ECS401 Procedural Programming
Assessment Booklet: September 2021

Contents

1. Learning Outcomes

On passing the module you will be able to:

- read programs
- write simple programs from scratch
- write larger programs in steps
- work out what a program does without executing it
- test and debug programs to ensure they work correctly
- explain and compare and contrast programming constructs and discuss issues related to programming

The assessment, consists of AUTUMN coursework (50% of your final mark) and a JANUARY final assessment (50% of your final 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 your assessed work (programs, explanations, test answers) is substantially the same as someone else's (whoever was the original source) THEN you automatically lose the grade.

It is only help if afterwards they can do it themselves without any more help.

You must NOT help others doing an assessed test at all.

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

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

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 answers 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 work and then edit it to pretend it is yours.

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.

3. Overview of Assessment

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.

The coursework consists of:

- A series of short programming exercises (worth 2 A-F grade)
- A programming mini-project (worth 3 A-F grades)
- A single mid term test consisting of:
  - A mid-term theory test (similar question style to final exam) (worth 1 A-F grades)
  - A mid-term dry-run test (also similar to the final exam) (worth 1 A-F grades)
  - A mid-term programming test (also similar to the final exam) (worth 1 A-F grades)
- A single end-term test consisting of:
  - An end-term theory test (also similar to the final exam) (worth 1 A-F grades)
  - An end-term dry run test (also similar to the final exam) (worth 1 A-F grades)
  - An end-term programming test (also similar to the final exam) (worth 1 A-F grades)

YOU MUST PASS THE END-TERM TEST TO PASS THE COURSEWORK

This gives 11 A-F grades, turned in to a single coursework mark at the end of the term (see the next page for the way 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 put in effort to learn from day 1 and throughout term by working hard 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.

*If you do not take one of the above components (eg miss a test, or fail to get any shorts or miniproject marked or submitted) you fail the **WHOLE** coursework (0% overall).*

January Final Assessment (“Exam”)

Aim: The final assessment 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).

What must I do to Pass the Module?

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

4. Calculating your Final Coursework Grade

Your overall grade for the coursework is calculated as follows from the 11 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 component missed completely (no shorts, no mini, no mid-term or no end-term submitted at all)	Grade Q	0%	FAIL
11A+s	Grade A*	100%	PASS
9A+s and 2D (or better)	Grade A++	96%	PASS
7A+s and 1A, 1D and 2 other (or better)	Grade A+	86%	PASS
7As and 1B, 1D and 2 other (or better)	Grade A	76%	PASS
7Bs and 1C and 1D and 2 other (or better)	Grade B	66%	PASS
7Cs and 2D and 2 other (or better)	Grade C	56%	PASS
9Ds and 2 other (or better)	Grade D	46%	PASS
9Ds and 2Fs and 2 other (or better)	Grade F	30%	NOT PASSED
11F-s (or better)	Grade F-	16%	NOT PASSED
Otherwise (including missing component(s))	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 URGENTLY.

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

5. Assessment Deadlines

For the purpose of this module weeks are counted as follows

Wk	**week start - end (mon - fri)**	**Advisory Deadlines** You must do the programs and get them marked week by week (**by online 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.	**Major Deadlines you MUST NOT MISS as you FAIL the module if you do miss them.**
1	27 Sept - 1 Oct		
2	4 Oct - 8 Oct	**Short programming exercise 1** marked no later than YOUR allocated lab	
3	11 Oct - 15 Oct	**Short 2 and Miniproject level 1** marked no later than YOUR allocated lab	
4	18 Oct - 22 Oct	**Short 2 and Miniproject level 2** marked no later than YOUR allocated lab	
5	25 Oct - 29 Oct		**MID-TERM TEST** (online open book) in YOUR timetabled lab
6	1 Nov - 5 Nov	**Short 3 and Miniproject level 3** marked no later than YOUR allocated lab	
7	8 Nov - 12 Nov	**Short 4 and Miniproject level 4** marked no later than YOUR allocated lab	
8	15 Nov - 19 Nov	*DON'T LEAVE IT TOO LATE TO GET PROGRAMS DONE AND MARKED!* At most 2 programs can be marked each week	
9	22 Nov - 26 Nov	At most 2 programs can be marked each week	
10	29 Nov - 3 Dec	At most 2 programs can be marked each week	**END-TERM TEST (online open book) keep free - ALL TAKE IT Monday 11pm-2pm**
11	6 Dec - 10 Dec	At most 2 programs can be marked this week	
12	13 Dec - 17 Dec	At most 2 programs can be marked this week	**YOUR ALLOCATED LAB IS THE FINAL DEADLINE FOR MARKING PROGRAMS.** Hand in previously marked programs (online) and double check all marked in master record

- 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 mark.
- **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 provided this isn't the first time it has been seen)

6. Literate Programming

Programs are not just for computers to read but must also be easy for humans to read.

Programs have to be developed over a period of time, are modified for new purposes, need found bugs fixed, are worked on in teams and so often need to be understood by people other than the original author. Therefore clear 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. 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.** The 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 in the notebook, broken into methods (see below) with appropriate "Markdown" descriptions and test calls
    - A final FULL program version at the end of the notebook with all methods and call.
        - for the miniproject level D on this version should compile and run outside the notebook as a standalone program.

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://jhub.eecs.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
    - Date
    - Version (eg new version after each time it has been assessed if assessed multiple times)
    - An overview of its purpose

7. Short Programming Exercises

This is assessed: counts for 2 A-F grades

- Whatever the grade you get for this, 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 virtual signature in the official online record).

Add a note at the end of your literate document that the program has been signed off, and a second one once you have checked the online confirmation.

