ECS401 Procedural Programming
Assessment Booklet: September 2016

Learning Outcomes
On passing the module you will be able to:

- 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 programming constructs and argue issues related to programming

The assessment, consisting of coursework and an exam, aims to develop these skills and test that you have gained them.

Coursework
Part of the assessment for this module consists of coursework during the term, made up of a series of components including programming exercises and tests.

Note that the exit test is likely to be ***in the last week of term (it could even be the last day)*** *and if you do not take it you fail the coursework (0% overall) so* ***DO NOT book travel home before term ends.***

> **Aim:** The aim of the coursework is that if you do it 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.

Exam
There is also an end-of-year written exam in the summer.

> **Aim:** The aim of the exam is to ensure you have gained those skills developed by doing the coursework and other exercises. This includes both programming and written theory skills.

Gaining the skills developed on this module is vital for later modules you will take. 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 exam (gain 40% or better) and your combined exam and coursework mark must also be over 40%. **If you fail the exam your mark is capped at 39%**

Students who do little work on the coursework fail the exam and so the module. Those who put in effort to learn throughout term by working hard on the coursework FROM THE START generally pass. The non-assessed exercises are designed so that if you do them you will learn the skills.

The coursework requires passing a series of separate components, including both programming and written work.

The coursework consists of the following components

- A series of short programming exercises (worth 1 A-F grade)
- A programming mini-project (worth 2 A-F grades)
    - A mid-term test (similar in question style to the exam) (worth 2 A-F grades)
- An exit test (similar in question style to the exam) (worth 2 A-F grades)

YOU MUST PASS THE EXIT TEST TO PASS THE COURSEWORK

This gives you 7 A-F grades at the end of the module. This is turned in to a single coursework mark as overleaf.

Calculating your Final Coursework Grade

Your overall grade for the coursework is calculated as follows

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
4A+s and 2A and 1 other (or better)	Grade A+	85%	PASS
4As and 2B and 1 other (or better)	Grade A	75%	PASS
4Bs and 2C and 1 other (or better)	Grade B	65%	PASS
4Cs and 2D and 1 other (or better)	Grade C	55%	PASS
6Ds and 1 other (or better)	Grade D	45%	PASS
4Ds and 2Fs and 1 other (or better)	Grade F	30%	FAIL
6Gs and 1 other (or better)	Grade F-	15%	FAIL
Otherwise (including missing component(s))	Grade Q	0%	FAIL

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

If you fail the exit test you will fail.

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 exam conditions). However we have added some flexibility allowing you to do badly on some – an odd nightmare day is allowed! The exception to this is to get an A++ ("I'm perfect") grade: you do have to be perfect.

The aim is to make you focus on gaining the skills – which is the point – not on just gaining marks (with or without skills) … and unless you gain the skills consistently you will not pass the exam.

Assessment Deadlines

For the purpose of this module weeks are counted as follows

Week	week start - end (mon - fri)	Major Deadlines (programs are marked week by week)
1	26 Sept - 30 Sep	
2	3 Oct - 7 Oct	Short 1 marked no later than YOUR allocated lab this week
3	10 Oct - 14 Oct	Short 2 and Mini F- - marked no later than YOUR allocated lab this week
4	17 Oct - 21 Oct	
5	24 Oct - 28 Oct	expected week of MID-TERM TEST (TBA-probably in your lab)
6	31 Oct - 4 Nov	
7	7 Nov - 11 Nov	Short 3,4 and Mini D marked no later than YOUR allocated lab this week
8	14 Nov - 18 Nov	
9	21 Nov - 25 Nov	***DON'T LEAVE IT TOO LATE TO GET PROGRAMS DONE AND MARKED!***
10	28 Nov - 2 Dec	
11	5 Dec - 9 Dec	AT MOST 2 MORE PROGRAMS CAN BE MARKED THIS WEEK
12	12 Dec - 16 Dec	expected week of EXIT TEST (To be confirmed but possibly wednesday pm - so please keep it free. An outside possibility is that it is on the friday so don't go home early!) AT MOST 2 MORE PROGRAMS CAN BE MARKED THIS WEEK

Coursework deadlines /assessment are in the following weeks:

Short programming exercises	(marked only in your allocated Lab)	week1-12
Miniproject programming exercises	(marked only in your allocated Lab)	week1-12
Final deadline for first assessment of short programming exercises 1:	(by end of your allocated Lab)	week 2
Final deadline for first assessment of mini-project (any level) and short exercise 2:		week 3
Mid term test:	(Expected week: Timeslot to be determined) (probably in your allocated lab of this week)	week 5
Deadline for shorts 3 and 4 and mini-project to be marked:		week 7
Exit test:	(Expected week: Timeslot to be determined) (probably wednesday afternoon - keep it clear)	week 12
Final deadline for any remaining short programming exercises completed:	(by end of your allocated Lab)	week 12
Final deadline for assessment of mini-project: (by end of your allocated Lab)		week 12
Hand in paper copy of previously marked programming exercises		week 12

You must get your programs assessed in your allocated lab as you go along.

