# ECS401/IOT401 Procedural Programming
# Assessment Booklet: September 2023

**YOU MUST READ THIS BOOKLET CAREFULLY. IF YOU DO NOT FOLLOW THE INSTRUCTIONS PENALTIES WILL APPLY TO YOUR WHOLE COURSEWORK GRADE. YOU WILL LIKELY NOT PASS. COLLUSION WITH OTHERS (PEOPLE/AI) OR COPYING ASSESSED WORK FROM OTHERS IS CHEATING EVEN IF YOU MAKE CHANGES. KEEP YOUR ASSESSED WORK SECURE - DO NOT LET OTHERS SEE IT.**

**Work hard on the assessed work as well as other work and learn the skills you MUST have to do well on your later modules**

# Contents



## 1. Learning Outcomes

On passing the module you will be able to:

- read programs
- write simple programs from scratch and larger programs in steps following a style guide.
- write **high quality human readable** programs (including that are documented through literate programming)
- work out what a program does without executing it
- test and debug programs to ensure they work correctly
- explain, compare and contrast programming constructs and discuss issues related to programming
- Use programming terminology correctly (“speak like a computer scientist”)

The assessment, consists of AUTUMN coursework (50% of your overall mark) and a JANUARY final assessment (50% of your overall mark). It develops these skills and tests that you have gained them. Both coursework and exam explicitly test the learning outcomes of the module.

## 2. Rules of Helping Others

**IF any of your assessed work (programs, explanations, test/exam answers) is not done by you or is substantially the same as someone else's or another source (whoever wrote the original)**

**THEN**

**you automatically lose the grade. You will also be investigated for cheating.**

**If you share your assessed work with others and they hand it in as their own (even with changes) then you will both be cheating and both be reported and suffer the penalties.**

**As in this module the coursework is classed as a single element, if you cheat on any piece of work, that will count as cheating on the coursework as a whole and penalties will apply to the whole coursework (eg the whole coursework failed and new work done with max mark 40%).**

**Some ways of helping other students (outside of tests) are really good.**

DO explain programming concepts to each other
DO explain using examples different to the actual exercises they are working on
DO explain how programs work
DO test their programs for them, pointing out bugs (but not giving solutions)
DO get them to explain their program to you (this is a good way to debug code)
DO show them ways to debug a program
DO point them to the right place to look to find bugs, rather than just pointing out the bug itself
DO explain what compiler messages mean (but not what their mistake is or how to fix it)
DO get them to help test *your* programs
DO get them to explain things that you don't understand, but they do
DO explain WHY things are the way they are
DO ask them questions that lead them to work out answers for themselves

**Some ways of trying to help are unhelpful (or just very stupid)!**

NEVER take over some one else's keyboard.
NEVER just give someone else solutions whether programs, program fragments or otherwise
NEVER pass code or written explanations of code to someone else
NEVER give answers without giving understanding.
NEVER accept explanations from someone else you do not understand. Have them explain again.
**NEVER EVER** give other people or let them see copies of your assessed programs or explanations.
**NEVER EVER** take someone else's program and (even with changes) present it as your own
**NEVER EVER** take someone else's assessed work and claim it is yours even if you modify it it is still cheating.
**NEVER EVER** help someone during an assessed test, or share your answers.
**NEVER EVER** cut and paste from anywhere including the Internet.
**NEVER EVER** get an AI to do assessed work for you.
**ALWAYS** keep your account and your work secure and cannot be accessed by others.

You can take Paul's programs from QM+ and edit them to do something different. But if you do make sure you understand how they work first. Experiment with them until you understand.

*It is only "help" if afterwards the person you helped can do it themselves without any more help.*

*You must NOT help or get help from others doing an assessed test at all.*

## 3. Overview of Assessment

### January Final Assessment ("Exam")

**Aim:** The final assessment (the Jan exam) ensures you have gained the skills developed by doing the coursework and other exercises. It tests BOTH programming AND written theory skills.

The written exam is at the end of the module, in January.

Gaining the skills and knowledge developed on this module is vital for later modules you will take. The concepts you have to be able to explain are core to later modules. You need to be able both to program and write to get a degree and so get a good job (even if not a programming job).

### Autumn Coursework

**Aim:** If you do the coursework well and work hard you will gain the core skills both to pass the exam and that are essential for your later modules as well as skills for your CV and future work.

The coursework consists of (and you must do all of these or you will get 0):

- A series of short programming exercises (worth 2 A-F grade)
- A programming mini-project (worth 2 A-F grades)
- A test (based on your best in each part from two) consisting of:
  - A theory test (similar question style to final exam) (worth 1 A-F grades)
  - A dry-run test (also similar to the final exam) (worth 1 A-F grades)
  - A programming test (also similar to the final exam) (worth 2 A-F grades)

The test grades are based on the mid-term test and the end-term test.

- Your theory test grade is the highest of the two theory grades you get.
- Your dry run test grade is the highest of the two dry run grades you get.
- Your program grade is the highest of the two program grades you get. It counts twice.

Overall this gives 8 A-F grades (2 for shorts, 2 for mini and 4 for the tests), turned in to a single coursework mark at the end of the term (see the next page for how this is done). Students who do little work on the coursework or start it late, fail the exam and so the module. They also gain no skills so waste a chunk of their life. Those who work hard from day 1 and throughout term, working on the coursework FROM THE START generally pass and gain useful skills. The non-assessed exercises (eg the JHUB interactive notebooks) are designed so that if you do them you will learn the skills and find the assessments easy.

*If you don't take all the above components (miss both tests, or fail to get ANY shorts or miniproject marked or submitted, or cheat on anything) you fail the **WHOLE** coursework (0% overall).*

### What must I do to Pass the Module?

- To pass the module you must pass the FINAL ASSESSMENT ("exam") (gain 40% or better) AND your combined mark from your final exam and coursework mark must be over 40%.
- If you fail the final assessment, your overall mark is capped at 39%

### Resits - Fail the module and you take a resit exam

- **If you do not pass the module** (because you fail the exam OR your combined exam-coursework mark is below 40) then you have one more chance to pass the module.
- You will take a resit exam (whatever the reason for failing - even if you failed the coursework).
- You must then get over 40% in that single resit exam alone to pass the module. If you do pass you will be given a capped mark of 40%.
- The resit will be held in the Late Summer Exam period (normally August).

## 4. Calculating your Final Coursework Grade

Your overall grade for the coursework is calculated as follows from the 8 components:

| **If you gain from the coursework components (start at the top and work down):** | **your final skill level grade is** | **which gives you a final coursework mark of …** | **Have you passed the coursework?** |
|---|---|---|---|
| Any part missed or cheated on (ie no shorts, no mini, no mid-term or no end-term submitted at all) | Grade Q | 0% | NOT PASSED |
| 8A+s | Grade A* | 100% | PASS |
| 7A+s and 1A | Grade A++ | 96% | PASS |
| 6A+s and 2A | Grade A+ | 86% | PASS |
| 6As and 2B (or better) | Grade A | 76% | PASS |
| 6Bs and 2C (or better) | Grade B | 66% | PASS |
| 6Cs and 2D (or better) | Grade C | 56% | PASS |
| 6Ds and 2F (or better) | Grade D | 46% | PASS |
| 6Ds and 2Gs (or better) | Grade F | 30% | NOT PASSED |
| 8Gs (or better) | Grade G | 16% | NOT PASSED |
| Otherwise | Grade Q | 0% | NOT PASSED |

If you do not attempt every component you will fail (0% overall).

**If you miss a test or major deadline contact Paul / Juntao / Ray URGENTLY by emailing**

**ecs401-personal-queries@lists.eecs.qmul.ac.uk**

Always include your 9 digit student ID and email from your college account.

Only if you pass the exam is your mark combined with your coursework mark to give the overall mark. If you failed the exam or passed the exam but failed to get 40% overall you will have to take the resit exam later in the year (See above). The maximum module grade you can then get is 40%.

Why is your mark calculated like this?

The reason the coursework is calculated using skill levels is to ensure that if you get a grade you have ***consistently achieved*** that level of skill. If you can really program to a pass level then you must be able to do it consistently both in theory and practice (and in test conditions). However we have added some flexibility allowing you to do badly on some – an odd nightmare day is allowed!

The aim is to make you focus on gaining the skills – which is the point – not on just gaining marks (with or without skills).

**Unless you gain the skills consistently you will not pass the exam.**

## 5. Assessment Deadlines

For the purpose of this module weeks are counted as follows

| Wk | week<br>start - end<br>(mon - fri)<br><br>2023 | Advisory Deadlines<br>You must do the programs and get them marked week by week (**by lab viva**). There are no weekly hard deadlines for getting any marked but if you are behind the schedule below then you are not keeping up so get help. Your grade is based on how many completed by the end of term so you do not need to do all to pass. | **Major Deadlines you MUST NOT MISS as you FAIL the module if you do miss them.** |
|---|---|---|---|
| **1** | 25 Sept - 29 Sep | | |
| **2** | 2 Oct - 6 Oct | **Short programming exercise 1**<br>marked no later than YOUR allocated lab | |
| **3** | 9 Oct - 13 Oct | **Miniproject level 1**<br>marked no later than YOUR allocated lab | **MINI-PROJECT WEEK**<br>Everyone must start their miniproject this week. |
| **4** | 16 Oct - 20 Oct | **Short 2 and Miniproject level 2**<br>marked no later than YOUR allocated lab | |
| **5** | 23 Oct - 27 Oct | | **MID-TERM TEST**<br>in YOUR timetabled lab |
| **6** | 30 Oct - 3 Nov | **Short 3 and Miniproject level 3**<br>marked no later than YOUR allocated lab | |
| **7** | 6 Nov - 10 Nov | **Short 4 and Miniproject level 4**<br>marked no later than YOUR allocated lab | |
| **8** | 13 Nov - 17 Nov | *DON'T LEAVE IT TOO LATE TO GET PROGRAMS DONE AND MARKED!* | At most 2 programs can be marked each week |
| **9** | 20 Nov - 24 Nov | At most 2 programs are guaranteed to be marked each week | At most 2 programs can be marked each week |
| **10** | 27 Nov - 1 Dec | At most 2 programs are guaranteed to be marked each week | **END-TERM TEST (online open book) TBA - PROBABLY either in place of lecture or Wednesday pm** |
| **11** | 4 Dec - 8 Dec | At most 2 programs are guaranteed to be marked each week | At most 2 programs can be marked each week |
| **12** | 11 Dec - 15 Dec | At most 2 programs are guaranteed to be marked each week | At most 2 programs can be marked each week<br>**YOUR ALLOCATED LAB IS THE FINAL DEADLINE FOR MARKING PROGRAMS.**<br>Hand in paper signature sheet (physically) at end of lab and previously marked programs (online). |

- You must get your programs assessed week-by-week in your allocated lab as you go along.
- You should get programs completed, tested and marked well before the above final deadlines.
- Aim to finish and get marked at least one each week to give you the best chance of a high grade.
- **Demonstrators will guarantee to mark AT MOST 2 programs for each student in weeks 8 onwards** i.e., each week 2 shorts OR 1 short and the mini-project (to any level including jumping multiple levels BUT you MUST get it marked 3 times on 3 weeks at 3 levels)

## 6. Literate Programming, good style, etc

**Programs are not just for computers to read. They must be easy for humans to read. You must learn to write high quality documentation using Literate programming.**

Programs have to be developed over a period of time, are modified for new purposes, new needs found, bugs fixed, and are worked on in teams. All of this means they often need to be understood by people other than the original author. Therefore clear, detailed explanations are important.

