ECS401 Procedural Programming
Assessment Booklet: September 2020

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 test that you have gained them.

IF your assessed work (programs, explanations or test answers) is found to be substantially the same as someone else's (whoever was the original source) THEN you will automatically lose the grade.

Rules of Helping Others (outside of assessed tests)

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 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 copies of your assessed programs or explanations.

NEVER EVER take someone else's program and (even with changes) present it as your own

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.

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 1 A-F grade)
• A programming mini-project	(worth 2 A-F grades)
• A mid-term theory test (similar question style to the final assessment)	(worth 1 A-F grades)
• A mid-term programming test (also similar the final assessment)	(worth 1 A-F grades)
• An exit theory test (also similar the final assessment)	(worth 1 A-F grades)
• An exit programming test (also similar the final assessment)	(worth 1 A-F grades)

YOU MUST PASS THE EXIT TEST TO PASS THE COURSEWORK

This gives 7 A-F grades, turned in to a single coursework mark at the end of the term (see over).

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 throughout term by working hard on the coursework FROM THE START generally pass. The non-assessed exercises (eg the 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) 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 you 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%

Calculating your Final Coursework Grade

Your overall grade for the coursework is calculated as follows from the 7 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?**
7A+s	Grade A*	100%	PASS
6A+s and 1D (or better)	Grade A++	96%	PASS
4A+s and 1A, 1D and 1 other (or better)	Grade A+	86%	PASS
4As and 1B, 1D and 1 other (or better)	Grade A	76%	PASS
4Bs and 1C and 1D and 1 other (or better)	Grade B	66%	PASS
4Cs and 2D and 1 other (or better)	Grade C	56%	PASS
6Ds and 1 other (or better)	Grade D	46%	PASS
4Ds and 2Fs and 1 other (or better)	Grade F	30%	NOT PASSED
6Gs and 1 other (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.

Assessment Deadlines

For the purpose of this module weeks are counted as follows

Wk	**week start - end (mon - fri)**	**Advisory Deadlines** You should do the programs and get them marked week by week. The 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**
1	21 Sept - 25 Sep		
2	28 Sep - 2 Oct	**Short programming exercise 1** marked no later than YOUR allocated lab	
3	5 Oct - 9 Oct	**Short 2 and Miniproject level 1** marked no later than YOUR allocated lab	
4	12 Oct - 16 Oct	**Short programming exercise 2** marked no later than YOUR allocated lab	
5	19 Oct - 23 Oct		**MID-TERM TEST** in YOUR timetabled lab
6	26 Oct - 30 Oct	**Short 3** marked no later than YOUR allocated lab	
7	2 Nov - 6 Nov	**Short 4 and Mini level 4** marked no later than YOUR allocated lab	
8	9 Nov - 13 Nov	*DON'T LEAVE IT TOO LATE TO GET PROGRAMS DONE AND MARKED!* At most 2 programs can be marked each week	
9	16 Nov - 20 Nov	At most 2 programs can be marked each week	
10	23 Nov - 27 Nov	At most 2 programs can be marked each week	**EXIT TEST: keep free Wednesday 2pm-5pm**
11	30 Nov - 4 Dec	At most 2 programs can be marked this week	
12	7 Dec - 11 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 7 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)

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 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 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 **interactive notebooks** as well as standalone programs.
- 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 single FULL version of the program at the end of the notebook
        - that will 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 changing version after each time it has been assessed if assessed multiple times)
    - An overview of its purpose

Short Programming Exercises (assessed 1xA-F)

- The short programming exercises count for 1 (A to F) coursework grade.
- 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
- ALL programs must be written in Java in a procedural programming style.

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 satisfactorily 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 handbook. 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.

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**
- ❑ **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. .

Student Loan Calculation

A Student Loan company is providing an online calculator for students to work out how much they owe. They first input how much they owed at the start of the year, then how much they paid off that year. You may assume these are in whole numbers of pounds. Write separate methods to input each of these values (they may possibly call other general methods). After subtracting the amount paid off, it then adds 7% interest to the end of year total and prints out the new total. The final answer should be given to one decimal place (rounded down). Use a final variable for the percentage rate.