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. The Demonstrators are your clients and 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 THIS SERIES OF EXERCISES (SHORT 1-8 and the MINIPROJECT) AND GET THEM MARKED IN ONLINE VIVAS WEEK-BY-WEEK.

Grade F- -: 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

- ❑ **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 and date**
- ❑ **Write a literate version of the code**

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.

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. A blank line should separate the initials. Each initial should 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
```

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

You can modify one of the programs from the course QM+ page or interactive notebooks rather than writing it out from scratch.

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:

- ❑ **The program compiles correctly**
- ❑ **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 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. Ask for help if you need 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.

Grade F-: Short Assessed Programming Exercise 2, Input, Calculation and Variables

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 a program that uses input instructions**
- ❑ **Write a program that uses variables**
- ❑ **Write a program that uses final variables**
- ❑ **Write a program that uses arithmetic expressions**
- ❑ **Write a program that is split in to methods at least one of which returns a result**
- ❑ **Write a program that makes simple use of comments – at least the author's name and date and 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. .

Ordering Enough Balloons

Write a program for an artist who is creating art instillation "experiences" consisting of rooms totally full of balloons that people will walk around in. Different instillations will be in different sized rooms and different sized balloons so need different numbers of balloons to fill them. The salesperson will enter the length, width and height of the room in centimetres. Write separate methods to input each of these values (they may possibly call other general methods) and return the results. These three numbers are multiplied together to get the volume of the room. This should by then be converted to m3 (calculated by dividing the volume by 1,000,000). This is then divided by the volume taken up by each balloon in m3 as given by the user. Print out the final number of balloons to be ordered along with the intermediate results as in the examples below. The final answer should be given as an integer (rounded down) as below.

An example run of the program is as follows (numbers in bold are typed in by the user, either pop-up boxes or the console may be used for input and output):

```
Length of the room (in cm)? 4600
Width of the room (in cm)? 310
Height of the room (in cm)? 200
What is the balloon volume (in m3)? 0.03
Your room volume is 285.2 m3.
You need 9506 balloons.
```

Another example run:

```
Length of the room (in cm)? 1000
Width of the room (in cm)? 520
Height of the room (in cm)? 330
What is the balloon volume (in m3)? 0.07
Your room volume is 171.6 m3.
You need 2451 balloons.
```

HINT: You can round down to an integer, getting rid of spurious decimal places, using the integer cast operator (int) - see the interactive notebook on types.

Take an example program from the QM+ page or interactive programs as your starting program – modifying it to do this.

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:

- ❑ **The program runs correctly for all input (not just he examples above) including extreme values.**
- ❑ **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 are able to do the following in solving the problem

- ❑ **Write a literate version of the code**
- ❑ **EXPLAIN HOW YOUR PROGRAM WORKS**
- ❑ **Write a program that uses if-then-else statements.**
- ❑ **Write a program that is split in to methods and includes methods that return a result.**
- ❑ **Write a program that includes useful comments, at least one per method saying what it does.**
- ❑ **Write a program that uses indentation in a way that makes its 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.

Parking Charges Write a program that works out the amount a person has to pay for parking in a car park in a tourist town. If they say they are disabled, they are told it is free. Otherwise they enter the number of hours as a whole number (1-8) that they wish to park as well as whether they have an "I live locally" badge or are an old age pensioner both of which leads to a discount. The program tells them the cost to park.

The calculation is done in this program as follows. If they are disabled it is free. Otherwise … Take number of hours they wish to park and give a basic charge:

- 1 hour: 3.00 pounds
- 2-4 hours: 4.00 pounds
- 5-6 hours: 4.50 pounds
- 7-8 hours: 5.50 pounds

Next modify the resulting charge based on whether they are local or not:

- If local: subtract 1 pound
- If OAP: subtract 2 pounds

Your program MUST include methods including ones that

- asks for the hours and returns the basic charge.
- asks whether they live locally / are an OAP and returns the amount to subtract

An example run of the program (words in **bold** are typed by the user: pop-up boxes may be used for I/O):

```
Are you disabled? Yes
Parking for you is free
```

Another example run:

```
Are you disabled? No
How many hours do you wish to park (1-8)? 5
Do you have an "I live locally badge"? Yes
Are you an OAP? Yes
The parking charge for you is 1.50 pounds.
```

Another example run:

```
Are you disabled? No
How many hours do you wish to park (1-8)? 5
Do you have an "I live locally badge"? Yes
Are you an OAP? Yes
The parking charge for you is 1.50 pounds.
```

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.**
- ❑ **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 method adding methods as you write them. It can be called from main but commented out of the final version

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: Records and For 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**
- ❑ **Write a program using counter controlled FOR loop statements.**
- ❑ **Write a program that creates user-defined types, defining and using records.**
- ❑ **Write a program that has at least one method that take argument(s) and returns a result**
- ❑ **Write a program that includes useful comments, at least one per method saying what it does.**
- ❑ **Write a program that uses indentation in a way that makes its structure clear.**
- ❑ **Write a program that uses 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.

Tourist Attraction Information Write a program that gives information about tourist attractions. The user should first input how many tourist attractions they wish to ask about and then be allowed to name that many places. The program should give their opening time (assumed on the hour and in the morning for the purposes of this exercise) and whether they open on bank holidays. A new type called Attraction must be created (a record type) and each separate piece of information about a station should be stored in a separate field of the record (its name - a String, opening time - an integer, closing time - an integer, and whether they open on bank holidays -a boolean).

A separate method must be written that given a String (an attraction name) as argument returns a String containing the correct information about the attraction to print. The String should then be printed by the calling method. An example run of the program (**bold** words are typed by the user): Your answer need only include the information about known stations as in this example.

