Higher

Computing Science

H

Software Design and Development

Testing & Evaluation

NAME:______________________________

Contents

TESTING STAGE

The Testing Stage is necessary in order to identify and correct and errors in the source code.

It is important that a carefully considered test plan is created. It is **vital** that a test plan is produced **before** the solution is implemented to ensure the software is tested **systematically**.

The test plan includes:

- Details of what is to be tested
- Test data values
- Expected outputs
- Type of testing

TEST PLAN

The test plan should ensure that testing is **comprehensive.**

Comprehensive testing is when the program is tested as thoroughly and completely as possible.

Ideally, ***exhaustive testing*** is used where every possible input and route through the program is tested – but this is not always practical or possible.

Comprehensive testing should involve using a range of **normal, extreme** and **exceptional** test data

A test table is used to record tests carried out, expected results and actual results. A test table should use all three types of test data.

Expected results are worked out **manually** without using the program. The same inputs are then entered into the program to give the **actual results**. If both match then the test has passed.

Example

This program should accept three test scores between 0 and 50 and calculate the total and average.

Test No.	**Reason**	**Test Data**	**Expected Result**	**Actual Result**	**Test Result**
1	Normal Test	Score1: 34 Score2: 45 Score3: 29	Total: 108 Average: 36	Total: 108 Average: 36	PASS
2	Normal Test	Score1: 12 Score2: 19 Score3: 2	Total: 33 Average: 11	Total: 33 Average: 11	PASS
3	Extreme Test	Score1: 50 Score2: 50 Score3: 50	Total: 150 Average: 50	Total: 150 Average: 50	PASS
4	Extreme Test	Score1: 0 Score2: 0 Score3: 0	Total: 0 Average: 0	Total: 150 Average: 50	PASS
5	Exceptional Test	Score1: 65 Score2: 52 Score3: 90	Not Accepted	Total: 207 Average: 69	FAIL
6	Exceptional Test	Score1: -1 Score2: -40 Score3: -100	Not Accepted	Not Accepted	PASS

Test 5 has failed because the program has accepted invalid score inputs (all above 50) when it should have asked the user to re-enter.

ERROR TYPES

There are three types of error that can occur in a program.

- Syntax Error
- Execution Error
- Logic Error

Syntax Error

Syntax is the rules of how a programming language must be written.

If code is written that **breaks the programming language rules** then a syntax error occurs.

The program **will not run at all** if there is a syntax error in the code.

Examples

Example 1	*Example 2*	*Example 3*

FOR index 1 TO 10
 	
NEXT index
	DIM first IS STRING	IF first = 10 OR 20
		
		END IF

There is an equals sign missing after the word index. FOR **index = 1** TO 10 NEXT index	The word IS should be **AS** DIM first AS STRING	A complex condition must contain complete conditions on each side of the logical operator (OR). IF first = 10 OR **first =** 20

Execution Error

An execution error occurs when the program tries to do something impossible during while the program is running.

An execution error will **cause the program to crash** mid-way through. Usually an error message will appear explaining the problem.

Examples

Example 1	*Example 2*	*Example 3*
DIM myArray**(5)** AS INTEGER FOR **loops** 1 TO **10** myArray(**loops**) = 0 NEXT index	DIM score AS **INTEGER** **score** = “**Henry**”	DIM result AS INTEGER DIM value AS INTEGER **value = 0** result = 0 result = **50 / value**
Error: Trying to access an array position bigger than the size of the array. **Explanation:** The value of the **loops** variable will increase until it gets to 10. When it reaches 6, this will exceed the size of the array which only has 5 positions.	**Error:** Trying to assign string data to an integer variable. **Explanation:** The score variable has been declared as an Integer data type. When the program tries to store a string value in score, it cannot be done.	**Error:** Trying to divide by 0 which is impossible. **Explanation:** The value variable has been initialised to 0. When the program tries to divide 50 by the contents of value (0), it cannot be done.

Logic Error

A logic error occurs when the program runs normally, does not crash, but produces an incorrect output (the wrong answer).

A logic error will <u>**not**</u> cause the program to crash. These type of errors can be difficult to fix because **no error messages** are produced.