You are expected to get programs completed, tested and marked well before the above final deadlines. Aim to get at least one finished and marked each week to give yourself the best chance of a high mark.

Tutors will mark at most 2 programs for each student in weeks 11 and 12
i.e., each week 2 shorts OR 1 short and the miniproject (to any level including jumping multiple levels provided this isn't the first time it has been seen)

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 tutor before moving on to the next.

Before each program is marked by the tutor however, explain it to and get it checked by a fellow student and have them sign your signature sheet (at the end of this booklet) in the student box.

You may have a particular program assessed more than once if it was not up to the standard required on the first attempt. Your tutor 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 your sheet and their record.

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 programming exercises on the website 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.

Testing your short programs

In industry most programming effort goes on testing – looking for mistakes in apparently working programs – not in writing those programs. The tutors are your clients and don't expect to be given faulty code to sign off! Your company also has a policy that style rules (comments, indentation, variable naming) must be adhered to. They have found it is vital if code is to be easily modified as on seeing the code, clients often think of extras to be added. When testing you should also check the style of the code too.

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.

Short Assessed Programming Exercise 1: Output

To pass this exercise you must demonstrate that you are able to

- ❑ **Write a simple program that compiles and runs**
- ❑ **Write a program that correctly uses output instructions**
- ❑ **Write a program split in to methods**
- ❑ **Include simple comments – at least the actual authors name and date**

*When your program works, explain it to a fellow student, have them test it and check it ticks all the boxes, and sign your signature sheet **at the end of this booklet**, then have it assessed by a tutor. If you are having problems 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 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**
- ❑ **Double check all comments make sense**

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

Short Assessed Programming Exercise 2, Input, Calculation and Variables

To pass this exercise you must demonstrate that you are able to

- ❑ **Write a program that uses input instructions**
- ❑ **Write a program that uses variables**
- ❑ **Write a program that uses arithmetic expressions**
- ❑ **Write a program that is split in to methods that return results**
- ❑ **Write a program that makes simple use of comments – at least the authors name and date and explanation of what the program does overall**

When your program works, explain it to a fellow student, have them test it and check it ticks all the boxes, and sign your signature sheet ***at the end of this booklet****, then have it assessed by a tutor. If you are having problems ask for help.*

Fitness Age

A fitness app includes a feature where it works out your "fitness" age as a way of indicating whether you are fitter than your actual age or not. Write a program to do a similar thing. The user enters their scores out of 40 for two exercises (the lower the score the fitter they are). Their "fitness" age is calculated from these two numbers as follows: First work out the average score. (By adding them together and dividing by two). Next multiply the average by 8. Divide the answer by 5. Finally add 10. The answer (in whole years) is the person's PC fit age. Write methods to get the two scores from the user. The calculation should be done by another method that calls the input methods and returns the result.

Print out the person's average score, fitness age and its difference from their real age. This should be done by another method. All answers should be in whole numbers (rounded down).

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

```
Score for fitness test 1? 11
Score for fitness test 2? 9
Your average score is 10
Your PC Fit age is 26 years old.
What is your actual age? 20
You are -6 years away from your PC fit age.
```

Another example run:

```
Score for fitness test 1? 23
Score for fitness test 2? 42
Your average score is 32
Your PC Fit age is 61 years old.
What is your actual age? 65
You are 4 years away from your PC Fit age.
```

You may want to take one of the example programs from the course web page as your starting program – and modify 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 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, checking all variables are given a value before the value is used, is always a good idea. Check the comments make sense.

Short Assessed Programming Exercise 3: Making Decisions

To pass this exercise you must demonstrate that you are able to

- ❑ **Write a program that uses if-then-else constructs,**
- ❑ **Write a program that is split in to methods that return results**
- ❑ **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, explain it to a fellow student, have them test it and check it ticks all the boxes, and sign your signature sheet, then have it assessed by a tutor. If you are having problems ask for help.

Underground zones Write a program that tells people the underground zone a particular station is in. If the program does not recognize the station it should just say so but it should know at least 5 stations in different zones. It should give the zone as a number between 1 and 6. Each separate station the program knows about should have a method that is called if it is typed in. It should return an integer giving the zone of that station.