Explaining your own code is also a good way to solidify your own understanding. **Literate programming** is one way to help with both these aims. You must learn to do it well.

Literate programming is a form of defensive programming (see the workbook). It involves writing detailed explanations of code you write, method by method in a way that is interspersed with the code and test plans. Jupyter (JHUB) Interactive notebooks (as used for the interactive exercises) are one way to do this and the way you MUST follow for the assessed programs. Literate programming in interactive notebooks complements (not replaces) the use of comments.

### Literate Programming of Assessed Exercises in Interactive Notebooks

- You must complete both assessed short programming exercises and an assessed mini-project (see the following pages).
- All must be developed as, and will be marked as, **literate programs** in JHUB **interactive notebooks.** Your miniproject must **also** work as a standalone program from level D on.
- You should create two versions of the program and include them in the Interactive Notebook
    - A literate version, broken into methods (see below), through the notebook with appropriate "Markdown" descriptions and test calls for each.
    - A final FULL program version at the end of the notebook with all methods and call.
        - for the miniproject level D on, this full version should compile and run outside the notebook as a standalone program too.

#### Literate programming descriptions of assessed programs

- Examples of what the literate programming versions of the assessed programs should look like are given in the Assessment section of the Interactive Notebook server
    - **https://hub.comp-teach.qmul.ac.uk**
- Each method should have a detailed description as text (in Markdown) before the method
- It should describe in detail how it works, giving an argument that it does work as required.
- You should explain why each method does the right thing in the notebook.
- You should test each method as you write it in the notebook.
- The full notebook for each exercise MUST be saved as a pdf and submitted to QM+ **AFTER** is has been marked as SATISFACTORY and signed off by the demonstrator.
    - If there are problems so it is not signed off, then you just modify it and get it assessed again until it is signed off.

#### Comments in assessed programs

- Every method should also have in-program comments in Java syntax.
- Each method should be preceded by a comment explaining what its purpose is
    - what it does (not how it does it which can be written in the Markdown)
- The program as a whole should start with a comment that includes
    - Author and Student number (the 9 digit number on your ID card)
    - Date
    - Version (eg new version after each time it has been assessed if assessed multiple times)
    - An overview of its purpose
    - A justification of why it passes the level against the exercise's requirement bullet points.

## Style Guide: Important considerations about your code

You are expected to write good maintainable code. The precise requirements increase through the units (see the tick boxes at the start of the exercises) - you are expected to gradually raise your game but ultimately you are aiming your programs to meet the requirements of the STYLE GUIDE booklet found on JHUB. You must read the STYLE GUIDE. The following are some key points.

1. **Be written as a literate program** (see above, not required in tests and exams)
   a) Be well documented (an industry requirement)
   b) Be thoroughly tested (an industry requirement)
   c) Include test plans (an industry requirement) as part of the literate document.
2. **Be a procedural program** (as this is a procedural programming module)
   a) All methods to be contained in a single class
   b) Other classes used only to define records (so have no methods)
3. **Do not use while (true) {…} constructs and break statements**
   a) It should be clear from reading the first line of a loop how and when it terminates. Otherwise the whole loop must be read to understand it. While loops should therefore be exited from the top of the loop because the condition has turned false.
   b) NEVER use **true** as an exit condition because of this
   c) Loops should NOT exit using break statements inside the body of the loop
   d) Loops should NOT exit using return statements inside the body of the loop (except in exceptional cases where the loop and so method is only a few lines long)
4. **Have excellent basic style** (an industry requirement)
   a) Use comments well (each method should have a comment saying what it does).
   b) Use indentation and blank lines well
   c) Name variables well
   d) Use final variables for literal values well
5. **Ensure all variables are defined inside methods** and in as small a scope as possible
   a) No variable should be defined outside the curly brackets of a method definition
   b) Information stored in variables should be passed to methods using arguments
   c) Information should be passed back from methods using return
   d) Variables should be declared in the inner-most block possible given their use (ie have smallest scope)
   e) Variables should be defined at the start of a method (or other block) not in the middle.
6. **Be decomposed well into separate methods** (basic good coding)
   a) Be fully decomposed into methods - so each method does a well defined task (no big, hard to understand method doing lots of different things)
   b) Each method should be documented and tested separately (in the literate document when written as one)
7. **Deal elegantly with any input** (basic good coding)

By the later programs, tests and certainly the exam, you must be following these rules well if you are to do well.

## 7. Short Programming Exercises

### This is assessed: it counts for 2 A-F grades

- Whatever the grade you get for the short programming exercises, **you get that grade twice**.
- The more you manage to complete by your last lab, the higher the grade you will get.
- They are intended to be relatively simple, developing your understanding of the constructs introduced in a single lecture.
- You must complete each to a satisfactory standard and get it assessed by a lab Demonstrator before moving on to the next. It meets the standard for that level if
    - it does the required task correctly
    - it meets ALL the tick box criteria given at the start of the exercise
- You should do all the short exercises in JHUB - it is the JHUB version that is assessed.
- **ALL programs must be written in Java in a procedural programming style** as covered in the lectures. There must be ONLY ONE class containing methods (any other classes are just data structures and contain NO methods at all.)

**You may have a particular program assessed more than once** if it was not up to the standard required on the first attempt. Your Demonstrator will explain any problems when he/she assesses it. You may also be asked questions about your program, or be asked to make modifications to prove that you understand it. When the program has been completed, **and you have demonstrated your understanding of it, to a satisfactory level**, the tutor will sign it off (you should check that you have correctly been given the signature on the sheet - see the end of this booklet).

Add a note in your literate document that the program has been signed off.

Note the **marks are not just for presenting a correct program but for convincing the assessor that you have the required skills and understanding**. If you cannot answer questions on your program or cannot make simple changes on the spot then you have not reached the required level. You will have to explain in writing similar things in the exam so this is good practice and if you do not gain the understanding you are on course to fail the exam.

The interactive programming exercises, and those on the module QM+ site for each week will help you develop the skill needed to do the assessed ones as well as give you extra practice to ensure you can program the simpler programs before you move on to more complex ones in subsequent weeks.

### Test your short programs ! LOTS

In industry most programming effort goes on testing – looking for mistakes in apparently working programs – not in writing those programs. Treat the Demonstrators as your clients who don't expect to be given faulty code to sign off!

The notes after each exercise give guidance on some things to look for in testing and some ideas on how to go about it. It is not exhaustive of course. You must use your own skills to find all the problems. More tips on ways to find problems as well as debugging can be found in the week-by-week sections of the module workbook. Interactive notebooks are a good place to test methods.

### Your programs must be written with STYLE

Code is for humans to read too, not just computers. Your company also has a policy that style rules (comments, indentation, variable naming, etc) MUST be adhered to, and all code goes through code audit, where others read it to look for problems. They have found good style is vital if code is to be easily read and modified - on seeing the code work, clients often think of extras to be added. When testing you should also check the style of the code too.

**YOU MUST WORK ON THE SERIES OF EXERCISES IN THE FOLLOWING PAGES**

- SHORT 1-8 (the more you do the higher the grade) AND
- ONE of the MINIPROJECTS through its stages (the more stages the higher the grade) AND
- GET THEM MARKED IN VIVAS IN THE LABS WEEK-BY-WEEK.
- When each program works, test it and check it ticks all the requirement boxes at the top,
- then have it assessed by a demonstrator.
- **IT MUST MEET THE TICK BOXES AT THE TOP OF EACH PAGE TO BE SIGNED OFF**
- If you are having problems or don't understand the requirements ask for help.
- Make sure you get a signature on the signature sheet.

## Grade G -: Short Assessed Programming Exercise 1: Output

**To pass this exercise you must demonstrate that you are able to do the following in solving the problem**

- ❑ **Write a literate version of the code**
- ❑ **EXPLAIN HOW YOUR PROGRAM WORKS**
- ❑ **Write simple code that compiles and runs in JHUB**
- ❑ **Write code that correctly uses output instructions**
- ❑ **Write code split in to methods**
- ❑ **Include simple comments – at least the actual author's name, student number and date**

**Big Initials** Write a program to print out your initials down the page in block letters using the same letter to draw it out of (see below). A blank line should separate the initials. Each initial MUST be printed by a separate method. For example, my initials are PC. My program should therefore print:

```
PPPPP
P    P
PPPPP
P
P

CCCCC
C
C
C
CCCCC
```

One method prints the 1st initial (for me the P) and a 2nd method prints the second initial (for me the C).

These methods must be both be called by a third method in sequence.

Each method must be documented and tested separately in the literate program document.

HINT: Plan what your output will look like before starting to write the program

You can modify one of the programs from the JHUB interactive notebooks rather than writing it out from scratch. **See the PC-ShortAssessedExercise1 notebook in the MY ASSESSED PROGRAMS directory of JHUB for my personal answer to the exercise.**

## TESTING Short Assessed Programming Exercise 1: Output

**You should check as a minimum that it meets the tick boxes at the top of the page and:**

- ❑ **All methods and the program as a whole compiles correctly**
- ❑ **Each method works on its own**
- ❑ **The program runs correctly**
- ❑ **It prints EXACTLY the right thing (reread the above description and make sure you didn't miss anything)**
- ❑ **Double check all comments and literate program descriptions make sense for THIS program**

As this program just does output you should check the output carefully – does it show both your first and last initial (at least). Are the correct letters used to form the shapes? Is there a blank line between the letters?