```
How many attractions do you need to know about? 4

Name attraction 1? The Eden Project
The Eden Project opens on bank holidays.
It opens at 9am.

Name attraction 2? Tate Modern
Tate Modern does not open on bank holidays.
It opens at 10am.

Name attraction 3? The Zoo
I have no information about that attraction.

Name attraction 4? London Zoo
London Zoo opens on bank holidays.
It opens at 10am.
```

TESTING Short Assessed Programming Exercise 4: For Loops and 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 methods return the correct result**
- ❑ **The program always does the correct number of iterations**
- ❑ **All FIELDS of any RECORD are set and accessed correctly**
- ❑ **The program deals with input it doesn't know about**
- ❑ **Running totals are correct at all points through the loop**

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. 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 waz 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 are able to do the following in solving the problem

- ❑ **Write a literate version of the code**
- ❑ **EXPLAIN HOW YOUR PROGRAM WORKS**
- ❑ **Write a program that uses an array manipulated in a loop.**
- ❑ **Write a program that includes methods that take multiple arguments including array arguments.**
- ❑ **Write a program that is well indented.**
- ❑ **Write a program that contains helpful comments.**
- ❑ **Write a program that uses variable names that give an indication of their use.**
- ❑ **Use final variables to store literal values rather than them appearing through the code.**
- ❑ **Ensure all variables have minimal scope**

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.

Top Olympic/Paralympic Medal winners Write a program that uses a for loop to ask the user to name 5 Olympians or Paralympians. They should say how many medals each won and what their sport is. Their names should be kept in one array, the number of medals in a second array in the corresponding position, and their sport in a third. A final variable should be used to store the array size.

The program should then give the average number of medals these people won (rounded to the nearest integer). It should finally print out each person's information in a list in reverse order, with commas separating the sport, name, medals won (suitable for input to a spreadsheet program).

```
Name Olympians/Paralympian 1? Kadeena Cox
How many medals did he/she win? 6
What sport did he/she compete in? Running/Cycling

Name Olympians/Paralympian 2? Laura Kenny
How many medals did he/she win? 6
What sport did he/she compete in? Cycling

Name Olympians/Paralympian 3? Mo Farah
How many medals did he/she win? 4
What sport did he/she compete in? Athletics

Name Olympians/Paralympian 4? Sarah Storey
How many medals did he/she win? 28
What sport did he/she compete in? Cycling/Swimming

Name Olympians/Paralympian 5? Adam Peaty
How many medals did he/she win? 5
What sport did he/she compete in? Swimming
Between them they won an average of 10 medals each.

Swimming, Adam Peaty, 5
Cycling/Swimming, Sarah Storey, 28
Athletics, Mo Farah, 4
Cycling, Laura Kenny, 6
Running/Cycling, Kadeena Cox, 6
```

HINT: Have a variable keep a running total. Round a number to the nearest int using the Math.rint() method.

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 it's always worth checking the code cannot run off the end of the array (make sure it can not visit non-existent position 5 in an array of length 5 for example – remember it starts counting positions from 0!) Checking position 0 is used correctly is important 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**
- ❑ **Write a program using WHILE loops.**
- ❑ **Use indentation well.**
- ❑ **Provide helpful comments in the code.**
- ❑ **Use variable names that give an indication of their use.**
- ❑ **Ensure all variables have minimal scope**

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.

Train Count Write a program that is to be used to investigate how late trains passing through a given station are. The trains are so dire that virtually all trains seem to be running late. It should use a while loop to repeatedly ask the user to name the destination of the train that just departed. It should stop when the special code XXX is entered, and then give the total minutes late of all trains departed and which train was most punctual (ie least late). For example, one run might be as follows.

```
What is the destination of the train that just departed? Liverpool
How many minutes late was it? 7

What is the destination of the train that just departed? Newcastle
How many minutes late was it? 1

What is the destination of the train that just departed? Southampton
How many minutes late was it? 5

What is the destination of the train that just departed? Taunton
How many minutes late was it? 3

What is the destination of the train that just departed? XXXX

The trains were in total 16 minutes late.
The most punctual train was to Newcastle. It was 1 minute late.
```

HINT: Have variables that store the most punctual train seen so far and its destination. If more than one train is equally punctual you may choose either.

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: 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**
- ❑ **Write programs consisting of multiple methods, call them with multiple arguments and return values from them**
- ❑ **Define an abstract data type with record fields accessed ONLY by procedural programming style accessor methods and an initialisation method all defined in the class containing main**
- ❑ **Write a program that is well indented.**
- ❑ **Write a program with each method individually and helpfully commented on its use.**
- ❑ **Write a program that uses variable names that give a clear indication of their use.**
- ❑ **Write a program where all variables have minimal scope**

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.

Paralympic Relay A new paralympic relay competitions involves competitors of different disability classes making up a team. The first leg must have someone from T11 or T13, leg two has someone from T61 or T62, leg three has someone from class T35 or T36 and the final leg has someone from the T51 or T52 class.

Write a program that checks the legality of relay teams given the country and the disability class of the entrant for each relay leg. A new type called **UniversalRelayTeam** should be created (a record type). Information about a team should be stored in fields of the record: (country - a String, Leg 1, Leg 2, Leg 3 and Leg 4 storing the disability classes of each person as 4 integers). This record **must** be defined in **a new class containing NOTHING but the field definitions (no methods at all in the record class)**.

In the main class (the one containing ALL methods including the main method), an initialisation method should be provided to create records. It should take as arguments the values previously input about a team. Accessor methods must also be defined for the fields in the main class (they should NOT be in the UniversalRelayTeam class for this exercise**).**

No other method should access the record fields directly apart from the accessor methods.

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