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 station do you need to know the zone of? Mile End
Mile End is in Zone 2
```

Another example run:

```
What station do you need to know the zone of? Liverpool Street
Liverpool Street is in Zone 1
```

Another example run:

```
What station do you need to know the zone of? Woodford
Woodford is in Zone 4
```

Another example run:

```
What station do you need to know the zone of? Oxford Street
Is Oxford Street a London Underground Station? Maybe check your
spelling.
```

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 constructs 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 several times with different values choosing values that will definitely test all the lines of code (so 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 special 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, dry running it. This is called doing a "Programmer's Walkthrough". Does it appear to do the right thing on paper!

Remember, programming can be tricky at first, but …
you are capable of learning to program if you keep at it.

The more programs you write the easier it gets. Do the programming exercises from the workbook will help as they take you through the concepts gradually.

Short Assessed Programming Exercise 4: Records

To pass this exercise you must demonstrate that you are able to
- ❑ **Write a program that defines and uses records.**
- ❑ **Write a program that uses getter and setter methods that take arguments and return results**
- ❑ **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, explain it to a fellow student, have them test it and check it ticks all the boxes, and sign your signature sheet, then have it assessed by a tutor. If you are having problems ask for help.

Art Gallery Write a program that gives information about paintings in an art gallery for visitors. It holds details of the most popular painting in each of the four rooms that can be visited in any order. The visitor types in the room number and is give a short description of where the painting is in the room, the title, the artist's name and the year it was painted and its dimensions. A new type called Painting should be created (a record type) and each separate piece of information (direction, artist, title, year, height and width) about a painting should be stored in a separate field of the record accessed using getter and setter methods you write.

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 if you wish):

```
What room are you in? 3
The painting on the left is by Olga Boznanska. It was painted in 1894
and is called Girl with Chrysanthemums. It is 88.5cm by 69.0cm.
```

Another example run:

```
What room are you in? 1
The painting ahead of you is by Mary Cassatt. It was painted in 1879 and
is called Woman with a Pearl Necklace in a Loge. It is 81.3cm by 59.7cm.
```

Another example run:

```
What room are you in? 2
The painting ahead of you is by Rembrandt. It was painted in 1659 and is
called Self-Portrait with Beret and Turned-Up Collar. It is 84.5cm by
66.0cm.
```

Another example run:

```
What room are you in? 4
The painting on your right is by Claude Monet. It was painted in 1872
and is called Impression, Sunrise. It is 48.0cm by 63.0cm.
```

TESTING Short Assessed Programming Exercise 4: 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**
- ❑ **All branches of the IF-THEN-ELSE constructs work**
- ❑ **All FIELDS of any RECORD are set and accessed correctly**
- ❑ **The program deals with input it doesn't know about**

For programs with records you need to be sure the fields are both set and accessed correctly. By creating getter and setter methods for each field you can check each method works independently of other fields.

Testing is easier if you write a special test method adding tests for new methods as you write them.

Short Assessed Programming Exercise 5: For Loops

To pass this exercise you must demonstrate that you are able to

- ❑ **Write a program using FOR loop constructs.**
- ❑ **Define and use methods including ones that take an argument and return a result**
- ❑ **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.**

When your program works, explain it to a fellow student, have them test it and check it ticks all the boxes, and sign your signature sheet, then have it assessed by a tutor. If you are having problems ask for help.

Paralympic swimming relay

A new Paralympic relay competition involves competitors of different disability levels making up a team. Each runner's disability is rated on a 10-point scale and the total points rating for the 4 runners must be no greater than 32 points. For example, a legal team might consist of a runner in each of disability classes 6, 7, 9 and 10.

Write a program that **uses a for loop** to work out the total disability point score for a single team, and check that it is legal. **HINT: Have a variable that keeps a running total.** The program must be split in to methods. One method should take the runner's number (1-4) as argument and return the disability class input. The program should include another method that calculates the legality of the team, returning a boolean answer, when given as an argument the total of adding the four disability classes.

The following is an example run of the program

```
What is the disability class of Runner 1? 10
What is the disability class of Runner 2?  7
What is the disability class of Runner 3?  7
What is the disability class of Runner 4?  6
That team has 30 points so is legal
```

The following is another example run of the program

```
What is the disability class of Runner 1? 10
What is the disability class of Runner 2?  8
What is the disability class of Runner 3?  8
What is the disability class of Runner 4?  9
That team has 35 points so is NOT legal
```

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

When your program works and has been tested have it assessed. If you are having problems talk to your program tutor about it. If you keep trying you will get better at programming.

TESTING Short Assessed Programming Exercise 5: For Loops

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

- ❑ **The program runs correctly for all input.**
- ❑ **Running totals are correct at all points through the loop**
- ❑ **The program uses a for loop construct and does the correct number of iterations**

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..

Short Assessed Programming Exercise 6: Arrays

To pass this exercise you must demonstrate that you are able to
- ❑ **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.**

When your program works, explain it to a fellow student, have them test it and check it ticks all the boxes, and sign your signature sheet, then have it assessed by a tutor. If you are having problems ask for help.

Endangered Animals Write a program that uses a for loop to repeatedly name an animal and asks the user to say how many are left in the wild. It should then name the animal that is *most* endangered. The program should then print out the information in a comma separated list form (suitable for spreadsheet input). It should ask about the five animals: the Komodo Dragon, the Manatee, the Kakapo, the Florida Panther and the White Rhino (see below). The information about the animals should be kept in one or more arrays.

