Assessment

Assessment 2

Outcomes covered 3 and 4

Instructions for learners

This pack contains 10 programing tasks for you to choose from. You can do as many or as few as you want as long as your chosen tasks cover **ALL** of the practical skills listed in the table on the next page. You should use these tables to record your progress and give to your assessor along with all your evidence for each program.

Evidence for each program will consist of program code, evidence of successful execution and test documentation.

You should ensure that all your programs contain:

- subprograms
- parameter passing
- meaningful variable names
- clear user prompts for input
- validated input
- internal commentary, explaining each section of the code
- modular code

While you are implementing your programs you should carry out and provide evidence of each of the following:

- unit testing
- integration testing
- system testing

You should also produce a test plan with exceptional, extreme and normal data using your own examples of test data.

At some point in this Implementation and Testing Unit, you have to provide evidence of using the following debugging techniques.

- dry runs
- walkthroughs
- breakpoints
- trace tables

You only need one piece of evidence for each technique — this can either be from the same program or from different programs.

Learner’s name:	**Date:**

Programming skills required	**Tick when completed**	**Task number(s)**	**Assessor’s initials**
Create from subprograms			
Pass parameters			
Expressions			
Sequencing			
Selection			
Iteration			
Pre-defined function			
File handling (open, create, read, close)			
String variable			
Integer variable			
Real variable			
Boolean variable			
1D array			
Records			
Array of records			
Input validation			
Linear search			
Finding max/min			
Count occurrences			
Simple sort			

Testing skills required	**Tick when evidence done**	**Task number(s)**	**Assessor’s initials**
Unit testing			
Integration testing			
System testing			
Produce a test plan (exceptional, extreme, normal)			
Prepare own test data			
Debug using a dry run			
Debug using a walkthrough			
Debug using a breakpoint			
Debug using a trace table			

Assessor’s signature:	**Date:**

Task 1

Scottish Athletics is running trials for 400 metre athletes. There are four groups of eight athletes. The runners run two separate laps and their times are recorded as they complete each of the two laps. Any runner who completes either of the two laps in under 50 seconds progresses to the next stage of selection. The runner from each of the groups with the fastest lap will receive a medal of commendation.

Your task is:

- ♦ to write a program for **one** of the groups of female runners
- ♦ find the runners who will progress to the next stage
- ♦ find the runner with the fastest lap

Save this information to a file. You can assume that no two runners have the same name.

The program requires the following inputs:

- ♦ a valid number of runners
- ♦ the first and second name of each runner
- ♦ the time taken to run the first lap
- ♦ the time taken to run the second lap

The output from the program should display the fastest runner, each runner's name and indicate whether they will progress to the next stage of selection.

An example output is provided below:

```
  Fastest lap was completed by Jenny Smith in 48 seconds

         Runner Name            Progresses to Next Stage
           J Smith                        Yes
           A Runner                       No
           .....                          .....
```

Task 2

Implement and test a program which will allow users to complete a survey about local facilities. The survey has 15 questions with different possible responses — some numerical answers and some text answers. Only one response is possible for each question. Every question has an option of 'Do not know'. You should use a record structure to store details of the survey questions. There should also be an option in the program to calculate the total number of 'Do not know' responses.

You should test the program with five users.

Task 3

Implement and test a program to ask a user 10 addition questions using randomly created numbers between 10 and 30. The program should:

- check if each of the user’s answer is correct
- keep a total of the number of correct answers
- repeat the quiz until the user achieves more than seven correct

Task 4

Implement and test a program to input a sentence, apply a code to each character (except spaces) and output the coded message to the screen.

For example, A becomes Y, B becomes Z, etc and thus Hello World would be coded as Fcjjm Umpjb.

Task 5

Implement and test a program to assign grades to exam marks for a class of 15 students, using the following criteria:

- more than 70% is an A
- more than 60% is a B
- over 50% is a C
- more than 45% is a D
- less than 45% is a fail

The program requires the highest possible score for an exam, each student’s name (first name and surname) and each student’s mark.

The program should then:

- calculate the percentage mark for each student
- sort the percentages into order from highest to lowest
- display each student’s initials, percentage and grade in the sorted order
- display the total numbers of As, Bs, Cs, Ds and fails

An example output is shown below:

```
                    Class Results

                  KW       84%   A
                  SJ       56%   C
                           .
                           .

            Number of A passes          3
            Number of B passes          4
            Number of C passes          5
            Number of D passes          2
            Number of Fails             1
```

Task 6

Implement and test a program to calculate the number of points gained by five hockey teams, given the number of wins, draws and lost games, assuming a win is worth 3 points, a draw 1 point, and no points for a lost game.

You should create your own test data and output the results on screen, in a suitable format.

Task 7

Implement and test a program to create a list of 35 random, upper case letters and then stores the list in an array. You should then sort the list into order, A to Z, and output the character in the middle of the list.

Task 8

Implement and test a program to enter and validate a password. The program should do the following:

- Prompt the user to enter their password that can contain letters and numbers and be at least six characters long.
- If the password is correct, it should display a welcome message.
- If not, it should notify the user that their password was entered wrongly, they should be warned that they will be locked out of the system after three attempts and let them try again, but only allow three tries.
- If the user makes three incorrect attempts they should receive a suitable message.

Task 9

Implement and test a program in your programming language which does the following:

- ♦ Fills an array with 20 random integers between 1 and 10.
- ♦ Prints the array contents to screen.
- ♦ Finds and displays the maximum and minimum values in the array.
- ♦ Asks the user for an integer between 1 and 10 using input validation and displays how many times it occurs in the array.
- ♦ Asks the user for an integer between 1 and 10 using input validation and displays where in the array that number first occurs and indicates when it is not present.

Task 10

Implement and test a program to simulate a simple game. Your program should do the following:

- ♦ Store six colours in an external file (Note: these colours can be your choice of words or graphics).
- ♦ Create three arrays and input the colours to each of the arrays.
- ♦ For each of the three arrays, create a random number between 1 and 6 and output that colour to the screen.
- ♦ If two of the colours match, the user wins 50 pence.
- ♦ If all three colours match, the user wins £1.
- ♦ Allow the user to have five 'shots' and keep track of their winnings.
- ♦ Display suitable messages on screen.

An example output is shown below:

```
   RED        PURPLE        RED
         You have won £0.50
         Your total winnings are £1.50
         You have one shot remaining
```