```
What is the classification (maximum points) of this relay event? 34
What country is the team representing? GB
What is the disability class for leg 1? T 11
What is the disability class for leg 2? T 61
What is the disability class for leg 3? T 52
What is the disability class for leg 4? T 12
The GB team is: Leg 1, T11; Leg 2, T61; Leg 3, T52; Leg 4, T12
Leg 3 (T52) is not legal.
Leg 4 (T12) is not legal.
```

TESTING Short Assessed Programming Exercise 7: 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.**
- ❑ **Contains multiple methods including accessor methods that each individually work and return the correct results for a range of inputs.**

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 called from main but commented away in the final version 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.

Grade A+: Short Assessed Programming Exercise 8: Recursion

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 a program that uses recursion in an appropriate way to solve a recursive problem.**
- ❑ **Write a program that is well indented, to clearly show the program structure.**
- ❑ **Write a program that contains helpful comments with ALL methods clearly commented.**
- ❑ **Write a program that uses variable names that give a clear indication of their use.**
- ❑ **Write a program where all variables have minimal scope**

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.

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. Legal expressions in this language involve putting the operator before its arguments (this is called Polish notation).

```
<EXP> =      * <DIGIT> <EXP> | T <EXP> | <DIGIT>
<DIGIT> =    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C
```

Instead of writing 3*2, in this language you write *32 (which evaluates to 6). EXP stands for expressions. * means multiply the two digits that follow (after evaluation) so *24 is 8 and *5*34 is 60 as it first multiplies 3 and 4 to get answer 12 then multiplies 5 and 12 to get 60. Tn means add the next 3 numbers from n (so eg T3 means 3+4+5=12, T7 means 7+8+9=24), DIGIT gives a way to express numbers up to 12 as a single digit (so A means 10, B means 11, C means 12).

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 B
The answer is 11
```

Another example run:

```
Please input the expression T4
The answer is 15
```

Another example run:

```
Please input the expression T*23
The answer is 21
```

Another example run:

```
Please input the expression T*2*1T2
The answer is 57
```

Another example run:

```
Please input the expression *2*3*A5
The answer is 300
```

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 3 A-F grades

Whatever the grade you get for this, you get that grade three times. You must write a literate version of 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 course. Possible programs are given on the subsequent pages. **You choose ONE of the 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 version (that has 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 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 checked by a fellow student and then marked by a demonstrator. If you are confident then you can skip getting some levels marked separately by the demonstrator. Earlier levels 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

- **at level 1**
- **at three different levels overall and**
- **in three different weeks' labs through term.**

The level achieved **MUST** be confirmed in the online master record.
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.

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 accepted.*

Example mini-projects (choose one)

The following are example programs with indicative grades for different levels of development. You must choose one of the 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 handle out of range values sensibly, is not commented correctly, or is not indented.

Suggestion 1: “Secret Traps and Passages” Quiz Board Game program

Write a multiple choice quiz **procedural program** where players roll a dice to move around a “board” that contains traps and secret passages. On each move they must answer a question. If they land on a trap and they get the question wrong they fall through the trap to an earlier square. If they land on a secret passage entrance and get the question right they jump ahead to a later square. On any other square they go back a square if they get the answer wrong.

Level 1 – F minus minus: Getting started

- ❑ 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 at start of program.

Example (one way to achieve the above): The program prints out a question with choices and allows the person to type an answer. The program then just repeats the question. A method is called to print the question.

Level 2 – F minus: Making progress

- ❑ All the constructs 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 give at least authors name and what the program as a whole does.
- ❑ Variable declarations are within methods to reduce scope to a minimum – **no use of global variables.**

Example (one way to achieve the above): As above, but the person first rolls a (virtual) dice and this is added to the value in a variable holding their current position. In this version they are asked the same question whatever they roll. They are now told whether the correct answer to the question. Their answer is returned as the result from a method, to be printed by the calling method “You answered …”. The main method is structured as a sequence of calls to methods: 1) call a method to roll a virtual dice and print current position 2) call a method to ask the question, 3) print their answer 4) call a method to give the correct answer.

Level 3 – F: Getting there

- ❑ All the constructs and features above AND
- ❑ Includes **Decision statements**.
- ❑ Defines and uses **at least one method that take argument(s)**
- ❑ Indentation consistent
- ❑ Simple comment for each method saying what it does.
- ❑ Variable declarations are within methods to reduce scope to a minimum – **no use of global variables.**

Example (one way to achieve the above): As above, but the person is now told whether they got the question right or not. If they got the wrong answer then one place is deducted from their board position. A separate method is called. It is given the person’s answer as an argument, checks it and prints whether they were right or not.

Level 4 - D: Bare Pass

- ❑ All the constructs and features above AND
- ❑ Be a complete program that compiles and runs locally on the desktop as well as a version in JHUB
- ❑ Includes **Loops**
- ❑ Includes **records**.
- ❑ **Methods** both **take arguments and return results**
- ❑ Comments included are helpful, and each method comments what it does.
- ❑ Some use of well chosen variable names which give some information about use
- ❑ Indentation is good making structure of program clear.
- ❑ **Well-structured** in to multiple **methods**
- ❑ **No use of global variables.**

Example (one way to achieve the above): As above, but now the question and correct answer are stored in a record as separate fields. The record is passed as a single argument to a method that asks the question and checks the answer. It returns a boolean indicating whether they were right. Another method is given the board position, and whether they were right or not, and returns their new position. Position 3 is a secret corridor. If they land there and get the question right they jump to position 7. Position 5 is a trap that takes them back to 1 if wrong. The program uses a loop to allow multiple people to have a turn (all asked the same question) and tells each where they finish.