```
Komodo Dragon:
How many are left in the wild? 5000

Manatee:
How many are left in the wild? 8000

Kakapo:
How many are left in the wild? 91

Florida Panther:
How many are left in the wild? 100

White Rhino:
How many are left in the wild? 18

The most endangered animal is the White Rhino.
There are only 10 left in the wild.

5000, Komodo Dragon
8000, Manatee
91, Kakapo
100, Florida Panther
18, White Rhino
```

HINT: Have a variable keep track of the most endangered so far.

TESTING Short Assessed Programming Exercise 6: 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 (visiting 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.

Short Assessed Programming Exercise 7: While Loops

To pass this exercise you must demonstrate that you are able to

- ❑ **Write a program using WHILE loop constructs.**
- ❑ **Use indentation well.**
- ❑ **Provide helpful comments in the code.**
- ❑ **Use variable names that give an indication of their use.**
- ❑ **Use final variables to store literal values rather than them appearing through the code.**

When your program works, explain it to a fellow student, have them test it and check it ticks all the boxes, and sign your signature sheet, then have it assessed by a tutor. If you are having problems ask for help.

GARDEN BIRD WATCH Write a program that is to be used as part of a garden bird watch conservation project where people watch for birds in their garden. It should use a while loop to repeatedly ask for the user to name a bird and then ask how many are seen at once. It should stop when the special code END is entered, and name the bird that *most* were seen of at one time in that garden.

```
Which bird have you seen? Robin
How many were in your garden at once? 1

Which bird have you seen? Magpie
How many were in your garden at once? 2

Which bird have you seen? Blue Tit
How many were in your garden at once? 5

Which bird have you seen? Kestrel
How many were in your garden at once? 1

Which bird have you seen? Robin
How many were in your garden at once? 2

Which bird have you seen? END

You saw 5 Blue Tits.
It was the most common bird seen at one time in your garden.
```

Make sure you comment your program with comments that give useful information, use indentation consistently and that your variable names convey useful information. **When your program works and has been tested have it assessed. If you are having problems talk to your program tutor about it.**

TESTING Short Assessed Programming Exercise 7: 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 by doing the loop body just once, or doing it no times at all. Doing dry run/programmer's walkthroughs can be especially important. Make sure loops can't run forever for any input including odd values input.

Remember, programming can be tricky at first, but …
you are capable of learning to program if you keep at it.

Short Assessed Programming Exercise 8: Methods and Abstract Data Types

To pass this exercise you must demonstrate that you are able to

- ❑ **Write programs consisting of multiple methods (here at least 9 others as well as main).**
- ❑ **Create an abstract data type with fields accessed only by getter and setter methods**
- ❑ **Declare functions, call them with multiple arguments and return values from them.**
- ❑ **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.**
- ❑ **Use final variables to store literal values rather than them appearing through the code.**

When your program works, explain it to a fellow student, have them test it and check it ticks all the boxes, and sign your signature sheet, then have it assessed by a tutor. If you are having problems ask for help.

Bonus Scheme Write a program that works out the amounts to pay out in a digital media company's bonus scheme. Each employee is rated out of 10 by the bonus committee on two factors: profit the person made and a rating of how hard they work. Life is tough in digital media, but as with most things the harder you work the better you do. In this company if things you try don't work its good as long as you keep trying. Your program converts these ratings in to a performance score and each bonus is calculated from it.

You should:
Write an abstract data type for an employee consisting of two numbers (a profit score and hard work score)
Write a method that given a number (a profit score) **as an argument** returns the result of multiplying it by 2.
Write a second that given a number (hard work score) **as argument** returns the result of multiplying it by 5.
Write a third method that given an employee record (holding a profit score and hard work score) **as argument**, calls the above two methods in turn, then adds the answers to get a score out of 70 then divides it by 7 to give a final **performance scor**e (an integer) out of 10 that is **returned** to the calling method.
Write a fourth method that given a number (a performance score) multiplies it by 5,000 to determine the bonus to be paid.

Both initial scores should be read into three separate variables by a separate method (itself called by main). It should call the performance score method as described above to calculate a performance score and print this result out. This method should then call the bonus method to calculate the bonus from this performance score. This method should then print the bonus.

Of those above only the last method should input or print anything. You must NOT use global variables except as final variables. All information needed by a method MUST be passed to it. All the above methods just do calculations. You should break the program in to other methods as appropriate too.

An example run might be