*Remember, programming can be tricky at first if you are a novice, but …*
*you are capable of learning to program if you keep at it. The more programs (however small and simple) you write the easier it gets. Doing the extra programming exercises from the interactive notebooks/workbook will help as they take you through the concepts gradually. Ask for help if you need it.*

## Grade G: Short Assessed Programming Exercise 2, Input, Calculation and Variables

**To pass this exercise you must demonstrate that you can do the following in writing the program**

- ❑ **Write a literate version of the code.**
- ❑ **EXPLAIN HOW YOUR PROGRAM WORKS**
- ❑ **Use input instructions**
- ❑ **Use assignment and variables**
- ❑ **Use final variables to store literal values rather than values appearing through the code.**
- ❑ **Define all variables inside methods**
- ❑ **Use arithmetic expressions**
- ❑ **Split code in to methods at least one of which takes arguments and returns a result**
- ❑ **Make simple use of comments – at least the author's name, student number and date and an explanation of what the program does overall**

*When your program works, test it and check it ticks all the above boxes then have it assessed by a demonstrator. If you are having problems or don't understand the requirements ask for help.*

**Women's World Cup Statistics** Write a program that works out statistics for the women's world cup. It should ask for the total matches played so far, the total goals scored so far and then give two statistics: the goals scored per match and the number of minutes per goal (assuming each match is 90 minutes long).

The number of goals scored per match is calculated by dividing the total goals scored by the total number of matches and rounding the result DOWN.

The minutes per goal is calculated by dividing the total number of minutes (matches x 90) by the total number of goals scored and rounding the result to one decimal place. Rounding to one decimal place should be done by multiplying the number by 10, rounding that number to the nearest integer using the Math.round() method and then dividing the result by 10.

The number of minutes in a match (90) MUST be stored in the program as a local final variable. The program MUST be structured as a series of methods where the first method calls a series of others in turn. They will either get values from the user and return results to the first method, or take arguments and return results and/or print results. The first method uses those returned values to pass arguments to the next method.

The output should be exactly as follows (if the values given in bold are typed by the user).

An example run:

```
How many matches have been played so far? 6
How many goals have been scored so far? 19
Goals per match =   3
Minutes per goal =  28.4
```

Another example run:

```
How many matches have been played so far? 63
How many goals have been scored so far? 164
Goals per match =   2
Minutes per goal =  34.6
```

Modify the example program (PC-ShortAssessedExercise2.ipynb) to do this new task. It shows how to share values between methods - call methods with arguments and return values, as needed here. Find it on JHUB in the MY ASSESSED PROGRAMS DIRECTORY.

## TESTING Short Assessed Programming Exercise 2, Input, Calculation and Variables

**You should check as a minimum that it meets the tick boxes at the top of the page and:**

- ❑ **Each method works correctly on its own with different arguments.**
- ❑ **The program runs correctly for all input of the right type (not just the examples above) including extreme values. At this stage it can crash if input is of the wrong type but document this.**
- ❑ **The program should clearly indicate what should be typed when the user is needed to enter values (including indicating values not acceptable).**

This program is now more complicated. You cannot just run it once and see if it works as what it does depends on the values input. You will need to run it lots of times – enough to convince yourself it always works. What about extreme values – very big or very small? Zero is always a good value to test for. For this exercise, if the program crashes when bad values are entered that is ok as long as the user was warned in some way not to type those values beforehand. Check the correct answer is given. Check carefully there are no missing spaces or punctuation in the output. Inspecting your code by eye, including checking all variables are given a value before the value is used, is always a good idea. Check the comments make sense.

## Grade F: Short Assessed Programming Exercise 3: Making Decisions

**To pass this exercise you must demonstrate that you can do the following in writing the program**

- ❑ **Write a literate version of the code**
- ❑ **EXPLAIN HOW YOUR PROGRAM WORKS**
- ❑ **Use if-then-else statements.**
- ❑ **Split code in to methods and includes at least one method that takes one or more arguments and (others that) return a result**
- ❑ **Use final variables to store literal values rather than values appearing through the code.**
- ❑ **Define all variables inside methods**
- ❑ **Includes useful comments, at least one per method saying what it does.**
- ❑ **Use indentation in a way that makes the program's structure clear.**

*When your program works, test it and check it ticks all the above boxes then have it assessed by a demonstrator. If you are having problems or don't understand the requirements ask for help.*

**Best Test Grades** Students do two tests with three parts each (theory, dry run and program). For each part they get the best grade achieved from the two tests. Write a program that works out their final three grades given the six separate grades. To enter a grade, the user types a number from a menu: that lists the options as below 1) A+ 2) A 3) B 4) C 5) D 6) F 7) G.

Your program MUST include methods including, at least, ones that

- asks for and returns a grade.
- is given as arguments a pair of grades and returns the best
- is given the best grade for each part of the test and prints them
- a method that calls these methods in sequence.

An example run of the program (words in **bold** are typed by the user):

```
Test 1: What is your 1st theory grade?  1)A+  2)A  3)B  4)C  5)D  6)F  7)G 4
Test 1: What is your 1st dry run grade? 1)A+  2)A  3)B  4)C  5)D  6)F  7)G 2
Test 1: What is your 1st program grade? 1)A+  2)A  3)B  4)C  5)D  6)F  7)G 6
Your grades were C A F
Test 2: What is your 2nd theory grade?  1)A+  2)A  3)B  4)C  5)D  6)F  7)G 4
Test 2: What is your 2nd dry run grade? 1)A+  2)A  3)B  4)C  5)D  6)F  7)G 3
Test 2: What is your 2nd program grade? 1)A+  2)A  3)B  4)C  5)D  6)F  7)G 5
Your grades were C B D
Your final three test grades are Theory: C, Dry Run: A, Program: D
```

Another example run:

```
Test 1: What is your 1st theory grade?  1)A+  2)A  3)B  4)C  5)D  6)F  7)G 5
Test 1: What is your 1st dry run grade? 1)A+  2)A  3)B  4)C  5)D  6)F  7)G 7
Test 1: What is your 1st program grade? 1)A+  2)A  3)B  4)C  5)D  6)F  7)G 2
Your grades were D G A
Test 2: What is your 2nd theory grade?  1)A+  2)A  3)B  4)C  5)D  6)F  7)G 1
Test 2: What is your 2nd dry run grade? 1)A+  2)A  3)B  4)C  5)D  6)F  7)G 7
Test 2: What is your 2nd program grade? 1)A+  2)A  3)B  4)C  5)D  6)F  7)G 3
Your grades were A+ G B
Your final three test grades are Theory: A+, Dry Run: G, Program: A
```

## TESTING Short Assessed Programming Exercise 3: Making Decisions

**You should check as a minimum that it meets the tick boxes at the top of the page and:**

- ❑ **The program runs correctly for all input of the right type including unknown values.**
- ❑ **All methods individually work correctly / return the correct result**
- ❑ **All branches of the IF-THEN-ELSE statement work**
- ❑ **The program deals with input it doesn't know about**

Decision statements add a new level of difficulty for testing. You need to make sure every line works, but when you run the code some of the lines are not executed – as in any if statement either the then case or the else case is executed not both. You must test the program by running it lots of times with different values, choosing values that will definitely test all the lines of code (**so test all branches**) between them. Also remember to check carefully the output is right (spaces, punctuation, etc). By breaking the program in to methods you can test each method separately (as you write it). Testing is easier if you write a test plan (or test method), testing methods as you write them. It is a good idea to inspect the code by eye too. This is called doing a "Programmer's Walkthrough", "tracing the code" or "dry running". Does it appear to do the right thing stepping through the execution on paper?

## Grade D: Short Assessed Programming Exercise 4: For Loops

**To pass this exercise you must demonstrate that you can do the following in writing the program**

- ❑ **Write a literate version of the code**
- ❑ **EXPLAIN HOW YOUR PROGRAM WORKS**
- ❑ **Use counter controlled FOR loop statements.**
- ❑ **Includes at least one method that take argument(s) and returns a result**
- ❑ **Has useful comments, at least one per method saying what it does.**
- ❑ **Use indentation in a way that makes its structure clear.**
- ❑ **Use final variables to store literal values rather than values appearing through the code.**
- ❑ **Defines all variables inside methods**
- ❑ **Use variable names that give an indication of their use.**

*When your program works, test it and check it ticks all the above boxes then have it assessed by a demonstrator. If you are having problems or don't understand the requirements ask for help.*

**River Pollution count** Write a program that collects information about the level of pollution in rivers. This is done by the monitor taking 5 samples at different times during the day measured as a whole number of "colony forming units of bacteria" (CFU) per 100 ml of water.

The program should use a for loop to ask for the 5 values as a whole number and then report the highest of those values and the average. Based on the highest value it should then indicate whether this is GOOD water quality (<= 200 CFU/100ml); SUFFICIENT water quality (<= 400 CFU/100ml) or POOR water quality (not meeting the sufficient criteria). The program should NOT use an array or other similar data structure.

A separate method should be written that given a total and a number of readings computes the average, rounded to the nearest integer. You can round a number to the nearest integer using the Math.round() method.

If the data as below is input your program should print exactly as in the example run of the program (**bold** words are typed by the user):

```
Sample 1: What was the water quality measurement in CFU/ml? 130
Sample 2: What was the water quality measurement in CFU/ml? 225
Sample 3: What was the water quality measurement in CFU/ml? 330
Sample 4: What was the water quality measurement in CFU/ml? 111
Sample 5: What was the water quality measurement in CFU/ml? 92
The worst water quality today was 330 CFU/ml.
The average water quality today was 177 CFU/ml.
The water quality is SUFFICIENT today.
```

HINT: Keep the greatest value found so far in a variable. Keep the total so far in another variable.

## TESTING Short Assessed Programming Exercise 4: For Loops

**You should check as a minimum that it meets the tick boxes at the top of the page and:**

- ❑ **The program runs correctly for all input.**
- ❑ **All methods return the correct result**
- ❑ **The program always does the correct number of iterations**
- ❑ **The program deals with appropriately with bad input**
- ❑ **Running totals are correct at all points through the loop**

As with the *if* statements, loops execute code depending on a test. You need to check that the loop does execute exactly the right number of times every time. Doing dry run/programmer's walkthroughs can be especially important. Make sure loops can't run forever for any input (including input the programmer didn't think of the user ever entering such as empty strings, zero and negative numbers). A neat little debugging hack is to stick print statements in the code, so that when you run it, you get some feedback on what the code is doing. Perhaps "This program woz ere" or even "This is the <*i*> th time through this loop", etc..