Continued overleaf

Level 5 - C: Pass

- ❑ All the constructs and features above AND
- ❑ Includes **Arrays**
- ❑ Defines and uses **accessor methods** (all defined in the main class) to access records.
- ❑ 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 – no use of global variables.

Example (one way to achieve the above):
As above but the program now keeps each person's current position in an array. Another array keeps track of where the traps and secret passages are. If it stores 0 then there is nothing there. If it stores a negative number then there is a trap with the number indicating how far they fall back, and a positive number indicates a passage and how far they go forwards. Each array should be initialised with a special method. The arrays are passed as an argument to methods that need the information stored in them. An Initialise method sets the question/answer record and Accessor methods are defined for question records and those methods are the only way the fields of those records are accessed.

Level 6 - B: Satisfactory

- ❑ All the constructs and features above AND
- ❑ Includes **Loops within loops**.
- ❑ Includes procedural programming style **abstract data type** with a clearly commented/specified set of operations.
- ❑ Excellent style over comments, indentation, variable usage etc.
- ❑ Methods individually commented and well indented.
- ❑ Clear structure of multiple methods doing distinct jobs that take arguments and return results

Example (one way to achieve the above):
As above but the program now asks a series of questions to each person controlled by another loop. The questions, held in records, are stored in an array of records but one that is embedded in its own abstract data type of type QuestionBank accessed by methods defining a clear set of primitive operations available on a question bank. Accessor methods should not be written for illegal operations on the QuestionBank, for example. All operations on the QuestionBank should be done through the primitives provided. The program uses a loop to control the number of moves, and the game finishes when someone gets to the end of the board.

Level 7 - A: Merit

- ❑ All the constructs and features above AND
- ❑ Includes a **sort algorithm** as a separate method.
- ❑ Excellent style over comments, indentation, variable usage etc.
- ❑ Consistent use of well-placed variable names that give a clear indication of their use (perhaps making comments about them redundant). Use of final variables for literal constants.
- ❑ Excellent use of methods throughout
- ❑ Variable declarations are within blocks to reduce scope to a minimum – no use of global variables.

Example (one way to achieve the above):
As above but prints after each turn a table of the player's current positions together with their name (as input at the start). This table should be printed in sorted order with the person who is leading first. The number of players should be input right at the start and the sort method should work whatever the number of players there are. Ensure your program is structured so that distinct tasks are done within individual methods that pass data between them using arguments.

Level 8 – A+: Distinction

- ❑ Includes everything required for an A grade and
- ❑ **BOTH file input AND file output**

Example (one way to achieve the above):
Allows the player names and board positions to be saved at the end of each round so that the game can be interrupted (ie program stopped) and the game continued later. The program loads in a specified file containing the positions at the start. Design a suitable file format to allow this to be done easily.

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 more advanced 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.

Suggestion 2: A Chat Bot

Write a chat bot **procedural program** that can have realistic conversations with people about a specific topic (eg films, football, musicals, pop music, …)

Level 1 – F minus minus: Getting started

- ❑ 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 at start of program.

Example (one way to achieve the above): The program introduces itself and asks the person how they are, but whatever the answer says that it is nice to meet them, but it just discovered it has to go.

Level 2 – F minus: Making progress

- ❑ All the constructs 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 give at least authors name and what the program as a whole does.
- ❑ Variable declarations are within methods to reduce scope to a minimum – **no use of global variables.**

Example (one way to achieve the above): As above, but the chatbot now also says something about its interests and asks the person a specific question about it. Things the chatbot says are now stored in local String variables with separate statements concatenated together. The question is asked by a method that returns the answer given. The main method is structured as a sequence of calls to methods: 1) call a method to describe its interests 2) call a method to ask the question, 3) repeat their answer "So you like …" 4) call a method to say goodbye.

Level 3 – F: Getting there

- ❑ All the constructs and features above AND
- ❑ Includes **Decision statements**.
- ❑ Defines and uses **at least one method that take argument(s)**
- ❑ Indentation consistent
- ❑ Simple comment for each method saying what it does.
- ❑ Variable declarations are within methods to reduce scope to a minimum – **no use of global variables.**

As above, but the chatbot now responds in a different way depending on the answer to their question. A separate 'choice' method is passed the person's answer as argument and prints the appropriate next thing to say eg it asks about their favourite kind of film and says something different depending on what genre they answer, though it may only know one or two genres. (eg if "Horror" is the answer to the first question that would lead to the next statement "I never watch them as they terrify me.")

Level 4 - D: Bare Pass

- ❑ All the constructs and features above AND
- ❑ Be a complete program that compiles and runs locally on the desktop as well as a version in JHUB
- ❑ Includes **Loops**
- ❑ Includes **records**.
- ❑ **Methods** both **take arguments and return results**
- ❑ Comments included are helpful, and each method comments what it does.
- ❑ Some use of well chosen variable names which give some information about use
- ❑ Indentation is good making structure of program clear.
- ❑ **Well-structured** in to multiple **methods**
- ❑ No **use of global variables.**

Example (one way to achieve the above): As above, but now after asking its first question it looks if the person's answer is a specific 'trigger' word (ie a word the program knows a response for) stored with the response in a record. The record includes a field for the trigger and a field for the response (eg "Horror" might be paired with the statement "I never watch them as they terrify me." and those two strings are stored in, and accessed from, different fields of the record). The record is passed as a single argument to the method(s) that looks for their answer matching and it returns the linked response back.