```
Profit Score?  5
Hard Work Score? 4
Your performance score this year is 4 out of 10.
Your bonus is 20,000 pounds.
```

When your program works have it assessed.

TESTING Short Assessed Programming Exercise 8: Methods / abstract data types

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 getter and setter 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 and checking they are correct.

Short Assessed Programming Exercise 9: Arrays of Abstract Data Types

To pass this exercise you must demonstrate that you are able to

- ❑ **Write programs consisting of arrays of program-defined abstract data types**
- ❑ **Write programs with records accessed via setter and getter methods as part of the record class**
- ❑ **Write programs consisting of multiple methods, with multiple arguments some returning results.**
- ❑ **Write functions that are passed arrays as arguments and return results within an array.**
- ❑ **Write a program that uses a loop and array**
- ❑ **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.**

When your program works, explain it to a fellow student, have them test it and check it ticks all the boxes, and sign your signature sheet, then have it assessed by a tutor. If you are having problems ask for help.

Film Box Office Write a program that **uses one or more arrays processed using for loops to** display information about the films at a cinema. The information about each film should be stored as **a new abstract data type** (so using setter and getter methods), consisting of the name and start time of the film. At the start of the day the program should first gather the names and start time of the four films to be shown that evening, storing it in an array of the abstract data type. The program should then use a second for loop to access the arrays and print the information to the screen (to be displayed on the box office screen). The data entered about the films should be stored in an array of the abstract data type created, and then printed out from the array. The following shows one example run of the program:

```
Film for Screen 1? Now You see Me 2
What time does it start? Hour  7
What time does it start? Minutes after the hour 30
Film for Screen 2? Spectre
What time does it start? Hour  6
What time does it start? Minutes after the hour 30
Film for Screen 3? The BFG
What time does it start? Hour  7
What time does it start? Minutes after the hour 15
Film for Screen 4? Eddie the Eagle
What time does it start? Hour  6
What time does it start? Minutes after the hour 0