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

```
Amount of loan at start of year? 1000
Amount paid off this year? 5
The new amount owed is (in pounds):1064.6
```

Another example run:

```
Amount of loan at start of year? 50
Amount paid off this year? 6
The new amount owed is (in pounds):47.0
```

HINT: You can round down to one decimal place by multiplying the answer by 10, getting rid of spurious decimal place using the integer cast operator (int) and then dividing that by 10 - 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.

Body Age Write a program that estimates a person's body age - whether they are ageing faster or slower than their actual years - as a measure of how fit and healthy they are. This simplified test is based on the person inputing their actual age in years, their heart rate (beats in a minute) and how far they can stretch forward when sitting on the floor (to the nearest cm). This is the distance between where their fingers fall when back straight and where they fall when stretching forward without bending knees.

The calculation is done in this program as follows.
Take their actual age and add or subtract years to it following the rules:

- Heart rate of less than 62, subtract 5
- Heart rate of 62-64, subtract 1.
- Heart rate of 65-70, add 1.
- Heart rate of 71 or more, add 2.

Next modify the resulting age based on the distance they can stretch:

- Stretch less than 20cm: add 4
- Stretch 20-32cm: add 1.
- Stretch 33-37cm, add 0.
- Stretch more than 37cm, subtract 3.

Your program should include methods including ones that
- asks for each heart rate and returns the amount to add or subtract
- asks for distance they can stretch and returns the amount to add or subtract

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):

```
What is your age? 55
What is your heart rate? 64
How far can you stretch? 37
Your body's age is 54
```

Another example run:

```
What is your age? 21
What is your heart rate? 61
How far can you stretch? 25
Your body's age is 17
```

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.

Station Information Write a program that gives information about underground stations. The user should first input how many stations they wish to ask about and then be allowed to name that many stations. The program should say whether they have step free access and the distance that must be travelled to get from entrance to platform. A new type called Station 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, whether they have step-free access - a boolean, and the distance in metres - an integer).

A separate method must be written that given a String (a station name) as argument returns a String containing the correct information about the station. The String should 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 stations do you need to know about? 4

What station do you need to know about? Stepney Green
Stepney Green does not have step free access.
It is 100m from entrance to platform.

What station do you need to know about? Kings Cross
Kings Cross does have step free access.
It is 700m from entrance to platform.

What station do you need to know about? Oxford Street
Oxford Street is not a London Underground Station.

What station do you need to know about? Oxford Circus
Oxford Circus does have step free access.
It is 200m from entrance to platform.
```

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 Women's Goal scorers Write a program that uses a for loop to ask the user to name 4 female footballers. They should say how many times each has scored in the World Cup and what their country is. The names of the footballers should be kept in one array, the number of goals in a second array in the corresponding position, and their country in a third. A final variable should be used to store the array size.

The program should then name the footballer that was top scorer and the total number of goals all these players scored. It should finally print out each player's information in a list with commas separating the name, country and goals scored (suitable for input to a spreadsheet program).

```
Name footballer 1? Ellen White
How many goals did they score? 6
What country did they play for? England

Name footballer 2? Megan Rapinoe
How many goals did they score? 7
What country did they play for? USA

Name footballer 3? Ajara Nchout
How many goals did they score? 2
What country did they play for? Cameroon

Name footballer 4? Cristiane
How many goals did they score? 4
What country did they play for? Brazil

The highest scorer was Megan Rapinoe.
These players scored 19 goals between them.

The details of the players are:
Ellen White, England, 6
Megan Rapinoe, USA, 7
Ajara Nchout, Cameroon, 2
Cristiane, Brazil, 4
```

HINT: Have variables that store the goals of the highest scorer found so far and its position in the array. Keep a running total in a variable.

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.

Bus Passengers Write a program that is to be used to count how many passengers are travelling on buses that pass a particular bus stop in a given hour. It should use a while loop to repeatedly ask the user to give the number of passengers on the bus that just passed. It should stop when the special code X is entered as the number of passengers. It should then give the number of buses and the total number of passengers counted in that hour. For example, one run might be as follows.