The program also uses a loop so that when the conversation ends the chatbot then starts a new conversation until someone types "Goodbye" or "Got to go" at which point the program ends.

Continued overleaf

Level 5 - C: Pass

- ❑ All the constructs and features above AND
- ❑ Includes **Arrays**
- ❑ Defines and uses **accessor methods** (all defined in the main class) to access records.
- ❑ 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.
- ❑ No use of global variables.

Example (one way to achieve the above):
As above but the program now has multiple things it might ask when no trigger word is given. These questions are stored in an array. If there is no trigger given, what is said is picked at random from the array based on the "roll" of a virtual dice. This array should be initialised by a special method. The array is now passed as argument to method(s) that need the information. Separate methods are separately called for checking the trigger-response way of answering and picking something at random. Accessor methods are now defined for TriggerResponse records and those methods are the only way trigger and response data is accessed. An initialise method creates the record.

Level 6 - B: Satisfactory

- ❑ All the constructs and features above AND
- ❑ Includes **Loops within loops**.
- ❑ Includes procedural programming style **abstract data type** with a clearly commented/specified set of operations.
- ❑ Excellent style over comments, indentation, variable usage etc.
- ❑ Methods individually commented and well indented.
- ❑ Clear structure of multiple methods doing distinct jobs that take arguments and return results

Example (one way to achieve the above): As above but the program now repeatedly chooses something to say from the array of statements controlled by a loop. It in addition it has multiple trigger-response records stored in an array that is checked whenever the person answers a question. The trigger response array is embedded in its own abstract data type, of type TriggerResponseDatabase and accessed by a set of methods defining clear operations available on the database. Accessor methods should not be written for illegal operations on the Trigger-Response Database, for example. All operations on the Database should be done through the primitives provided.

Level 7 - A: Merit

- ❑ All the constructs and features above AND
- ❑ Includes a **sort algorithm** as a separate method.
- ❑ Excellent style over comments, indentation, variable usage etc.
- ❑ Consistent use of well-placed variable names that give a clear indication of their use (perhaps making comments about them redundant). Use of final variables for literal constants.
- ❑ Excellent use of methods throughout
- ❑ Variable declarations are within blocks to reduce scope to a minimum – no use of global variables.

Example (one way to achieve the above):
As above but now the trigger response records in the database also include a third field holding a number indicating how useful they are expected to be (eg more people watch Romcoms than horror, so the former has a higher usefulness score. When the program first runs the records are sorted by this field so that the more useful trigger words are checked first. Ensure your program is structured so that distinct tasks are done within individual methods that pass data between them using arguments.

Level 8 – A+: Distinction

- ❑ Includes everything required for an A grade and
- ❑ **BOTH file input AND file output**

Example (one way to achieve the above): As above but now trigger and response data is loaded at the start from a file. The usefulness score is updated each time the program is run so that the usefulness of triggers that came up are increased. If data is to be input then the user inputs a new set of triggers and answers and stores them in a file. Design a suitable file format to allow this to be done easily.

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 more advanced 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.

Suggestion 3: Explore the forest, heal the animals 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 – F minus minus: Getting started

- ❑ 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 at start of program.

Example (one way to achieve the above): The program asks the person for their adventurer's name which it then uses. 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…" A method is called to print the message.

Level 2 – F minus: Making progress

- ❑ All the constructs 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 give at least authors name and what the program as a whole does.
- ❑ Variable declarations are within methods to reduce scope to a minimum – **no use of global variables.**

Example (one way to achieve the above): As above, but they are now asked their adventurer name by a method which returns the name as a result. The adventurer has "healing points", starting with an initial total of eg 10 points stored in a variable initialised in the main method. They heal the wolf and are deducted two healing points for doing so. The main method is structured as a sequence of calls to methods: 1) call a method to ask their name that returns their answer, 2) print their name 3) call a method to print the rest of the description of the situation 4) call a healing method (that just prints that the animal was healed).

Level 3 – F: Getting there

- ❑ All the constructs and features above AND
- ❑ Includes **Decision statements**.
- ❑ Defines and uses **at least one method that take argument(s)**
- ❑ Indentation consistent
- ❑ Simple comment for each method saying what it does.
- ❑ Variable declarations are within methods to reduce scope to a minimum – **no use of global variables.**

Example (one way to achieve the above): As above, but now their name is passed to the method that describes the situation which prints it. The healing method now asks if they wish to heal the animal or not and if so rolls two virtual six sided dice. If the total dice score is more than the injury score of the the wolf, set as 5 then it is healed. If they succeed in healing the animal their healing points score goes up by 1. If they fail it goes down by 2. If they did not attempt the healing it stays the same. The healing method returns the amount to change the healing points by (+1, -2, 0). The main method updates the score based on this.

Level 4 - D: Bare Pass

- ❑ All the constructs and features above AND
- ❑ Be a complete program that compiles and runs locally on the desktop as well as a version in JHUB
- ❑ Includes **Loops**
- ❑ Includes **records**.
- ❑ **Methods** both **take arguments and return results**
- ❑ Comments included are helpful, and each method comments what it does.
- ❑ Some use of well chosen variable names which give some information about use
- ❑ Indentation is good making structure of program clear.
- ❑ **Well-structured** in to multiple **methods**
- ❑ **No use of global variables.**

Example (one way to achieve the above): As above, but now the adventurer has a healing score and a survival score. These details of the adventurer are stored as a Healer record. The record includes separate fields for: their name, their current healing points and a survival score. If they try and fail to heal an animal it will attack them. They roll a virtual dice and must get a score less than their survival score, or lose a point from their survival score. Separate methods are called for healing and defending. The Healer record is passed as a single argument to methods that need it. The program uses a loop to allow a series of rounds to be played. On each round the adventurer travels on and comes across sick animals to heal. The game finishes when they drop to 0 survival points. Their final score is the final number of healing points they have.