CinemaWorld Films Tonight
Screen 1: Now You See Me 2                7:30
Screen 2: Spectre                         6:30
Screen 3: The BFG                         7:15
Screen 4: Eddie the Eagle                 6:00
```

Make sure you split the program in to multiple methods, 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 9: Methods with 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.**
- ❑ **Each method individually works**
- ❑ **Each abstract data type is correctly implemented**

The same points apply as for testing methods above (and arrays).

Programming mini-project (assessed 2xA-F)

You must write a mini-project 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. **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. Once your program has reached a level (see below) you should get it checked by a fellow student and then marked by a tutor. If you are confident then you may wish to skip getting some levels marked by the tutor.

Start early!

For your grade to count, your mini-project **MUST** be marked

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

The level achieved **MUST** be confirmed in writing on your cover sheet.

Different tutors 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 tutor that you have met the learning outcomes not just for presenting a correct program.

Whole programs submitted at the end of the term for which intermediate stages 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 the precise thing your program does in the example.

All students' miniproject 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: Multiple choice Music Quiz program

Write a multiple choice quiz program about music for teams to play.

Level 1 – F minus minus minus: Getting started

- ❑ Includes **Screen output, Keyboard input**
- ❑ **Defines and uses a method doing a** well-defined task
- ❑ Comment gives at least authors name at start of program.

Example:
The program prints out a question and allows the person to type the answer they believe to be correct. The program always prints the same message saying "That's an intriguing answer!" A method is called to print the question. A separate method prints the final message.

Level 2 – F minus minus: Making progress

- ❑ All the constructs and features above AND
- ❑ Includes **variables, assignment and expressions**
- ❑ **Defines and uses a method that returns a result**
- ❑ Comments gives at least authors name and what the program as a whole does.

Example:
As above, but a score is kept for the person. Right or wrong the person is given a score just for answering the question. A method is defined to randomly roll a dice to decide the score given. The method returns the score awarded. The question is asked by a specially defined method that returns the answer given.

Level 3 - F minus: Getting there

- ❑ All the constructs and features above AND
- ❑ Includes **Decision statements**.
- ❑ **Defines and uses methods including one(s) that take arguments and return a result**
- ❑ Comments included (may be poor giving little additional information over the lines they comment)
- ❑ Indentation attempted (it may be inconsistently applied)

Example:
As above, but the person is now told the correct answer and whether they got it right or not. They roll the dice only if they got it right. If they roll 1-5 they get 3 points added to their score. If they roll a 6 they get 6 points. A separate method is passed the person's answer and the correct answer to the question as arguments and returns a boolean result of whether it was right or not. A different method asks the player for their answer.

Level 3 - F: Almost a pass

- ❑ All the constructs and features above AND
- ❑ Includes **records**.
- ❑ **Defines and uses getter and setter methods**
- ❑ Comments included are helpful and comment blocks of code (though if too many are given that is ok).
- ❑ Indentation consistently applied throughout

Example:
As above, but now the question and correct answer are stored as a record accessed only by getter and setter methods. It is passed as a single argument to the methods that ask the question and check the person's answers.

Level 4 - D: Bare Pass

- ❑ All the constructs and features above AND
- ❑ Includes **Loops**
- ❑ **Well-structured in to multiple methods using multiple arguments and returning 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.

Example:
As above but the program uses a loop to allow a team of people (the number of which is input at the start) to be separately asked the same question and be told their combined team score on the question at the end. A method is defined that is passed as arguments: the team's current score (initially 0) and dice roll. It returns the new score.

Continued overleaf

Level 5 - C: Pass

- ❑ All the constructs and features above AND
- ❑ Includes **Arrays**
- ❑ 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 within blocks that reduce scope to a minimum – no use of global variables.

Example:
As above but the program also now stores the current score of each team member in an array with one entry per player. The program gives both team and individual scores for each player at the end. This array should be initialised by a special method. The arrays are now passed as arguments to methods that for example calculate the new score of a player and update the array.

Level 6 - B: Satisfactory

- ❑ All the constructs and features above AND
- ❑ Includes **Loops within loops**.
- ❑ Includes **abstract data types**.
- ❑ 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:
As above but the program now has a series of rounds (so a series of different questions) with each player taking a turn in each round, controlled by a loop. The question-answer records are now stored in an array but one that is embedded in its own abstract data type. This abstract data type is initialised at the start of the program using method calls (eg a constructor) and is passed to methods that use the question and answers. The program uses a loop to control the rounds. The program also keeps track of each players scores for each round that can be printed out on request. The program is structured into sensible 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

Example:
As above but includes an option to print a table of the questions in order of how easy they were (ie how many points scored). This table should be printed in sorted order with easiest question first. The question-answer record should now include an extra field of the number of people who got that question right so far. The question-answer array should be embedded in an abstract data type presenting it as a question bank. 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
- ❑ **file input AND file output**

Example:
Allows the current total quiz 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.

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. Make it a program someone would really want to use!

Suggestion 2: Dinosaur Pet program

Write a program that simulates bio-engineered dinosaur pets that you must look after.

Level 1 – F minus minus minus: Getting started

- ❑ Includes **Screen output, Keyboard input**
- ❑ **Defines and uses a method doing a** well-defined task
- ❑ Comment gives at least authors name at start of program.

Example:
The program asks you to name your dinosaur pet and then prints a message using the name (eg Happy 0th Birthday Bill the Brontosaurus) for a pet named Bill. A separate method is called at the start that explains what the program is for. A different method is called to ask you to name the pet.

Level 2 – F minus minus: Making progress

- ❑ All the constructs and features above AND