```
How many passengers were on the bus? 2
How many passengers were on the bus? 5
How many passengers were on the bus? 10
How many passengers were on the bus? 3
How many passengers were on the bus? 12
How many passengers were on the bus? 1
How many passengers were on the bus? 0
How many passengers were on the bus? X

There were a total of 33 passengers on 7 buses.
```

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 Swimming Relay Paralympic swimming relay competitions involve competitors of different disability levels making up a team. For those with physical disabilities, each swimmer's disability is rated on a scale of 1-10 and the total points rating for the 4 swimmers must be no greater than a given number of points. For example, a legal relay team for a 34 point freestyle relay may consist of two S8 swimmers and two S9 swimmers (9 + 9 + 8 + 8 = 34), or an S10 swimmer and three S8 swimmers (10 + 8 + 8 + 8 = 34).

Write a program that gives information about relay teams given the country and the maximum points score for the event. It should check that the team is legal. A new type called ParaRelayTeam should be created (a record type) and each separate piece of information (country - a String, and points - 4 integers) about a team should be stored in fields of the record. 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 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 ParaRelayTeam class for this exercise).** These methods must be the only way the record is accessed.

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 category of Swimmer 1? 10
What is the disability category of Swimmer 2? 8
What is the disability category of Swimmer 3? 8
What is the disability category of Swimmer 4? 8
That GB team has 34 points so is 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 testplan method called from main but commented away in the final version that tests each method printing results returned checking they are correct.

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.

<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**

Programming mini-project (assessed 2xA-F grades)

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 below. It must be written as a literate program with both a notebook and final local runnable version. **You choose ONE of these projects.**

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.

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: “Pointless” Answer Quiz program

Write a short answer quiz **procedural program** where the player gets points based on the number of members of the public (out of 100 people asked) who gave that answer (for the purposes of testing you can make up the response numbers). The aim is to choose an answer that as few people as possible thought of so gain as FEW points as possible (eg Name UK olympic track and field gold medal winners. … Suppose 78 people named Dina Asher-Smith, 42 named Sebastian Coe, and 2 named Fatima Whitbread, then an answer of FatimaWhitbread gains 2 points and is the best answer. Wrong answers gain 100 points.)

Level 1 – F minus: Getting started

- ❑ Is a litewrate 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 three options and allows the person to type the answer they believe to be best (ie lowest points). They must type in the full word or phrase answer. The program then prints the points (always the same score in this version). A method is called to print the question.

Level 2 – F: Making progress

- ❑ All the constructs 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 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 is now told whether they got the question right or not and the points score for the answer they chose. That number of points are added to their score. If they gave a wrong answer, their score is set to 100. A separate ‘marking’ method is given the person’s answer as an argument and prints the score.

Level 3 - D: Bare Pass

- ❑ All the constructs and features above AND
- ❑ 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 question, correct answers and their point scores are stored in a record as separate fields. The record is passed as a single argument to methods that ask the question and it checks the person’s answers. The method that checks the answer, returns the points scored. The points are added to the total by the calling method.

The program uses a loop to ask the question to more than one person and give them each a score in turn.

Continued overleaf

Level 4 - 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 multiple questions stored in a question bank - an array of records - and one is chosen at random to ask. This array should be initialised by a special method. The array is passed as an argument to methods that need the question bank. An Initialise method sets the record and Accessor methods are defined for question records and those methods are the only way the fields of those records are accessed.

Level 5 - 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 a loop. The question records are stored in an array 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. All operations on the QuestionBank should be done through the primitives. Accessor methods should not be written for illegal operations on the QuestionBank. The program uses a loop to control the number of questions asked.

Level 6 - 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 an option to print a table of player's current scores. This table should be printed in sorted order with lowest scorer (ie the one who is leading) first. It 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 7 – A+: Distinction

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

Example (one way to achieve the above):
Allows quiz question sets on different topics to be created in an admin mode of the program and to be saved into a file with name corresponding to the topic. The program loads in a specified files of question 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: Jurassic World program