## Grade C: Short Assessed Programming Exercise 5: Arrays

**To pass this exercise you must demonstrate that you can do the following in writing the program**

- ❑ **Write a literate version of the code**
- ❑ **EXPLAIN HOW YOUR PROGRAM WORKS**
- ❑ **Use arrays manipulated by loops.**
- ❑ **Includes methods that take multiple arguments including array arguments.**
- ❑ **Use indention in a way that makes structure clear**
- ❑ **Contains helpful comments.**
- ❑ **Use variable names that give an indication of their use.**
- ❑ **Use final variables to store literal values rather than them appearing through the code.**
- ❑ **All variables are defined inside methods**

### Overall module grade calculation

Write a program that collects information about the grades a student got using a for loop, and tells them their final grade. They should input the number of times they earned each grade storing the numbers in an array. From these numbers the program should calculate how many of "each grade or better" they earned by adding the numbers for that grade with the numbers for higher grades, again storing the results in an array. The program then tells them their final grade by checking first if they have gained an A* and if not have they gained an A++, and if not have they gained an A+ and so on. This is done as follows:

- To get an A* they need all 8 grades to be A+.
- To get an A++ they need at least 7 of the grades to be an A+ or better and all 8 to be an A or better.
- To get an A+ they need at least 6 of the grades to be an A+ or better and all 8 to be an A or better.
- To get an A they need at least 6 of the grades to be an A or better and all 8 to be a B or better.
- To get a B they need at least 6 of the grades to be a B or better and all 8 to be a C or better.
- To get a C they need at least 6 of the grades to be a C or better and all 8 to be a D or better.
- To get a D they need at least 6 of the grades to be a D or better and all 8 to be an F or better.
- To get an F they need at least 6 of the grades to be a D or better and all 8 to be a G or better.
- To get a G they need all 8 grades to be a G or better. Otherwise they get a Q grade.

HINT: Keep an array of Strings that stores the different grades ( "A+", "A", "B" etc) with a for loop looking up in it each grade to print next when asking for the grades.

An example run of the program (numbers in **bold** are typed by the user):

```
How many A+ grades did you get? 1
How many A grades did you get? 4
How many B grades did you get? 1
How many C grades did you get? 2
How many D grades did you get? 0
How many F grades did you get? 0
How many G grades did you get? 0
You consistently gained a B grade or better.
Therefore you gained a B grade overall.
```

Another example run:

```
How many A+ grades did you get? 0
How many A grades did you get? 1
How many B grades did you get? 0
How many C grades did you get? 3
How many D grades did you get? 2
How many F grades did you get? 2
How many G grades did you get? 0
You consistently gained a D grade or better.
Therefore you gained a D grade overall.
```

## TESTING Short Assessed Programming Exercise 5: Arrays

**You should check as a minimum that it meets the tick boxes at the top of the page and:**

- ❑ **The program runs correctly for all input.**
- ❑ **No out of bound errors through the way the loop processes the array**

Arrays are a common source of errors. The points about loops apply, but also always check the code cannot run off the end of the array. eg. Make sure it cannot visit non-existent position 5 in an array of length 5 for example: remember it starts counting positions from 0! Check position 0 is used correctly too.

## Grade B: Short Assessed Programming Exercise 6: While Loops

**To pass this exercise you must demonstrate that you are able to do the following in solving the problem**

- ❑ **Write a literate version of the code**
- ❑ **EXPLAIN HOW YOUR PROGRAM WORKS**
- ❑ **Use WHILE loops. They must not have as condition a literal true value.**
- ❑ **Is decomposed into methods that take arguments and return results.**
- ❑ **Use indentation well.**
- ❑ **Has useful comments, at least one per method saying what it does.**
- ❑ **All variable names give an indication of their use.**
- ❑ **All variables have minimal scope and final variables used for literal constants**

**Lap times** Write a program that is to be used to record lap times for a runner when she trains. She runs the same forest circuit each day but runs for different lengths of time and so a different number of laps of that circuit each day. For each lap she does do in a session she wants to record the lap time in seconds and be told the total time. She is also aiming for consistency so wants to keep track of the difference in time between consecutive laps.

The program should use a while loop to repeatedly ask the runner to give the lap times. The program should stop when the special code XXX is entered instead of a lap time. The program should then give the total number of laps completed and the total time in seconds. For example, one run might be as follows.

```
Training Run Data
-----------------
What was lap time 1 (in s)? 1243
What was lap time 2 (in s)? 1245
Difference 2 seconds
What was lap time 3 (in s)? 1220
Difference 25 seconds
What was lap time 4 (in s)? 1227
Difference 7 seconds
What was lap time 5 (in s)? XXX
You did 4 laps. Your total time today was 4935s
```

Make sure you comment your program with comments that give useful information, use indentation consistently and that your variable names convey useful information.

## TESTING Short Assessed Programming Exercise 6: While Loops

**You should check as a minimum that it meets the tick boxes at the top of the page and:**

- ❑ **The program runs correctly for all input.**
- ❑ **The loop is entered correctly the first time.**
- ❑ **The loop terminates correctly even if terminated immediately.**

When the number of times round the loop can vary, use test input values that check it for different numbers of iterations (times round the loop). Odd cases to check are programmers messing up so the loop body always runs just once, or no times at all. Make sure it works if the user ends the program immediately. Doing dry run/programmer's walkthroughs can be especially important. Make sure loops can't run forever for any input including odd values input.

You can see what is happening inside a loop by adding an extra print statement in the body of the loop eg System.out.println("IN THE LOOP") so you can see it is entering the loop. The number of times that message is printed also shows you how many times the loop body repeats. Remember to delete or comment out such test statements from the final version!

**Remember, programming can be tricky at first, but …**

**you are capable of learning to program if you keep at it.**

## Grade A: Short Assessed Programming Exercise 7: Records

**To pass this exercise you must demonstrate that you can do the following in writing the program**

- ❑ **Write a literate version of the code**
- ❑ **EXPLAIN HOW YOUR PROGRAM WORKS**
- ❑ **Create user-defined types using records as classes with no methods.**
- ❑ **Include a method to create such a record**
- ❑ **Check and print information stored in records**
- ❑ **Has at least one method that take argument(s) and returns a result**
- ❑ **If given invalid data, it repeatedly asks for a new value using a while loop.**
- ❑ **Good indentation, comments and variable/method naming.**
- ❑ **All variables have minimal scope and final variables used for literal constants**

**Election Results Information** Write a program that inputs and then prints information about the result of an election where two candidates stand. The program should ask for and be given information about each candidate as below, each stored in a separate record. It should then print out who won and their majority.

A new type should be created (as a record type by creating a class with no methods). Each separate piece of information about an election candidate should be stored in a different field of the record (their name; their party and the number of votes casts for them). A final boolean field should hold whether they won or not.

A method should be written that creates records. It should be called with the values to store in a single record as arguments and return a new record containing that information. It should store a default value of false for who won. It should be called twice to create records for the two candidates.

The program should then determine who won, updating the field in the winner's record to indicate they were the winner.

Finally it should print out the details of the winner including their majority, reading the information out from the record. It should print out the majority by reading the votes from each record and calculating the difference. If it is a draw, then a message is just printed saying there will be a new election rather than a result and the winner's field is left unchanged.

If a negative number is input for the number of votes it should repeatedly ask for a new value (using a while loop) until a positive number of votes is input.

An example run of the program (**bold** words are typed by the user):

```
What is the name of the first candidate? Fotherington-Smythe
What party did they stand for? The Greedy Party
How many votes did they gain? 5221

What is the name of the second candidate? Smith
What party did they stand for? The Equality Party
How many votes did they gain? 16457

Smith of The Equality Party is declared the winner with 16457 votes.
They have a majority of 11236
```

## TESTING Short Assessed Programming Exercise 7: Records

**You should check as a minimum that it meets the tick boxes at the top of the page and:**

- ❑ **The program runs correctly for all input.**
- ❑ **All FIELDS of any RECORD are set correctly by the create method**
- ❑ **All FIELDS of any RECORD are accessed correctly.**
- ❑ **The program deals correctly with invalid input / all situations that can occur.**

For programs with records, you need to be sure the fields are both set and accessed correctly. Testing is easier if you write a special test method adding tests for new methods as you write them.

## Grade A+: Short Assessed Programming Exercise 8: ADTs and Accessor Methods

**To pass this exercise you must demonstrate that you are able to do the following in solving the problem**

- ❑ **Write a literate version of the code**
- ❑ **EXPLAIN HOW YOUR PROGRAM WORKS**
- ❑ **Define and use an abstract data type with record fields accessed ONLY by procedural programming style accessor methods including a create method, all defined in the main class.**
- ❑ **Decomposed fully into multiple methods, call them with multiple arguments and return values from them**
- ❑ **All methods individually and helpfully commented on their use (not over-commenting).**
- ❑ **Good indentation, comments and variable/method naming.**
- ❑ **All variables have minimal scope and final variables used for literal constants**

**Disability Rating for restaurants** A program is needed to rate the disability access rating of restaurants. A score of 1-3 is given for each of step free access; disabled toilets and disabled parking. Total scores of 9 lead to a rating of OUTSTANDING, a score of over 5 leads to GOOD and 5 or less is POOR. If a restaurant has not been evaluated it has a total of 0 and rating of UNRATED. Write a program that repeatedly asks for the three scores and prints the rating of restaurants given the scores.

A new record type (**as a class with no methods**) should be created that is used as an abstract data type. Information about a restaurant should be stored in fields of the record: fields for the name; step-free access score; toilet score; parking score; total score; and a calculated rating of OUTSTANDING, GOOD or POOR.

A create method should be provided to create initial unrated records and set the restaurant name (passed as an argument to the method). The initial scores should be 0 (meaning it is UNRATED).

A second method should be provided to allow the user to set the rating (by setting the three scores provided as arguments). It should calculate the total score and set the rating based on the scores provided. It should not accept invalid scores (anything other than 1-3). Input of scores should be done independently of this method.

A further methods should be provided to convert the record to a String in the standard form : ”Posh Nosh has a disability rating of GOOD.” Printing ratings should be done independently by calling this method.

No methods should be provided to set or read score fields individually, or to change the rating other than the method as above. In this way, provided the rest of the program only uses these three primitive methods, the total and rating should be guaranteed to always be consistent with the scores.

All methods including these primitive accessor methods must also be defined for the fields in the main class (they MUST NOT be in the newly created record class)**. ONLY the accessor methods should access the record fields directly. All other methods should call the primitive methods to access the record fields.**

The following is an example run of the program (which will continue asking while y is input):

```
What is the name of the restaurant? Posh Nosh
What is the score for step-free access? 2
What is the score for disabled toilets? 3
What is the score for disabled parking? 1

Posh Nosh has a disability rating of GOOD.

Another (y/n)? n
```

## TESTING Short Assessed Programming Exercise 8: Accessor methods

**You should check as a minimum that it meets the tick boxes at the top of the page and:**

- ❑ **The program runs correctly for all input.**
- ❑ **The accessor methods each individually work and return the correct results**
- ❑ **All other methods individually work correctly for all input.**

Methods make the tester’s job easier – a reason for using them! You can test each method separately – make sure it works as it should before worrying about whether the program as a whole does. If methods work then later errors found will be in the code that uses them not the methods. Create a test plan method in your literate document that tests each method printing results returned checking they are correct.