- ❑ Includes **variables, assignment and expressions**
- ❑ **Defines and uses a method that returns a result**
- ❑ Comments gives at least authors name and what the program as a whole does.

Example: As above, but the pet can be thirsty. A method is written to return the thirst level. A random number determines how thirsty. A message is printed depending on the number (eg "Bill's thirst level is 9/10"). The method used to get the name of the pet, now returns the name given for use elsewhere in the program.

Level 3 - F minus: Getting there

- ❑ All the constructs and features above AND
- ❑ Includes **Decision statements**
- ❑ **Defines and uses methods including one(s) that take arguments and return a result**
- ❑ Comments included (may be poor giving little additional information over the lines they comment)
- ❑ Indentation attempted (it may be inconsistently applied)

Example: As above, but the pet now also has a name and species stored as well as a hunger score. The thirst and hunger scores are turned into a level of anger – eg 1: "serene", 2: "grouchy" or 3: "dangerous" depending on how large the combined score is. A method is defined that returns this anger level given the thirst and hunger scores as arguments. Other methods print out appropriate messages depending on the thirst and anger scores (eg "Bill is looking dangerous...GET OUTA THERE NOW!! Your pet is being put down for everyone's safety").

Level 3 - F: Almost a pass

- ❑ All the constructs and features above AND
- ❑ Includes **Records**
- ❑ **Defines and uses getter and setter methods**
- ❑ Comments included are helpful and comment blocks of code (though if too many are given that is ok).
- ❑ Indentation consistently applied throughout

Example: As above, but now the name, species, hunger and thirst values (along with any other information about the individual pet) are stored as a pet record accessed using getter and setter methods. The record is passed as a single argument to the methods that use and change the information.

Level 4 - D: Bare Pass

- ❑ All the constructs and features above AND
- ❑ Includes **Loops**
- ❑ **Well-structured in to multiple methods using multiple arguments and returning 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.

Example: As above but the program uses a loop to allow time to pass - ie a series of rounds to be played. The pet has a 'state of mind' given by a whole series of different numbers (eg hunger, thirst and irritability) that affect its anger level). It can be looked after in several ways – eg fed (lowers the hunger score), sung to (lowers the irritability score). On each round the different state of mind elements are changed by a series of random numbers. A single method is used to change each aspect of the state of mind.

Continued overleaf

Level 5 - C: Pass

- ❑ All the constructs and features above AND
- ❑ Includes **Arrays**
- ❑ 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 within blocks that reduce scope to a minimum – no use of global variables.

Example:
As above but the program also records the emotional state of the pet at each time step (up to some limit like 5 time steps) in an array of records, allowing the game to be restarted at an earlier time and continued from that point, eg if a pet has been put down due to excessive anger it can be resurrected. This array should be initialised by a special method. It is passed to methods that change the pet's state of mind (for example) allowing the state to be recorded.

Level 6 - B: Satisfactory

- ❑ All the constructs and features above AND
- ❑ Includes **Loops within loops**
- ❑ Includes **abstract data types**.
- ❑ 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:
As above but the program now allows multiple pets to be kept, each with its own current state of mind. The details of all pets are stored in an array with one entry per pet. The player must choose a pet to attend to on each round. The array of pet details array is embedded in an abstract data type (representing a pet menagerie) with implementation details hidden. This abstract data type is initialised at the start of the program using method calls (eg a constructor) and is passed to methods that use the question and answers. Pets are also stored as an abstract data type.

On each time tick, the player must choose a single pet to look after. Then every pet's emotional state is changed using a loop. The program is structured into sensible methods. Loops are used both to control the time/rounds and to work through each pet in turn to update its state. You win if all pets are serene at once. reaching a state of Dinosaur Nirvana.

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

Example:
As above but includes a way to list the pets sorted by anger, allowing the player to choose which to look after in this round, depending on how critical they 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
- ❑ **file input AND file output**

Example:
As above but also allows the current pet menagerie state to be saved into a file AND reloaded from a file so the program can be quit and continued later without having to start again from the beginning. Design a suitable file format to allow this to be done easily.

Now carry on and make your program really **something special**, using other more advanced constructs or algorithms 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. Make it a program someone would really want to use!

Suggestion 3: Olympic Information and Medal Table Program

Write a program for TV broadcasters to keep track of the Olympic Information including the medal table.

Level 1 – F minus minus minus: Getting started

- ❑ Includes **Screen output, Keyboard input**
- ❑ **Defines and uses a method doing a** well-defined task
- ❑ Comment gives at least authors name at start of program.

Example:

The program prints out a question asking if you want to see how many medals we've won. The program always prints the same message giving the final GB (or country of your choice) medal total

	G	S	B	Total
Great Britain	27	23	17	67

A method is called to print the question. A separate method prints the medal totals.

Level 2 – F minus minus: Making progress

- ❑ All the constructs and features above AND
- ❑ Includes **variables, assignment and expressions**
- ❑ **Defines and uses a method that returns a result**
- ❑ Comments gives at least authors name and what the program as a whole does.

Example:

As above, but the gold, silver, bronze and total numbers are stored in variables. The program separately and in turn asks the user how many people won a Gold, a Silver and then a Bronze today. Each question is asked by a separate method that returns the answer. These values are added to the previous total for each medal. The program adds the separate totals to work out the overall total number of medals.

Level 3 - F minus: Getting there

- ❑ All the constructs and features above AND
- ❑ Includes **Decision statements**
- ❑ **Defines and uses methods including one(s) that takes argument(s) and return a result**
- ❑ Comments included (may be poor giving little additional information over the lines they comment)
- ❑ Indentation attempted (it may be inconsistently applied)

Example:

As above, but after inputing new medals won, the program asks which medal total the person is interested in (Gold, Silver, Bronze) and prints only that. A further alternative given is to print everything as before. A separate methods is now used that is given old and new numbers of medals for the day (whether gold, silver or bronze) and returns the new total for that medal. A separate method is written that, given the gold, silver and bronze totals, returns the total number of medals.