Write a **procedural program** that simulates bio-engineered dinosaurs that must be cared for.

Level 1 – F 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 the species of dinosaur you have created and prints a message using it (eg "Diplodocus Rex are vicious"). A method called at the start explains what the program is for. A different method called from the same method asks you to name the dinosaur.

Level 2 – F: Making progress

- ❑ All the constructs 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 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 dinosaur now has a calmness score (low calm is bad) and a hunger score (high hunger is bad) stored. The combined calmness and hunger score is turned into a danger level – eg 1: "CALM", 2: "AROUSED", 3: "DANGEROUS" or 4: "KILLER" depending on how large the combined score is. A method is defined that given this danger level as argument prints out an appropriate messages (eg "The Diplodocus Rex is looking like a killer...RUN!!").

Level 3 - D: Bare Pass

- ❑ All the constructs and features above AND
- ❑ 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 species, hunger and calmness values (along with any other information about the individual dinosaur) are stored as a dinosaur record. The record is passed as a single argument to the methods that use and change the information. The player can choose to feed the dinosaur to decrease its hunger by a random amount or to sooth it to increase its calmness level.

The program uses a loop to allow a series of rounds to be played. On each round the hunger and calmness of the dinosaur are changed randomly (up or down). The player chooses which way to look after the dinosaur in that round. The player wins the game if they survive to a preset round without dying (due to the dinosaur becoming a killer).

Continued overleaf

Level 4 - 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 dinosaurs to be created and their details stored in an array of records. On each round, one is chosen at random to be changed. The array is passed to methods that use it. Accessor methods are defined for the dinosaur records and those methods are the only way a dinosaur record is accessed. An initialise method creates the record.

Level 5 - 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, while only two dinosaurs can be attended to, all change emotional state. Loops are used both to control the rounds and to cycle round printing the details of and giving the choice of updating each dinosaur in a single round. Dinosaurs can interact so if one gets dangerous and is hungry it might eat another chosen at random. The separate dinosaur records are still stored in an array but one that is embedded in its own abstract data type of type DinosaurWorld accessed by methods defining the operations available on the dinosaurs. All operations on the DinosaurWorld should be done through the primitives. Operations that are illegal on a dinosaur record should not have accessor methods.

Level 6 - 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 dinosaurs sorted by danger level, allowing the player to choose which it is most urgent to deal with in each round.

Level 7 – 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 dinosaur world 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.

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 3: Gamified Olympics Medal Table Game

Write a **procedural program** that is a gamified version of the Olympic medal table.

Level 1 – F 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 asking which event is next. The program then always prints the same message giving the same fake final medal total of a single country
Great Britain   G: 27 S: 23 B: 17   TOTAL: 67
A method is called to print the question. A separate method called from the same method prints the total.

Level 2 – F: Making progress

- ❑ All the constructs 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 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 program now asks the user to indicate which country won gold, silver and bronze for this event. A menu of countries is given. Each country is numbered and the program converts the number in to the appropriate country. The program at this stage just keeps track of the totals of one country (of your choice). Note it is not a problem if one country wins multiple medals in an event. The medals of the specific country being tracked are added to its medal totals. A separate method is written that, given the gold, silver and bronze totals of the country, prints the total number of each medal and the overall total for that country.

Level 3 - D: Bare Pass

- ❑ All the constructs and features above AND
- ❑ 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 medal details of a country are stored as a record. The record includes fields for: country name, golds won, silvers won and bronzes won. The record is passed as a single argument to method sthat work out totals and print results. A separate method given the separate totals of gold, silver and bronze, returns the total number of medals. Medal winners are determined by the roll of a virtual dice instead of being emptied to gamify the table.

The program uses a loop to process a series of winter olympic events. At this stage they are not named, but are just numbered Event 1, Event 2 and so on.

Continued overleaf

Level 4 - 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 track of multiple countries. The country details are stored in an array of records. The array is passed to methods that use it. An initialise method creates the record and accessor methods are defined for the country records and those methods are the only way a country record is accessed.