Once you are sure accessor methods work, you can ignore how the record is implemented and treat them as primitives.

## BONUS EXERCISE: Recursion

**This exercise is not assessed - do it (if you finish all other exercises before the end of term) to increase your understanding and prepare for the exam**

**It will not be assessed but in attempting it you will learn how to**

- ❑ **Uses recursion in an appropriate way to solve a recursive problem.**

### Recursive Parsing Calculator

**HINT**: Before attempting this exercise, see the simplerecursiveparser.java in the example programs of the recursion unit on QM+ and the related booklet about Language, grammars and recursion that explains it.

Write a program that recursively parses expressions, input as strings, from the following recursively defined language and calculates and prints out the answer to the calculations.

<EXP> = + <DIGIT> <EXP> | - <DIGIT> <EXP> | &<EXP> | <DIGIT>

<DIGIT> = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Here EXP stands for expressions, + means do addition, - means subtract and & means calculate the sum up to the given number.

Legal expressions in this language involve putting the operator before its arguments (this is called Polish notation). Instead of writing 1+2, in this language you write +12. Notice only single digits are allowed and spaces are not allowed. Further legal expressions can be seen below.

An example run of the program (characters in **bold** are typed in by the user and pop-up boxes may be used for input and output):

```
Please input the expression 3
The answer is 3
```

Another example run:

```
Please input the expression &3
The answer is 6
```

Another example run:

```
Please input the expression &+23
The answer is 15
```

Another example run:

```
Please input the expression &+1-82
The answer is 28
```

Make sure you split the program in to multiple methods, and use a series of recursive methods following the structure of the recursive definition about expressions. You must **not** use an explicit loop at all in your program. Comment your program with useful comments that give useful information, with every method commented with what it does, use indentation consistently and ensure that your variable names convey useful information. Your variables should be positioned so that their scope is small.

## TESTING Short Assessed Programming Exercise 8: Recursion

**You should check as a minimum that it meets the tick boxes at the top of the page and:**

- ❑ **The program runs correctly for all input.**
- ❑ **Each method individually works**
- ❑ **Each recursive method works for both base cases and step cases.**
- ❑ **The program gives a suitable message for different kinds of invalid input**

## 8. Programming mini-project

### This is assessed: counts for 2 A-F grades

Whatever the grade you get for this, **you get that grade twice**. You must write a literate version of a single mini-project procedural Java program in stages over the term. It should demonstrate your understanding of and ability to use the different constructs covered in the module. Possible projects are given on the subsequent pages. **You choose ONE of the four projects.** It must be written as a literate program in JHUB but from the D level onwards there should **also** be a local runnable version compiled and run on your desktop.

ALL programs must be written in Java in a procedural programming style as covered in the lectures. There must be ONLY one class containing methods (any other classes are just data structures and contain NO methods at all.)

You **MUST** develop your program in stages – modifying your earlier versions (that have already been marked and achieved a given level of proficiency) for the next level version. This is an important form of program development for you to learn, used in industry. It is also vital you gain the skills of modifying code and understand how important it is to program in a way that makes modification easy. Once your program has reached a level (see below) you should get it marked by a demonstrator. If you are confident then you can skip getting some levels marked separately by the demonstrators. Earlier level criteria must still be met and will be checked when the later one is.

**Start early!**

For your grade to count, your mini-project **MUST** be marked in the labs BEFORE it is submitted:

- **3 times at 3 different levels overall and**
- **in 3 different weeks' labs through term.**

The level achieved **MUST** be confirmed on your signature sheet.

**Different demonstrators can mark the different levels. For each level you will be required to state what your program does, the grade that you believe the program should obtain and also explain why it deserves that grade**.

As with the short programs you may be asked questions on how it works etc – the mark is for you convincing the demonstrator that you have met the learning outcomes including that you understand the code, not just for presenting a correct program. Your literate program document should justify that it meets the criteria (see examples on JHUB).

At the end of term you must submit the three distinct versions that you had successfully marked (ie each at a different level). If you had more than three programs marked, then submit the three at the highest levels. ***Whole programs submitted at the end of the term for which earlier versions have not previously been marked will not be marked beyond a bare pass level.***

### Example mini-projects (choose one)

The following are example programs with indicative grades for different levels of development. You must choose one of these four topics. However, your program does not have to do exactly as outlined in the step-by-step examples for each level as long as it fulfils the overall description: use your imagination (though obey the specific restrictions so all the boxes can be ticked)! What matters is that you demonstrate you can use the different programming constructs like loops, arrays, abstract data types, etc., and have achieved all the other criteria for a given level as described, rather than that your program does the precise thing in the example.

All students' mini-project programs should be different in what they do and how they do it.

What matters is how many criteria you have achieved in your program. A program might drop a grade or more if, for example, it does not use methods appropriately, is not commented correctly, or is not indented.

**!!!YOU MUST HAVE YOUR MINIPROJECT ASSESSED 3 TIMES IN 3 LABS AT AT LEAST 3 LEVELS!!!**

## Suggestion 1: Short answer quiz program

Write a quiz **procedural** program where answers must be typed in by the player (it MUST NOT be multiple-choice). The first question is worth one point. When the player gets an answer right, the points for the next question increase. However, if they get a question wrong the next question drops back to being worth only 1 point.

**Level 1 – G minus: Getting started: Input-Output**

- ❑ Is a **literate program**
- ❑ Includes **Screen output, Keyboard input**
- ❑ Defines methods doing a well-defined task and uses a sequence of **method calls**
- ❑ Comment gives at least author's name and student number at start of program.

**Example (one way to achieve the above)**: A single outer method calls a sequence of other methods to get things done (as follows). It first calls a method that prints to the screen the rules. The outer method then calls a method that prints out a question and allows the person to type their answer. The program then calls a method tells them the correct answer.

**Level 2 – G: Making progress: Assignments and Expressions**

- ❑ All the constructs, style and features above AND
- ❑ Includes **variables, assignment and expressions**
- ❑ Defines and uses **at least one method that returns a result**
- ❑ Indentation attempted (it may be inconsistently applied)
- ❑ Comments gives at least authors name, student number and what the program as a whole does.
- ❑ All variables are defined inside methods

**Example (one way to achieve the above)**: As above, but a separate question method is called that asks the question and returns the person's answer back to the outer method, where it is printed eg "You chose Queen". A variable holds the points for this question. The person is told the points score they gained (in this version always 1 point as it is the first question). The variable is updated to hold the points that the next question will be worth (assuming they got it right), and a message is printed saying how much the next question is worth.

**Level 3 – F: Getting there: Decisions**

- ❑ All the constructs, style and features above AND
- ❑ Includes **variables, assignment and expressions**
- ❑ Includes **Decision statements**.
- ❑ Defines and uses **a method that take argument(s)**
- ❑ Indentation attempted (it may be inconsistently applied)
- ❑ Comments gives at least authors name and student number what the program as a whole does.
- ❑ All variables are defined inside methods

**Example (one way to achieve the above)**: As above, but the person is now told if they got the answer right or not. If they got it wrong the points for the next question are set back to 1. They are increased if they got the question right. The score awarded for this question is added to a total variable. A separate marking method is given their answer and the correct answer as arguments and returns a boolean result indicating if it was correct or not. Their score is updated based on this by the outer method that calls the marking method.

**Level 4 - D: Bare Pass: Loops**

- ❑ All the constructs, style and features above AND
- ❑ Includes **Loops**
- ❑ **Well-structured** into multiple **methods** that both take arguments and return results
- ❑ Comments included are helpful, and **each method has a comment** stating what it does.
- ❑ Some use of well chosen variable names which give some information about use
- ❑ All variables are defined inside methods
- ❑ Final variables are used for literal constants.
- ❑ Indentation is good making structure of program clear.

**Example (one way to achieve the above)**: As above, but now the program uses a loop to ask the same question to more than one person in turn and calculates, then prints, a score for each.

**Continued overleaf**

**!!!YOU MUST HAVE YOUR MINI-PROJECT ASSESSED 3 TIMES IN 3 LABS AT AT LEAST 3 LEVELS!!!**

**Level 5 - C: Pass: Arrays and Loops inside loops**

- ❑ All the constructs, style and features above AND
- ❑ Includes **Arrays, accessed with a loop**
- ❑ Includes **Loops within loops**.
- ❑ Defines and uses **methods** including at least one that is passed and uses **array arguments**
- ❑ Methods individually commented about what they do and all clearly indented.
- ❑ Variable declarations are within blocks to reduce scope to a minimum.
- ❑ Consistent use of well chosen variable names that give a clear indication of their use (perhaps making comments about them redundant). Consistent use of final variables for literal constants.

**Example (one way to achieve the above)**: As above but the program now asks each person in turn further questions with the points for each question increasing after each correct answer. Each player's score and the points they will get for their next question are held in an array. The arrays are passed as an argument to methods that need them.

**Level 6 - B: Satisfactory: Records**

- ❑ All the constructs, style and features above AND
- ❑ Includes **records defined as a special class with no methods just field definitions**.
- ❑ Excellent style over comments, indentation, variable usage, final variables, etc.
- ❑ Clearly decomposed into multiple small methods doing distinct jobs that take arguments / return results

**Example (one way to achieve the above)**: As above but now a new record type is created to hold all information about a question (with fields for the question itself, and the correct answer). One record is created for each question and they are all stored in an array of records (so the first question record is stored in position 0 of the array, the second question record is stored in position 1 of the array, etc). Information about the question is pulled from the appropriate record when needed using the question number to look up the correct record.

**Level 7 - A: Merit: File I/O**

- ❑ All the constructs, style and features above AND
- ❑ **BOTH file input AND file output**
- ❑ Excellent style over comments, indentation, variable usage, final variables etc.
- ❑ Program is fully composed into methods so excellent use of methods throughout
- ❑ All variables are defined in methods and have minimum scope.

**Example (one way to achieve the above)**: As above, but provides an option to create new sets of questions that can be saved to different files. The user is asked which set they wish to use at the start and it is read in from the given file. Design a suitable file format to allow this to be done easily.

**Level 8 – A+: Distinction: ADTs**

- ❑ All the constructs, style and features above AND
- ❑ Includes procedural programming style **abstract data type** with a clearly commented/specified set of operations, defines and uses **accessor methods** (all defined in the main class) to access records.