Examples

total = score1 ***** score2
 	
Msgbox(“score1 + score2 is ” & total)
	IF answer1 = “A” **AND** answer1= “B” THEN
	        result = “pass”
	ELSE
	        result = “fail”
	END IF

The expression multiplies the scores instead of adding them, giving the wrong output. The first line should be: total = score1 **+** score2	The complex condition (line 1) is checking for answer1 to contain A and B at the same time. It is impossible for a variable to contain both A and B so the result will always be FAIL. The first line should use an OR instead of AND: IF answer1 = “A” **OR** answer1= “B” THEN

DEBUGGING

Debugging is the process of finding and correcting errors.

Some types of error are easier to identify than others because the programming environment will help.

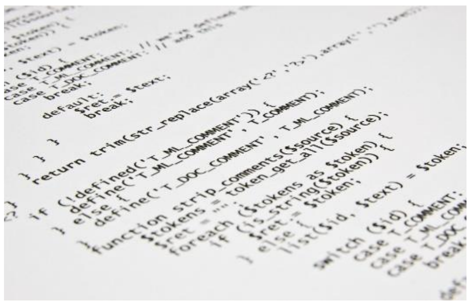
- The program will not run at all if there is a ***syntax*** **error**
- The program will stop running if an ***execution*** **error** is encountered

Logic errors are more difficult to find because the program will run but will produce incorrect results. There are a number of debugging techniques that can be used to identify logic errors.

- Dry Run
- Trace Table
- Trace Tools
- Breakpoints
- Watchpoints

Dry Run

A Dry Run involves **manually** stepping through each line of code using test data.



As lines of code that make changes to variables are reached, these changes are recorded using a **table**.

This should highlight positions in the code where variables are changing to unexpected values.

Trace Table

A trace table is similar to the table used to record variable values during a dry run.

Trace is often used to record the changes to variables when testing an algorithm for a specific sub-program.

A trace table allows the tester to check the result of a number of different values of a variable.

Line by Line recording

To record each variable change, line by line, the following steps should be followed.

- List each variable along the top.
- Work through the program line by line and record the changes in each line where a change is made to a variable.
- If the program loops, you must repeatedly work through the lines within the loop

Example

```
Line 1  FUNCTION findMinimum (ARRAY OF INTEGER list)
        RETURNS INTEGER
Line 2      DECLARE min INITIALLY list[0]
Line 3      FOR index FROM 1 TO length(list)-1 DO
Line 4          IF min < list[index] THEN
Line 5              Min = list[index]
Line 6          END IF
Line 7      END FOR
Line 8      RETURN min
Line 9  END FUNCTION
```

Line	List	Min	index
1	[4,5,2,1,3]		
2		4	
3			1
3			2
5		2	
3			3
5		1	
3			4

Breakpoint Recording

If a breakpoint is set at the end of a loop, it will break at the end of the first pass of the loop.

The program can then be continued. The breakpoint will cause the program to halt at the end of each repetition.

To record the values of variables each time a breakpoint is reached.

- List each variable along the top.
- Work through the program line by line and record the values of all variables every time the breakpoint is reached.

Example

```
  Line 1   FUNCTION findMinimum (ARRAY OF INTEGER list)
           RETURNS INTEGER
  Line 2       DECLARE min INITIALLY list[0]
  Line 3       FOR index FROM 1 TO length(list)-1 DO
  Line 4           IF min < list[index] THEN
  Line 5               Min = list[index]
  Line 6           END IF
● Line 7       END FOR
  Line 8       RETURN min
  Line 9   END FUNCTION
```

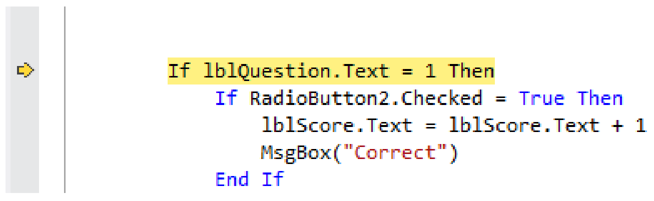
Line	List	Min	index
Break 1	[4,5,2,1,3]	4	1
Break 2	[4,5,2,1,3]	2	2
Break 3	[4,5,2,1,3]	1	3
Break 4	[4,5,2,1,3]	1	4

Breakpoint is shown at line 7

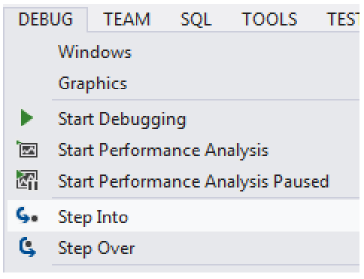
Trace Tools

Trace tools are a debugging feature of some programming environments.