Continued overleaf

Level 5 - C: Pass

- ❑ All the constructs and features above AND
- ❑ Includes **Arrays**
- ❑ Defines and uses **accessor methods** (all defined in the main class) to access records.
- ❑ 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 – no use of global variables.

Example (one way to achieve the above):
As above but the program now has different kinds of animals to come across. The possible animals are stored as an array of Strings and chosen at random. They are given an injury score at random. The array is passed as argument to methods that need it. Accessor methods are now defined for Healer records and those methods are the only way healer data is accessed. An initialise method creates the record.

Level 6 - B: Satisfactory

- ❑ All the constructs and features above AND
- ❑ Includes **Loops within loops**.
- ❑ Includes procedural programming style **abstract data type** with a clearly commented/specified set of operations.
- ❑ Excellent style over comments, indentation, variable usage etc.
- ❑ Methods individually commented and well indented.
- ❑ Clear structure of multiple methods doing distinct jobs that take arguments and return results

Example (one way to achieve the above):
As above but the program now allows a person to repeatedly try and heal an animal if they wish to if it is not cured. Healing plants can also be discovered on a round rather than animals and these add to the healing score. Some plants however are poisonous and so reduce the healing score. Each animal also has a score that indicates how many healing points are gained if healed. The details of each animal/plant is stored as a record. The records include fields for the kind of thing found (animal, plant or neither so no use), name and healing points gained/lost. The information is stored in an array but one that is embedded in its own abstract data type, of type Forest and accessed by a set of methods defining clear operations available on a Forest. All operations on Forest values must be done through the primitives provided. Operations that are illegal on a forest should not have accessor methods.

Level 7 - A: Merit

- ❑ All the constructs and features above AND
- ❑ Includes a **sort algorithm** as a separate method.
- ❑ Excellent style over comments, indentation, variable usage etc.
- ❑ Consistent use of well-placed variable names that give a clear indication of their use (perhaps making comments about them redundant). Use of final variables for literal constants.
- ❑ Excellent use of methods throughout
- ❑ Variable declarations are within blocks to reduce scope to a minimum – no use of global variables.

Example (one way to achieve the above):
As above but details of the animals/plants encountered are stored in an array as they are discovered. On request these can be viewed at which point a sort method is called so that they can be listed in order of total healing points gained (or lost) from encountering them.

Level 8 – A+: Distinction

- ❑ Includes everything required for an A grade and
- ❑ **BOTH file input AND file output**

Example (one way to achieve the above):
Allows the current details of the adventurer (the game "state") 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.

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 more advanced 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.

Suggestion 4: Pet Program

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

Level 1 – F minus minus: Getting started

- ❑ 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 at start of program.

Example (one way to achieve the above): The program asks you to name your pet, then prints a message using the name (eg Happy 0th Birthday Aled the Anteater). A separate method is called to explain what the program does at the start.

Level 2 – F minus: Making progress

- ❑ All the constructs 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 give at least authors name and what the program as a whole does.
- ❑ Variable declarations are within methods to reduce scope to a minimum – **no use of global variables.**

Example (one way to achieve the above): As above, but 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). A method is now used to name the pet, returning the name given. The main method is structured as a sequence of calls to methods: 1) call a method to ask them to name the pet that returns their answer, 2) print the name 3) work out the new score for the hunger level 4) call a method to print a message about the state of the pet (but in this version always prints the same message.

Level 3 – F: Getting there

- ❑ All the constructs and features above AND
- ❑ Includes **Decision statements**.
- ❑ Defines and uses **at least one method that take argument(s)**
- ❑ Indentation consistent
- ❑ Simple comment for each method saying what it does.
- ❑ Variable declarations are within methods to reduce scope to a minimum – **no use of global variables.**

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

Level 4 - D: Bare Pass

- ❑ All the constructs and features above AND
- ❑ Be a complete program that compiles and runs locally on the desktop as well as a version in JHUB
- ❑ Includes **Loops**
- ❑ Includes **records**.
- ❑ **Methods** both **take arguments and return results**
- ❑ Comments included are helpful, and each method comments what it does.
- ❑ Some use of well chosen variable names which give some information about use
- ❑ Indentation is good making structure of program clear.
- ❑ **Well-structured** in to multiple **methods**
- ❑ **No use of global variables.**

Example (one way to achieve the above):
As above, but pets also have a health level: a description of the pet consisting of its name, species, hunger and happiness level are stored as a Pet record. The record is passed as a single argument to the methods that use and change the information. The hunger and sadness levels increase by a random amount. The player can choose to feed the pet to decrease its hunger by a random amount or cuddle it to make it happier. The program uses a loop to allow a series of rounds to be played. On each round the hunger and sadness of the pet are changed randomly (up or down). The player chooses one way to look after the pet in that round. The pet dies if at the highest hunger level or highest sadness level for two consecutive rounds. The player wins the game if they survive to a given round without the pet dying.

Continued overleaf

Level 5 - C: Pass

- ❑ All the constructs and features above AND
- ❑ Includes **Arrays**
- ❑ Defines and uses **accessor methods** (all defined in the main class) to access records.
- ❑ 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 – no use of global variables.

Example (one way to achieve the above):
As above but the program now allows multiple pets to be owned and their details stored in an array of records. On each round, hunger and sadness levels of all change, but only one can be chosen to be looked after. The array is passed to methods that use it. Accessor methods are defined for the pet records and those methods are the only way a pet record is accessed. An initialise method creates the record.