**Example (one way to achieve the above)**: As above, but now, the records for questions are ONLY accessed via accessor methods including a create method (the only use of the dot notation is in accessor methods). The accessor methods are part of an Abstract Data Type with clearly specified primitive operations: only operations that are legal on a question record should be implemented. The ADT is clearly documented in a comment with the ADT methods.

**And then …**

Keep Learning! Now carry on and make your program really **something special**, using other more advanced constructs or algorithm from the course, or something you have read up on yourself. For example, use recursive methods in sensible ways or include search and sorting methods. Restructure it to use more abstract data types including a high score table. Make it a program someone would really want to use!

Use it as an excuse to learn more.

**!!!YOU MUST HAVE YOUR MINI-PROJECT ASSESSED 3 TIMES IN 3 LABS AT AT LEAST 3 LEVELS!!!**

## Suggestion 2: Pet program

Write a **procedural program** that simulates pets that the player must look after.

### Level 1 – G minus: Getting started: Input-Output

- ❑ Is a **literate program**
- ❑ Includes **Screen output, Keyboard input**
- ❑ Defines methods doing a well-defined task and uses a sequence of **method calls**
- ❑ Comment gives at least author's name and student number at start of program.

**Example (one way to achieve the above)**: A single outer method calls a sequence of other methods (as follows) to get things done. It first calls a method that prints to the screen what the program is for (to look after pets etc). The outer method then calls a method that asks you to name your pet. It then prints a message using the name given (eg "Happy 0th Birthday Aled the Anteater.", etc.

### Level 2 – G: Making progress: Assignments and Expressions

- ❑ All the constructs, style and features above AND
- ❑ Includes **variables, assignment and expressions**
- ❑ Defines and uses **at least one method that returns a result**
- ❑ Indentation attempted (it may be inconsistently applied)
- ❑ Comments gives at least authors name and student number what the program as a whole does.
- ❑ All variables are defined inside methods

**Example (one way to achieve the above)**: As above, but now a separate method asks the user for the pet name and returns it to be used by the outer calling method. The pet can be hungry. It starts with a given hunger score from 1 to 5. A message is printed giving the number (eg "On a scale of 1-5, Aled's hunger rates 3"). The score is either increased or decreased (at random).

### Level 3 – F: Getting there: Decisions

- ❑ All the constructs, style and features above AND
- ❑ Includes **Decision statements**.
- ❑ Defines and uses **a method that take argument(s)**
- ❑ Indentation attempted (it may be inconsistently applied)
- ❑ Comments give at least authors name and what the program as a whole does.
- ❑ All variables are defined inside methods

**Example (one way to achieve the above)**: As above, but the hunger number is printed in words not numbers (eg 1 means "ravenous", 2 means "famished", … 5 means "full"). The method that prints the state of the pet is now given the hunger level as an argument and prints a message describing it in words ("He is ravenous").

### Level 4 - D: Bare Pass: Loops

- ❑ All the constructs, style and features above AND
- ❑ Includes **Loops**
- ❑ **Well-structured** in to multiple **methods** that both take arguments and return results
- ❑ Comments included are helpful, and **each method has a comment** stating what it does.
- ❑ Some use of well chosen variable names which give some information about use
- ❑ All variables are defined inside methods
- ❑ Final variables are used for literal constants.
- ❑ Indentation is good making structure of program clear.

**Example (one way to achieve the above)**: As above, but the program uses a loop to allow a series of rounds to be played. On each round the hunger level of the pet is changed randomly (up or down). (HINT: see the module workbook part 1 for how to roll a virtual dice!) The pet also has a health level from 0-8. If the hunger level drops to 2 or less the health level is reduced. On each round the player chooses which of several ways to look after the pet in that round (eg feed it, hug it, give it medicine, etc). These reduce the hunger and health scores. The player wins the game if their pet survives a given number of rounds without dying (due to the health level dropping to 0).

**Continued overleaf**

**!!!YOU MUST HAVE YOUR MINI-PROJECT ASSESSED 3 TIMES IN 3 LABS AT AT LEAST 3 LEVELS!!!**

## Level 5 - C: Pass: Arrays and Loops inside loops

- ❑ All the constructs, style and features above AND
- ❑ Includes **Arrays, accessed with a loop**
- ❑ Includes **Loops within loops**.
- ❑ Defines and uses **methods** including at least one that is passed and uses **array arguments**
- ❑ Methods individually commented about what they do and all clearly indented.
- ❑ Variable declarations are within blocks to reduce scope to a minimum.
- ❑ Consistent use of well chosen variable names that give a clear indication of their use (perhaps making comments about them redundant). Consistent use of final variables for literal constants.

**Example (one way to achieve the above)**: As above but the program now stores the words that describe each hunger level in an array where it is looked up based on the hunger level. If the choice of action the player types is unknown then the program repeatedly (using a while loop) asks them to try again until they type a legal command.

## Level 6 - B: Satisfactory: Records

- ❑ All the constructs, style and features above AND
- ❑ Includes **records defined as a special class with no methods just field definitions**.
- ❑ Excellent style over comments, indentation, variable usage, final variables etc.
- ❑ Clearly decomposed into multiple small methods doing distinct jobs that take arguments / return results

**Example (one way to achieve the above)**:

As above but the program now allows multiple pets to be created with the details of each stored in a record (with fields for its name, health and hunger levels as well as other new information about the pet such as its favourite food, etc). The records are stored in a single array. On each round for each player, all the pets hunger values are randomly chosen to be changed or not. The details of all are printed to allow the player to decide what is best to do. The arrays are passed to methods that use them.

## Level 7 - A: Merit: File I/O

- ❑ All the constructs, style and features above AND
- ❑ **BOTH file input AND file output.**
- ❑ Excellent style over comments, indentation, variable usage, final variables, etc.
- ❑ Excellent use of methods throughout
- ❑ All variables are defined in methods and have minimum scope.

**Example (one way to achieve the above)**: As above, but now allows the current pet world state (ie the round number, and all the details of arrays of records) to be saved into a file so the program can be quit and continued later from where the player stopped, without having to start again from the beginning. Design a suitable file format to allow this to be done easily.

## Level 8 – A+: Distinction: ADTs

- ❑ All the constructs, style and features above AND
- ❑ Includes procedural programming style **abstract data type** with a clearly commented/specified set of operations, defines and uses **accessor methods** (all defined in the main class) to access records.

**Example (one way to achieve the above)**: As above, but now, the records for pet records are ONLY accessed via accessor methods including a create method (the only use of the dot notation is now in accessor methods). The accessor methods are part of an Abstract Data Type with clearly specified primitive operations: only operations that are legal on a pet record should be implemented. The ADT is clearly documented in a comment with the ADT methods.

## And then …

Keep Learning! Now carry on and make your program really **something special**, using other more advanced constructs or algorithm from the course, or something you have read up on yourself. For example, use recursive methods in sensible ways or add search and sorting methods. Allow multiple people to play. Restructure it to use more abstract data types including a high score table. Make it a program someone would really want to use!

Use it as an excuse to learn more.

**!!!YOU MUST HAVE YOUR MINI-PROJECT ASSESSED 3 TIMES IN 3 LABS AT AT LEAST 3 LEVELS!!!**

## Suggestion 3: Fantasy Football League Game

Write a **procedural program** that is a game based on a Fantasy Football game.

### Level 1 – G minus: Getting started: Input-Output

- ❑ Is a **literate program**
- ❑ Includes **Screen output, Keyboard input**
- ❑ Defines methods doing a well-defined task and uses a sequence of **method calls**
- ❑ Comment gives at least author's name and student number at start of program.

**Example (one way to achieve the above)**: A single outer method calls a sequence of other methods (as follows) to get things done. It first calls a method that introduces the game. It then prints a question asking which country's team the person wish to be and waits for a country to be typed. The named country is then printed as part of a message. The outer method then calls a method that prints the team to be played (always the same in this early version).

### Level 2 – G: Making progress: Assignments and Expressions

- ❑ All the constructs, style and features above AND
- ❑ Includes **variables, assignment and expressions**
- ❑ Defines and uses **at least one method that returns a result**
- ❑ Indentation attempted (it may be inconsistently applied)
- ❑ Comments gives at least authors name and student number what the program as a whole does.
- ❑ All variables are defined inside methods

**Example (one way to achieve the above)**: As above, but the program now stores the number of goals currently scored by each team in separate variables. A method that acts as a virtual 4-sided dice is used twice - to determine each team's score. (HINT: see the module workbook part 1 for how to roll a virtual dice!). The score is printed.

### Level 3 – F: Getting there: Decisions

- ❑ All the constructs, style and features above AND
- ❑ Includes **Decision statements**.
- ❑ Defines and uses **a method that take argument(s)**
- ❑ Indentation attempted (it may be inconsistently applied)
- ❑ Comments give at least authors name and what the program as a whole does.
- ❑ All variables are defined inside methods

**Example (one way to achieve the above)**: As above, but now the program prints not just the score but who won (or whether it was a draw). This is determined by a method given the team names and goals scored as arguments.

### Level 4 - D: Bare Pass: Loops

- ❑ All the constructs, style and features above AND
- ❑ Includes **Loops**
- ❑ **Well-structured** in to multiple **methods** that both take arguments and return results
- ❑ Comments included are helpful, and **each method has a comment** stating what it does.
- ❑ Some use of well chosen variable names which give some information about use
- ❑ All variables are defined inside methods
- ❑ Final variables are used for literal constants.
- ❑ Indentation is good making structure of program clear.

**Example (one way to achieve the above)**: As above, but the program uses a loop to play a series of games. The team gains points for each win or draw. After a fixed number of games the total number of points is their final score.

**Continued overleaf**

**!!!YOU MUST HAVE YOUR MINI-PROJECT ASSESSED 3 TIMES IN 3 LABS AT AT LEAST 3 LEVELS!!!**

## Level 5 - C: Pass: Arrays and Loops inside loops

- ❑ All the constructs, style and features above AND
- ❑ Includes **Arrays, accessed with a loop**
- ❑ Includes **Loops within loops**.
- ❑ Defines and uses **methods** including at least one that is passed and uses **array arguments**
- ❑ Methods individually commented about what they do and all clearly indented.
- ❑ Variable declarations are within blocks to reduce scope to a minimum.
- ❑ Consistent use of well chosen variable names that give a clear indication of their use (perhaps making comments about them redundant). Consistent use of final variables for literal constants.

**Example (one way to achieve the above)**: As above but the team play a whole series of different countries. The names of the teams are stored in an array and used in turn based on the number of the game. Instead of just rolling a die to get the number of goals, a series of dice rolls determine events such as goal scored, shot wide, hit the post, goal keeper saved etc. Each side has a roll in turn over a fixed number of rounds and the score changed as goals scored. When the player's team scores a goal, the name of the player who scored it is selected at random from a list stored in an array and printed.