Trace tools allow the program to be executed but the programmer can step through one line at a time.



```
If lblQuestion.Text = 1 Then
    If RadioButton2.Checked = True Then
        lblScore.Text = lblScore.Text + 1
        MsgBox("Correct")
    End If
```



This lets the programmer view the line of code being executed.

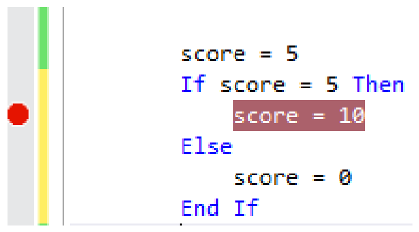
Breakpoints

Breakpoints are another debugging feature of some programming environments.

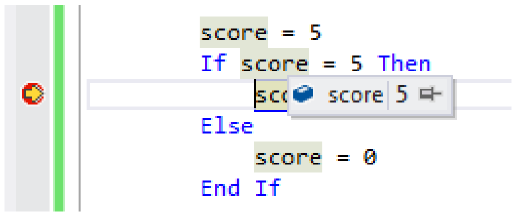
Setting a breakpoint sets **a point in the code where the program will stop execution**.

Breakpoints are set to stop executing at a particular line of code.

Once the program has stopped, the values of variables can be examined and recorded in a trace table.



```
score = 5
If score = 5 Then
    score = 10
Else
    score = 0
End If
```

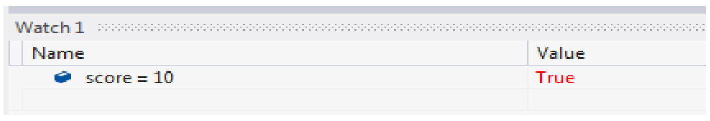


```
score = 5
If score = 5 Then
    score = 10
Else
    score = 0
End If
```

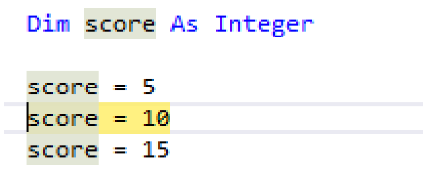
Watchpoint

A watchpoint is similar to a breakpoint but it does not depend on reaching a particular line of code.

Instead, the program is set to stop executing when the value of a variable changes.



Name	Value
score = 10	True



```
Dim score As Integer

score = 5
score = 10
score = 15
```

Again, once the program has stopped, the values of variables can be inspected and recorded in a trace table.

PRACTISE QUESTIONS

Question 1 (2019 Qu 10c)

A shop has a unique product code for each item it sells, for example X756. The linear search algorithm shown below is used to find the position of a product code in an array.

```
FUNCTION linearSearch (ARRAY OF STRING list)
RETURNS INTEGER
   DECLARE position INITIALLY 0
   DECLARE target INITIALLY ""
   SEND "Enter target value" TO DISPLAY
   RECEIVE target FROM KEYBOARD
   FOR index FROM 0 TO length(list)-1 DO
      IF target=list[index] THEN
         SET position TO index
      END IF
   END FOR
   RETURN position
END FUNCTION
```

The array of unique product codes is shown below.

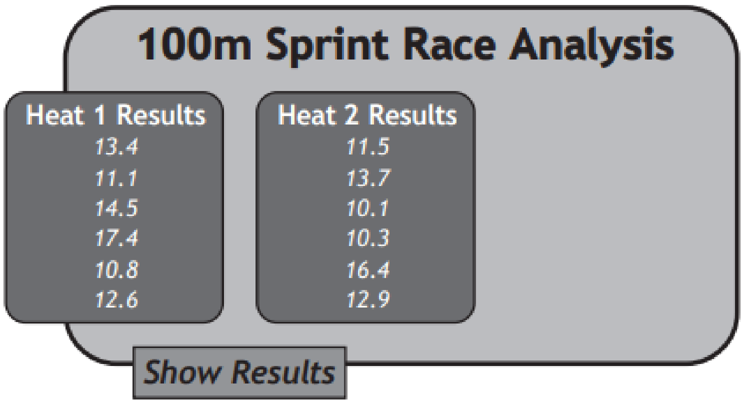
C232	T546	X756	W482	B629	....