Level 6 - B: Satisfactory

- ❑ All the constructs and features above AND
- ❑ Includes **Loops within loops**.
- ❑ Includes procedural programming style **abstract data type** with a clearly commented/specified set of operations.
- ❑ Excellent style over comments, indentation, variable usage etc.
- ❑ Methods individually commented and well indented.
- ❑ Clear structure of multiple methods doing distinct jobs that take arguments and return results

Example (one way to achieve the above):
As above but on each round, loops are used both to control the rounds and to print the current state of each pet so the decision of which to attend to can be decided. Pets can interact so if two are happy they both become very happy as they play together. The separate pet records are still stored in an array but one that is embedded in its own abstract data type of type Pets accessed by methods defining the operations available on the pets. All operations on the Pets should be done through the primitives provided. Operations that are illegal on Pets should not have accessor methods.

Level 7 - A: Merit

- ❑ All the constructs and features above AND
- ❑ Includes a **sort algorithm** as a separate method.
- ❑ Excellent style over comments, indentation, variable usage etc.
- ❑ Consistent use of well-placed variable names that give a clear indication of their use (perhaps making comments about them redundant). Use of final variables for literal constants.
- ❑ Excellent use of methods throughout
- ❑ Variable declarations are within blocks to reduce scope to a minimum – no use of global variables.

Example (one way to achieve the above):
As above but includes a sort method to list the pets sorted in order of combined hunger and sadness levels, allowing the player to choose which it is most urgent to deal with in each round.

Level 8 – A+: Distinction

- ❑ Includes everything required for an A grade and
- ❑ **BOTH file input AND file output**

Example (one way to achieve the above):
Allows the current pet world state to be saved into a file, storing the details of each pet, 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.

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 more advanced 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.

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

The mid-term and end-term open book tests are assessed and count for 6 A-F grades in total.

You will be required to take a series of tests through term (see below). Because of the pandemic, this year, all such tests will be done online and be open book. This means they are not in full exam conditions and you can consult the learning materials of the module.

However, you must NOT get help from any other people 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 only slightly reworded. Practice writing answers yourself. It 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 a reason for work being late - the extra time is already added to allow for that.

Mid–term test (assessed: counts as 3 A-F grades)

You will be required to take a mid term test part way through term. 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.

For the mid term test **only**, any one who gets LESS than a D grade on ALL three parts will be given a second chance at the end of term in their own time to do it again, though capped at a single D grade. This is to ensure it is what you can do at the end that matters over what you do earlier.

End-term test (assessed: counts as 3 A-F grades) must be passed to pass coursework)

You must also take a written end-term test (previously known as the end-term test) towards the end of term. This may be split over two weeks. You must pass at least one question 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 with each weekly unit. 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 of 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 of 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 the 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 in steps
- work out what a program does without executing it
- debug programs to ensure they work correctly
- explain programming constructs and
- 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) - Extenuating Circumstances (ECs)

If you miss a test or exam (or any coursework deadline) and have a good reason (eg illness) you MUST apply for ECs or you will get 0/Q. 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 need documentary evidence (so if ill you need a Drs note).

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

The exam is 50% of your final mark, and MUST be passed to pass module.

The exam is an open book exam - see the advice about open book exams with respect to tests as they apply to the exam too.

You must do a written final assessment / exam in January. It is combined 50-50 with the overall coursework mark. You must pass this to pass the module, however, otherwise 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 with each weekly unit as well as 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, must be able to explain them in your own words, and must also clearly demonstrate you understand and can write about programming constructs.

The grade for your coursework components (A-F grades) are only combined with the exam result if you pass the exam. If you fail the exam your overall mark will be capped so that you fail the module. If you do not pass the module then you will have to take a resit exam in 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 module content 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.

Resit Exam

Anyone who fails the module 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) 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.

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. ECS401 Assessed **PROGRAM** Progress Sheet

You may wish to print this sheet and tick off your progress as the demonstrators mark your programs and you see the grades appear in the Master record.

OR

do so in an online JHUB Notebook version of it - see the ASSESSMENT folder.

Short Level	**Grade**	**SHORTS IS WORTH DOUBLE**	**Date Signed Off**	**Mini-Proj level**	**Grade**	**MINI IS WORTH TRIPLE**	**Date Signed Off**
1	**F- -**	**Output**		1	**F - -**	**Input / Output**	
2	**F-**	**Input**		2	**F -**	**Assignment / Expression**	
3	**F**	**Decisions (If statements)**		3	**F**	**Decisions (If statements)**	
4	**D PASS**	**For loops + Records**		**4**	**D PASS**	**Loops + Records**	
5	**C**	**Arrays**		**5**	**C**	**Arrays**	
6	**B**	**While loops**		**6**	**B**	**Loops inside loops+ADT**	
7	**A**	**ADT**		**7**	**A**	**Sort method**	
8	**A+**	**Recursion**		**8**	**A+**	**File I/O**	

To get a coursework mark for your programming work you must by the deadline at the end of term:

1. **CHECK that every program a demonstrator marked as passed has appeared in the master online record** as passed

DEADLINE: The start of your last lab, Tuesday 14 December 2021

2. **Hand in 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 tutor in the labs,
   - a pdf of the **LAST THREE** literate program versions of your mini-project downloaded from the Interactive Jupyter Notebook already assessed by a tutor 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 tutors are only required to mark at most 2 of your programs in the last 4 programming labs so do not leave it until the last minute.

DEADLINE: 10 am GMT, Wednesday 15 December 2021

If it is not signed off in the online master record (which you find on QM+ - see the Assessment section for details of where) then you have NOT got the grade.