## Level 6 - B: Satisfactory: Records

- ❑ All the constructs, style and features above AND
- ❑ Includes **records defined as a special class with no methods just field definitions**.
- ❑ Excellent style over comments, indentation, variable usage, final variables etc.
- ❑ Clearly decomposed into multiple small methods doing distinct jobs that take arguments / return results

**Example (one way to achieve the above)**: As above but now the program keeps track of the detail of each match. A new record type is created to hold all details about individual matches (with fields for the country played, goals for, goals against, etc). One record is created for each match and a single array of records stores the information about all the matches played. The program prints out a summary of all the matches at the end.

## Level 7 - A: Merit: File I/O

- ❑ All the constructs, style and features above AND
- ❑ **BOTH file input AND file output**
- ❑ Excellent style over comments, indentation, variable usage, final variables, etc.
- ❑ Excellent use of methods throughout
- ❑ All variables are defined in methods and have minimum scope.

**Example (one way to achieve the above)**: As above, but now, the program allows the current game state including details of each match so far played to be saved into a file so the program can be quit and continued later from where the player stopped without having to start again from the beginning. Design a suitable file format to allow this to be done easily.

## Level 8 – A+: Distinction: ADTs

- ❑ All the constructs, style and features above AND
- ❑ Includes procedural programming style **abstract data type** with a clearly commented/specified set of operations, defines and uses **accessor methods** (all defined in the main class) to access records.

**Example (one way to achieve the above)**: As above, but now, the records for matches played are ONLY accessed via accessor methods including a create method (the only use of the dot notation is in accessor methods). The accessor methods are part of an Abstract Data Type with clearly specified primitive operations: only operations that are legal on a match record should be implemented. The ADT is clearly documented in a comment with the ADT methods.

## And then …

Keep Learning! Now carry on and make your program really **something special**, using other more advanced constructs or algorithm from the course, or something you have read up on yourself. For example, use recursive methods in sensible ways or add search and sorting methods. Allow multiple people to play. Restructure it to use more abstract data types. Make it a program someone would really want to use!
Use it as an excuse to learn more.

## Suggestion 4: Explore the forest game

Write a **procedural program** game that allows you to wander through a forest, healing sick animals and collecting plants that improve your healing skills.

### Level 1 – G minus: Getting started: Input-Output

- ❑ Is a **literate program**
- ❑ Includes **Screen output, Keyboard input**
- ❑ Defines methods doing a well-defined task and uses a sequence of **method calls**
- ❑ Comment gives at least author's name and student number at start of program.

**Example (one way to achieve the above)**: A single outer method calls a sequence of other methods (as follows) to get things done. It first calls a method that explains what the game is. The outer method then calls a second method that asks the person for their adventurer's name and prints a message using the name. For example, if they give their name as Dr. Smolder Bravestone, it replies "Dr. Smolder Bravestone, you are in a forest and can see a wounded wolf…".

### Level 2 – G: Making progress: Assignments and Expressions

- ❑ All the constructs, style and features above AND
- ❑ Includes **variables, assignment and expressions**
- ❑ Defines and uses **at least one method that returns a result**
- ❑ Indentation attempted (it may be inconsistently applied)
- ❑ Comments gives at least authors name and student number what the program as a whole does.
- ❑ All variables are defined inside methods

**Example (one way to achieve the above)**: As above, but the adventurer now has "healing points", starting with an initial total of eg 20 points stored in a variable initialised in the main method. They heal the wolf and are deducted two healing points for doing so. A message saying this is printed.

### Level 3 – F: Getting there: Decisions

- ❑ All the constructs, style and features above AND
- ❑ Includes **variables, assignment and expressions**
- ❑ Includes **Decision statements**.
- ❑ Defines and uses **a method that take argument(s)**
- ❑ Indentation attempted (it may be inconsistently applied)
- ❑ Comments gives at least authors name and student number what the program as a whole does.
- ❑ All variables are defined inside methods

**Example (one way to achieve the above)**: As above, but now their name is passed to a method that describes the situation which prints it. A healing method now asks if they wish to heal the animal or not and if so rolls a virtual ten sided dice. (HINT: see the module workbook part 1 for how to roll a virtual dice!) If the total dice score is more than the injury score of the the wolf, set as 4 then it is healed. If the animal is healed then their healing points score goes up by 2. If they fail it goes down by 3. The healing method returns the amount to change the healing points by (either +2 or -3). The main method updates the healing points based on this.

### Level 4 - D: Bare Pass: Loops

- ❑ All the constructs, style and features above AND
- ❑ Includes **Loops**
- ❑ **Well-structured** in to multiple **methods** that both take arguments and return results
- ❑ Comments included are helpful, and **each method has a comment** stating what it does.
- ❑ Some use of well chosen variable names which give some information about use
- ❑ All variables are defined inside methods
- ❑ Final variables are used for literal constants.
- ❑ Indentation is good making structure of program clear.

**Example (one way to achieve the above)**: As above, but the program uses a loop to allow the player multiple attempts to heal the animal until they succeed or run out of healing points. The game finishes when they drop to 0 healing points.

**Continued overleaf**

**!!!YOU MUST HAVE YOUR MINI-PROJECT ASSESSED 3 TIMES IN 3 LABS AT AT LEAST 3 LEVELS!!!**

## Level 5 - C: Pass: Arrays and Loops inside loops

- ❑ All the constructs, style and features above AND
- ❑ Includes **Arrays, accessed with a loop**
- ❑ Includes **Loops within loops**.
- ❑ Defines and uses **methods** including at least one that is passed and uses **array arguments**
- ❑ Methods individually commented about what they do and all clearly indented.
- ❑ Variable declarations are within blocks to reduce scope to a minimum.
- ❑ Consistent use of well chosen variable names that give a clear indication of their use (perhaps making comments about them redundant). Consistent use of final variables for literal constants.

**Example (one way to achieve the above)**: As above but a loop allows a series of rounds to be played. On each round the adventurer travels on and comes across new sick animals to heal. Their final score at the end of the game is the total number of animals they healed. The kind of animals encountered in turn is stored in an array of Strings.

## Level 6 - B: Satisfactory: Records

- ❑ All the constructs, style and features above AND
- ❑ Includes **records defined as a special class with no methods just field definitions**.
- ❑ Excellent style over comments, indentation, variable usage, final variables etc.
- ❑ Clearly decomposed into multiple small methods doing distinct jobs that take arguments / return results

**Example (one way to achieve the above)**: As above but now a new record type is created to hold different information about animals (with fields for the the type of animal, some kind of description eg it is caught on barbed wire, has a broken foot, is bleeding from a shotgun wound, etc, how many healing points are needed to cure it, dice roll needed to heal it, etc). One record is created for each animal. All are now stored in a single array of records that replaces the previous array of names.

## Level 7 - A: Merit: File I/O

- ❑ All the constructs, style and features above AND
- ❑ **BOTH file input AND file output**
- ❑ Excellent style over comments, indentation, variable usage, final variables, etc.
- ❑ Program is fully composed into methods so excellent use of methods throughout
- ❑ All variables are defined in methods and have minimum scope.

**Example (one way to achieve the above)**: As above, but the program now provides an option at the start to create new animal records and save them into a file. Another option allows the details of animals to use to be loaded in from a file. One set might be fantasy animals, another real animals, etc. Design a suitable file format to allow this to be done easily.

## Level 8 – A+: Distinction: ADTs

- ❑ All the constructs, style and features above AND
- ❑ Includes procedural programming style **abstract data type** with a clearly commented/specified set of operations, defines and uses **accessor methods** (all defined in the main class) to access records.

**Example (one way to achieve the above)**: As above, but now, the records for animals are ONLY accessed via accessor methods including a create method (the only use of the dot notation is in these primitive accessor methods). The accessor methods are part of an Abstract Data Type with clearly specified primitive operations: only operations that are legal on an animal record should be implemented. The ADT is clearly documented in a comment with the ADT methods.

## And then …

Keep Learning! Now carry on and make your program really **something special**, using other more advanced constructs or algorithm from the course, or something you have read up on yourself. For example, use recursive methods in sensible ways or search and sorting methods. Restructure it to use more abstract data types. Make it a program someone would really want to use!

Use it as an excuse to learn more.

## 9. Open Book Tests (mid-term and end-term tests)

**The mid-term and end-term tests are assessed and count for 4 A-F grades in total.**

You will be required to take TWO tests through term in addition to the exam after Christmas. They will be open book. This means that you can consult the learning materials of the module during the test. They are a chance to judge your progress and prepare for the exam as well as counting to the coursework.

You must NOT get help from any other people or AI/chatbot programs during the tests (that would be cheating). All your answers (whether explanations or programs) **MUST BE IN YOUR OWN WORDS**. There are in any case no marks for any answers or part answers that are the same as any source whether on the Internet, from a book, the same as another student for whatever reason, or the same as any other material. That includes cut and paste text that is reworded. **Practice writing answers yourself.** Ability to explain is a very important skill for a successful future career.

When writing programs you can use a compiler but note that compilers focus on syntax (eg minor spelling/punctuation mistakes in the code) and when I mark programs I ignore minor syntax errors - the marks are for demonstrating you can solve the problem, can write Java programs that are logically correct and can write readable, well-structured defensive code. Take care not to use up all your time worrying about compiler errors and not solving the problem!

**Do not leave handing work in to the last minute.** All online tests include additional time for handing in already, and if it is handed in even a few seconds late the online university system will apply automatic penalties. Excuses such as your computer crashing or the Internet being slow are not accepted as a reason for work being late - the extra time is already added to allow for that.

Save your work during a test regularly and keep backups.

Make sure you hand in the correct version ie your final version (and not eg a blank empty booklet). DOUBLE CHECK!

### Mid–term test (assessed: best of this and end-term test counts as 4 A-F grades)

You will be required to take a mid term test part way through term in your lab of week 5. Precise details will be released nearer the time. It will consist of exam-style questions covering the first part of the course (**i.e. the mid term test is on Part 1 of the workbook only**). Examples of the kinds of question are included in the module workbook and on the QM+ site with each unit. There is also a revision section for it with past papers and feedback. It will include questions requiring you, for example, to write programs of a similar complexity to the short programming exercises on paper, explain concepts in your own words, analyse what a program fragment does without running it (ie on paper), compare and contrast concepts, etc.  In fact it will test the skills the other coursework and non-assessed exercises are helping you develop.

It is important that you work hard, and revise from week 1 for this as if you do badly then it will pull down your final module mark. However the later test and exam are most critical.

**If you do badly on this do not worry.** As long as you do better on the end term test those better grades will be the ones that count. Work hard between now and the end term test to improve your skills and your understanding especially by practice. Start revising now for the end-term test!