The product code ‘F333’ is entered. It is not in the array.

State the type of error. Explain your answer. (2)

Question 2 (2018 Qu15)

SportsStats is a program that processes the results of athletics competitions. The results of two different heats are compared to find which heat had the fastest time.



When a user presses the 'Show Results' button, the program should output the number of the heat that had the fastest runner, for example:

```
"The fastest runner ran in heat 2"
```

The program makes use of the following function:

```
Line 1    FUNCTION fastest_time(ARRAY OF REAL list) RETURNS REAL
Line 2       DECLARE min INITIALLY list[0]
Line 3       DECLARE upper INITIALLY length(list[])
Line 4       FOR index FROM 1 to (upper-1) DO
Line 5         IF min < list[index] THEN
Line 6           SET min TO list[index]
Line 7         END IF
Line 8       END FOR
Line 9       RETURN min
Line 10   END FUNCTION
```

The function is used in the following section of code:

```
...
Line 21   SET heat1 TO [13.4, 11.1, 14.5, 17.4, 10.8, 12.6]
Line 22   SET heat2 TO [11.5, 13.7, 10.1, 10.3, 16.4, 12.9]
Line 23   SET first_result TO fastest_time (heat1)
Line 24   SET second_result TO fastest_time (heat2)
Line 25   IF first_result < second_result THEN
Line 26      SEND "The fastest runner ran in heat 1" TO DISPLAY
Line 27   ELSE
Line 28      SEND "The fastest runner ran in heat 2" TO DISPLAY
Line 29   END IF
```

A) Testing reveals an error in the function. The function is first called during execution of line 23 of the main program. In order to identify this error, a watchpoint has been set to show the value of the min variable each time it is changed. Complete the table to show the values that would be shown when this watchpoint is triggered. (3)

Function Line	min
2	
6	
6	

B) Testers report that the program sometimes outputs the incorrect result.

(i) Identify the error in the function that causes incorrect output. (1)

(ii) State the type of error that has caused this issue. (1)

(iii) Explain why the incorrect code outputs the correct statement. Your answer should make reference to the original heat results shown on lines 21 and 22 of the code. (2)

EVALUATION STAGE

During the Evaluation Stage, the overall success of the entire project is considered. This is an objective review of the software to establish whether it meets the required criteria.

An evaluation report would discuss the following aspects of the solution.

FITNESS FOR PURPOSE

This reflects whether the software carries out all the tasks required of the software specification.

Your evaluation should identify any discrepancies between the software specification and the completed software.

EFFICIENT USE OF CODING CONSTRUCTS

This reflects whether the software writers have used their knowledge of constructs to help them create efficient code. For example using:

- suitable data types or structures
- conditional or fixed loops
- arrays
- nested selection
- procedures or functions with parameter passing

Your evaluation should identify where your coding has been efficient.

USABILITY

This reflects how intuitive the software is from a user's perspective and should include:

- the general user interface
- the user prompts
- the screen layout
- any help screens

Your evaluation should identify features of the software that have enhanced usability for the user.

MAINTAINABILITY

This reflects how easy it is to alter the software. The factors affecting maintainability include:

- readability of the code — made easier by using meaningful variable names, comments, indentation and whitespace
- amount of modularity — using functions and procedures effectively

Your evaluation should identify how your code helps with the maintainability of the software.

ROBUSTNESS

This reflects how well the software copes with errors during execution including:

- exceptional data, e.g. the computer crashing if “out of range”
- incorrect data entered

Your evaluation should reflect the testing that has been undertaken to meet the specification, as well as to demonstrate some degree of robustness.

PRACTISE QUESTIONS

Look back at your code for the Olympics task.

With reference to your own program code, evaluate:

a) the fitness for purpose of your program (1)

b) the maintainability of your program with reference to readability and modularity (2)