Level 3 - F: Almost a pass

- ❑ All the constructs and features above AND
- ❑ Includes **Records**
- ❑ **Defines and uses getter and setter methods**
- ❑ Comments included are helpful and comment blocks of code (though if too many are given that is ok).
- ❑ Indentation consistently applied throughout

Example:

As above, but now the country name, number of golds, silvers and bronzes (and any other information about a single country) are stored as a record which is passed as a single argument to the methods that change and access the values.

Level 4 - D: Bare Pass

- ❑ All the constructs and features above AND
- ❑ Includes **Loops**
- ❑ **Well-structured in to multiple methods using multiple arguments and returning 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.

Example:

As above but the program uses a loop to allow the medal data to be updated and queried daily.

Continued overleaf

Level 5 - C: Pass

- ❑ All the constructs and features above AND
- ❑ Includes **Arrays**
- ❑ 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 within blocks that reduce scope to a minimum – no use of global variables.

Example:
As above but the program now stores medal data from the end of each day in an array. The program allows the user to print out the details from any past day, or of all days completed. This array should be initialised by a special method. The array is passed as an argument to all methods that need to access or update it.

Level 6 - B: Satisfactory

- ❑ All the constructs and features above AND
- ❑ Includes **Loops within loops**
- ❑ Includes **abstract data types**.
- ❑ 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:
As above but the program now keeps track of the data for multiple countries. The medal records are now abstract data types and are now stored in an array but one that is embedded in its own separate abstract data type of 'medal table'. This abstract data type is initialised at the start of the program using method calls (eg a constructor) and is passed to methods that use or change the data. The program uses a loop to print a full medal table (though in no particular order). The program is structured into sensible methods.

Level 7 - A: Merit

- ❑ All the constructs and features above AND
- ❑ Includes a **sort algorithm**
- ❑ 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

Example:
As above but includes an option to print the medal table in order of total medals won. (Optionally a slightly harder alternative is to give a choice of printing the table in terms of most golds,with ties sorted by most silvers, and ties then sorted by most bronzes.) The country abstract data type should be able to return the totals (either because it is stored as a field or because it is calculated when needed but which way is used should not be visible to the rest of the program). 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
- ❑ **file input AND file output**

Example: Allows the daily medal table data 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.

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. Make it a program someone would really want to use!

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 be taken in exam conditions. It will consist of exam-style questions covering the first part of the course (Part 1 of the handbook).

Examples of the kinds of question are included on the module QMplus 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, explain concepts in your own words, analyze what a program 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.

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

You must do a written exit test at the end of term. 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 on the module Qmplus 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 (Workbook Parts 1 and 2) and so has questions on more difficult concepts.

It 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
- test and debug programs to ensure they work correctly
- explain programming constructs and argue issues related to programming

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 constructs.

You must take and pass the exit test or you will get a fail mark for the coursework.

Exam (assessed – must be passed to pass module)

You must do a written exam in the summer. You must pass this to pass the module. Examples of the kinds of questions are included on the module intranet site with each weekly unit. 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. You must understand all work covered on the module and demonstrate your programming and writing skills in exam conditions.**

It will test you have achieved the learning outcomes of the module:
On passing the module you will be able to:

- 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 programming constructs and argue issues related to programming

To pass the exam you MUST be able to write programs in exam conditions and 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%.

The good news is that if you have taken the coursework seriously - both practical and theory and been doing all the exercises available (not just the assessed ones) from the start of the year then you will be well-prepared for the exam.

Rules of Helping Others

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

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 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

Some ways of trying to help are unhelpful!

NEVER take over some one else's keyboard.

NEVER just give someone else solutions whether programs, program fragments or otherwise

NEVER pass code to someone else

NEVER give answers without giving understanding.

Some ways of trying to 'help' are just very stupid!

NEVER give other people copies of your programs.

ECS401 Assessed PROGRAM Cover Sheet

Module Leader: Paul Curzon

Student Name (BLOCK CAPITALS): _____

Student ID (from ID card) : _____

Lab Class Day / Time: _____

Short Level	Grade		Tester student's signature after checking	Tutor Date and signature	Mini-Proj level		Grade	Tester student's signature after checking	Tutor date and signature
1	F- - -	Output			1	Input / Output	F- - -		
2	F- -	Input			2	Assignment /Expression	F- -		
3	F -	Decisions			3	Decisions	F -		
4	F	Records			4	Records	F		
5	D PASS	For			5	Loops	D PASS		
6	C	Arrays			6	Arrays	C		
7	B	While			7	Loops in loops+ADT	B		
8	A	Methods +ADT			8	Sort+ADTs	A		
9	A+	Array+ ADT			9	File I/O	A+		
Final Grade					Final Grade				

This cover sheet should be handed in with a paper copy of each of the programs you had assessed. The programs should be handed in, securely stapled in the corner and not in any kind of binder. **This cover sheet MUST be clearly visible on the outside without turning pages**. It must contain at least:

- a cover sheet containing a student tester and lab tutor's signature for each program completed and assessed in the labs,
- a printout of each short programming exercise that has already been assessed by a tutor in the labs, and
- a printout of the **final** version of your programming mini-project, already assessed by a tutor in the labs.

All programs must have been assessed by your tutor in or before your last programming lab.

You must get the programs marked as you do them – the tutors are only required to mark at most 2 programs for each student in the last 2 programming labs so do not leave it until the last minute.

This must be handed in to the Computer Science Reception in the ITL by no later than:

12 noon, Wednesday 14 December 2016

If you do not have the signature on this sheet and in the tutors tick list, you have not got the grade.

-