **plagiarism**. See the NTU Guide to Citing References [3] for guidance on correct use of citations and references.

If you are in any doubt about whether your use of material from another source is correctly acknowledged, then ask a module tutor for guidance before submitting your work.

8. Aspects for Professional Development

This project will give you practical experience of software development, and increase your familiarity with the C# programming language. It should also raise your awareness of code-quality considerations.

References

[1] Jamie Chan. Learn C# in One Day and Learn It Well. CreateSpace Independent Publishing Platform, revised edition, 2017. [https://rl.talis.com/3/ntu/items/451090C9-BE80-D8CF-7D4E-0206EE3ADC68.html]

[2] NTU Library. NTU Library bookings. [Accessed 28/09/20] [https://librarybookings.ntu.ac.uk/]

[3] NTU Library. Citing references: A guide to NTU Library Harvard Style, 10th edition, 2016. [/d2l/lor/viewer/view.d2l?ou=52836&loIdentId=25435&contentTopicId=1019510] [/d2l/le/content/52836/viewContent/1019510/View.]

[4] LifeWiki. Conway's game of life. [Accessed 28/09/20] [https://www.conwaylife.com/wiki/Conway%27s_Game_of_Life]

[5] Robert C. Martin. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall, 2008. ISBN 978-0132350884. [https://rl.talis.com/3/ntu/items/3081C691-AD92-595B-2118-0C4886B11C9A.html]

[6] Rob Miles. C# Programming Yellow Book. Imprint: Independently Published, 9th edition, 2018. ISBN 978-1728724966. [https://rl.talis.com/3/ntu/items/4A8399CB-F2FC-31AF-E314-FEF7533D6142.html]

[7] NTU. Plagiarism. [Accessed 28/09/20] [https://www.ntu.ac.uk/m/skills-for-success/copyright-and-plagiarism]

[8] NTU Information Systems. OneDrive basics: Student user guide, 2016. [/d2l/lor/viewer/view.d2l?ou=6605&loIdentId=44358&contentTopicId=1667248]

[9] Radek Vystavěl. C# Programming for Absolute Beginners. Apress, 2017. ISBN 978-1-4842-3317-7. DOI: 10.1007/978-1-4842-3318-4. [https://rl.talis.com/3/ntu/items/5AC08A51-18DE-A484-FD1E-82A7FA28B66F.html]

Go to top