## End-term test (assessed: best of this and mid-term test counts as 4 A-F grades)

You must also take a written end-term test towards the end of term (Likely either in the lecture slot or on the Wednesday of Week 10 but this is still to be confirmed.)

This is NOT in labs. It will be an open book online test that you can take from home if you wish. If you wish a place in college (including a machine) to take it then you will need to book a place. Places with machines will be limited.

You must pass at least part of this to pass the coursework however good your other coursework grades are. Examples of the kinds of questions are included in the module workbook and on the module QM+ site. There is also a revision section for it with past papers and feedback. The mid-term test also gives you an idea of what the end-term test involves, though the end term test covers the first and second part of the course (ie **The end-term test is on Workbook Parts 1 and 2 only**) and so has questions on harder concepts.

There is no second chance for the end term test (so do not miss it), but if you do not pass it but then do well in the exam you can still pull your mark up to a pass or better.

To pass the end-term test you MUST be able to write programs in exam conditions and also clearly demonstrate you understand and can clearly explain programming concepts.

**You must take either the mid-term or end-term test or you will get a fail mark for the coursework.**

**The mid-term and end-term tests will test you have achieved the learning outcomes of the module on that material:**

- write simple programs from scratch
- write larger programs
- work out what a program does without executing it
- debug programs to ensure they work correctly
- explain programming constructs and use programming terminology correctly
- explain and compare & contrast issues related to programming / programming language design

## Revision and Past Papers

There are sections in the module content TAB of the module QM+ page corresponding to the week the tests happen containing past papers with feedback and model answers for each of the mid-term and end-term tests. Do lots of past papers to prepare for them.

Revising well for these tests will mean you will be far better prepared for the final January exam and will find revising for that much easier.

## Missing tests (or exam) or other hard deadlines - Extenuating Circumstances (ECs)

If you miss a test, exam or other coursework deadline with good reason (eg illness) you MUST apply for Extenuating Circumstances (ECs) or you will get 0/Q for the coursework as a whole.

- This must be done VERY SOON after the missed test/deadline. If you submit it too late it will be rejected out of hand.
- You MUST include documentary evidence (so if e.g. ill you need a Drs note) for the date concerned.
- You MUST submit the ECs via the online university system.
- Module leaders and other academics can NOT give extensions or make other allowances for late submission. It can ONLY be done by you applying for, and being awarded, ECs by the EC panel.
  - You will normally be given at most a one week extension
    - ECs are NOT a way to deal with long term illness or disability. Other mechanisms are for that.
- When filling out the form you must choose the right option. The “EXAMINATION” option is ONLY for missing the January exams. The other (probably “IN-CLASS TEST”) option is for all Autumn (so coursework) related ECs.
- Make sure you include the CORRECT date of the missed assessment in your application.
- The way short programs and mini-project work are designed (eg no weekly deadlines, you know the work to be done from day 1, etc) means unless you are ill in the last weeks of term you should not need to apply for ECS for missing weeks of labs and so missing getting work signed off. Flexibility to allow for missed weeks due to illness or other reasons is already built in. If you miss a week’s lab you can already just bring the work to the following lab, add yourself to the queue, and get it assessed and signed off then.
  - If you are ill in the final weeks of term, however, and so miss the final chance to get programs signed off in the lab you WILL need to apply for ECs.
  - If successful you will then be given a week longer to complete the work. In this and only this situation you hand in the work online unmarked. We will then arrange a later time for you to be viva’d based on what you handed in by the deadline. This ONLY applies to those awarded ECS for this purpose.

## Cheating leads to taking an exam alternative assessment

If you are found to have cheated (including assisting others) on ANY part this module’s assessment (however small), you will likely be required to redo the assessed component cheated on as part of the penalty. On this module that will mean taking an alternative assessment for the whole coursework. In all cases this will consist of taking an exam paper at the next possible normal exam opportunity after the decision (normally either January or August). This will likely be combined with an additional penalty as decided by the university authorities such as capped overall marks, lost resit chance, or worse.

## 10. January Final Assessment / “Exam” (Assessed)

**The January exam is 50% of your overall mark. It MUST be passed to pass module.**

You must take an exam in January. We are currently waiting to hear from college whether it will be a physical invigilated or online open book exam. Details nearer the time.

See the advice about open book exams with respect to tests as they apply to the exam too. You can consult the module resources, but it would be cheating to get help of any kind from anyone else during the exam. The extra time is to allow for login / submission problems. DO NOT leave it to the last half hour to submit as if you miss the deadline even by seconds you will not be allowed to submit anything and so will fail. You will likely **access the exam through a special QM+ exam page (not the module QM+ page). EECS teaching services will email you details nearer the time.**

The January exam is combined 50-50 with the overall coursework mark. You must however pass the exam to pass the module. If you do not pass it (get at least 40%) then your overall mark will be capped at a maximum of 39%.

Examples of the kinds of questions are included in the module workbook and on the module QM+ site in a revision section at the end. The coursework tests also give you an idea, though the exam is longer with more questions. The EXAM will assess you on THE WHOLE ECS401 SYLLABUS - all learning outcomes from ALL THREE WORKBOOKS. You must understand all concepts covered on the module and demonstrate your programming and writing skills in exam conditions.

**To pass the exam you MUST be able to write programs, work out what they do, must be able to explain and compare and contrast them in your own words, and must clearly demonstrate you understand and can write about programming constructs, and use terminology correctly.**

The mark for your coursework components (A-F grades) is only combined with the exam result if you pass the exam. If you fail the coursework and so get less than 40% you will also fail the module. If you do not pass the module for whatever reason then you will have to take a resit exam in the late summer.

If you have taken the coursework seriously and done lots of exercises available, practical and theory and not just assessed ones from the start of the year then you will be well-prepared for the exam.

### Revision and Past Papers

A section at the end of the Weekly Work TAB of the module QM+ page contains a section on exam revision advice including past papers with feedback and model answers. Do lots of past papers to prepare for the exam. There will likely be fewer questions this year than in past papers (eg only one program to write).

### Revising for the Resit Exam

Anyone who fails the module (whether they failed due to the exam, coursework or both) gets one more chance. You will need to take a resit exam in the Late Summer Resit period (normally August). If you pass that resit exam (ie gain 40% or higher in it alone) then you pass the module (though with your mark capped at 40%) whatever you did in the original exam and coursework.

If you do fail initially it is very important that you use the 6 months or so you have to the resit exam improving your ability to program. That can only be done with lots of practice (writing programs, dry running programs, debugging programs) as well as making sure you deeply understand the concepts and can explain them well. You will only pass if you can program.

Go back to the earliest unit that you do not fully understand and work forward from there, mastering a unit before moving on to the next unit. When learning programming it is important to understand earlier concepts to have a chance at understanding later ones.

## 11. Penalties

**IF you**

- do not attempt any of the short programs so do not get at least the first signed off, or
- do not attempt the mini-project at all so do not get at least one level signed off, or
- do not attempt at least one of the mid term test and the end term test
- or do not hand in your program signature sheet
    - **THEN** you will get 0 for the whole coursework

**IF you**

- do not complete all the short programming exercises but attempt some or
- do not complete all the mini-project levels but attempt some
    - **THEN** you will get the grade corresponding to those that you do based on the level you achieve. It is not expected everyone will complete all of the levels.

**IF you**

- only get the mini-project marked in two separate weeks and your grade is higher than a D
    - **THEN** your grade for the miniproject will drop by one level

**IF you**

- only get the mini-project marked once / in one week
    - **THEN** the highest grade you will get for it is a D.

**IF you**

- submit the previously marked programs after the deadline or do not submit one of the marked programs as required
    - **THEN** you will lose those grades gained for the missing work

**IF you**

- hand in short programs or mini-project without having had it marked in labs FIRST
    - **THEN** you will not gain anything for the unmarked work (it will not be marked)

**IF you**

- miss a test and then do any retake while waiting for ECs but do not get ECs awarded
    - **THEN** the grades for it will not count

**IF you**

- hand in online mid term or end term test work late
    - **THEN** you will lose a grade for each part late from the highest graded answer increasing to an additional grade with every further 30 minutes late

**FULL NAME:** **STUDENT ID** _ _ _ _ _ _ _ _ _ _

## 12. ECS401 Assessed PROGRAM Progress Sheet

**DETACH THIS SHEET TO HAND IN AT THE END OF TERM…**You must get this signed in the lab each time the demonstrator marks your program and agrees it passes the given level. If you do not have the signature then you have not yet passed that level.

Look after this sheet. Bring it to the labs EVERY WEEK. It is YOUR responsibility.

| Short Level | Grade | SHORTS WORTH DOUBLE | Date and Signature of Assessor | Mini-Proj level | Grade | MINI IS WORTH DOUBLE | Date and Signature of Assessor |
|---|---|---|---|---|---|---|---|
| 1 | G - | Output | | 1 | G - | Input / Output | |
| 2 | G | Input | | 2 | G | Assignment / Expression | |
| 3 | F | Decisions (If statements) | | 3 | F | Decisions (If statements) | |
| 4 | D PASS | For loops | | 4 | D PASS | Loops | |
| 5 | C | Arrays | | 5 | C | Arrays and loops within loops | |
| 6 | B | While loops | | 6 | B | Records | |
| 7 | A | Records | | 7 | A | File Input/ Output | |
| 8 | A+ | ADT | | 8 | A+ | ADT | |

To get a coursework mark for your programming work you must:

1. Have had your programs marked weekly through term in the labs, getting this sheet signed each time.
2. CHECK that every program you think a demonstrator marked as passed has been signed off as passed on this sheet AND
3. Hand in this PAPER signature sheet with physical signatures for all that you have had signed off.

**DEADLINE: The END of YOUR last lab, 12 December 2023**

4. **Hand in on QM+ a pdf copy of EVERY program you had assessed i.e.**
   - a pdf literate program file downloaded from the Interactive Jupyter Notebook of each short programming exercise already assessed by a demonstrator in the labs,
   - a pdf of the **LAST THREE** literate program versions of your mini-project marked as passed downloaded from the Jupyter Notebook already assessed by a demonstrator in the labs
   - You must submit them electronically via QM+ by the deadline below

You MUST get the programs marked in labs as you do them – the demonstrators are only required to mark at most 2 of your programs in each programming lab so do not leave it until the last minute.

**DEADLINE: 23:59 GMT, end of Friday 15 December 2023**

If it is not signed off on this paper or you do not hand the paper in in the last lab then you have NOT got the grade.

If you lose this sheet you will have to get all the signatures re-added. If they are not on our master record you will need to have them remarked