Level 5 - 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 tracks more countries and the array of country records is embedded in its own abstract data type, of type MedalTable. It is accessed by a set of methods defining clear primitive operations available on a Medal Table. All operations on the Medal Table should be done through the primitives. Operations that are illegal on a Medal table should not have accessor methods. The program prints out a full medal table for all countries.

Level 6 - 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 an option to print the medal table showing the details of each country in order of total medals won via a sort method.

Level 7 – 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 medal table state to be saved into a file AND reloaded from a file so it can be continued later even if the program ends. 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: A Help Bot

Write a help bot **procedural program** that can give help in a realistic way answering text questions about a specific subject (eg cooking / gardening problems, how to play Minecraft, etc)

Level 1 – F 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 to type a question and whatever they type says it is sorry it does not know the answer.

Level 2 – F: Making progress

- ❑ All the constructs 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 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 helpbot now answers in a different way depending on the question asked. A separate 'choice' method is passed the person's question as argument and prints the appropriate answer. Things the helpbot says are now stored in local String variables with separate statements concatenated together.

Level 3 - D: Bare Pass

- ❑ All the constructs and features above AND
- ❑ 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 when the person asks a question it looks if the person's answer contains a specific 'trigger' word (ie a word the program knows a response for). That trigger word and a response to it is stored as a record. The record includes a field for the trigger and a field for the response (eg if "greenfly" is in a question that would lead an answer "Encourage ladybirds." then 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 look for the response and gives it.

The program also uses a loop so that when the question is answered it invites the user to ask another until someone types "Goodbye" at which point the program ends.

Continued overleaf

Level 4 - 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 trigger word-response pairs. The trigger word records are stored in an array. This array should be initialised by a special method. The array is now passed as argument to method(s) that need the information. 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 5 - 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 if the answer given helped. If the answer is no it looks for a different matching trigger word and keeps doing so until the answer is yes or it runs out of possibilities. The trigger response array is embedded in its own abstract data type, of type QuestionAnswerDatabase and accessed by a set of methods defining clear operations available on the database. There should be no accessor method for illegal operations on the question bank.

Level 6 - 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 count of how often they were useful, updated each time the given trigger is detected. When the program first runs it sorts the trigger records in order of most useful and prints a list of the triggers with their usefulness count. From then on they are checked in that order with the more common words checked before less common ones.

Level 7 – 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 provides a choice at the start of the program running to either answer questions or input data. If data is to be input then the user inputs a new set of triggers and answers and stores them in a file. Based on an early question about the person's interests, the program now loads in a different file of triggers and responses (eg if they say they want to know about gardening it loads in a file of gardening triggers, but if instead they ask about cooking then a file specifically for that topic is loaded). 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.

Open Book Tests

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 2xA-F)

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 handbook 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.

Exit test (assessed 2xA-F – must be passed to pass coursework)

You must also take a written exit 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 exit test involves, though the end of term test covers the first and second part of the course (ie **The exit test is on Workbook Parts 1 and 2 only**) and so has questions on harder concepts.

To pass the exit 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 exit test or you will get a fail mark for the coursework.

The mid term and exit 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

January Final Assessment / "Exam": assessed: must pass to pass module)

You must do a written final assessment / exam in January. You must pass this to pass the module. 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 at a maximum of 39%. 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.

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 the online Notebook version of it - see the ASSESSMENT folder.

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

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: Your last lab, Tuesday 8 December 2020

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 **FINAL** literate program version 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, Wednesday 9 December 2020

If it is not signed off in the online master record you have NOT got the grade.