OCR
Oxford Cambridge and RSA

…day June 20XX – Morning/Afternoon

GCSE (9–1) Computer Science

J277/02 Computational thinking, algorithms and programming

SAMPLE MARK SCHEME

Time allowed: 1 hour 30 minutes

MAXIMUM MARK 80

SAMPLE MARK SCHEME

Version 1.6

This document consists of 23 pages

MARKING INSTRUCTIONS

PREPARATION FOR MARKING

SCORIS

1. Make sure that you have accessed and completed the relevant training packages for on–screen marking: *scoris assessor Online Training*; *OCR Essential Guide to Marking*.

2. Make sure that you have read and understood the mark scheme and the question paper for this unit. These are posted on the RM Cambridge Assessment Support Portal http://www.rm.com/support/ca

3. Log–in to scoris and mark the **required number** of practice responses ("scripts") and the **required number** of standardisation responses.

   YOU MUST MARK 10 PRACTICE AND 10 STANDARDISATION RESPONSES BEFORE YOU CAN BE APPROVED TO MARK LIVE SCRIPTS.

	Assessment Objective
AO1	Demonstrate knowledge and understanding of the key concepts and principles of computer science.
AO1 1a	Demonstrate knowledge of the key concepts and principles of computer science.
AO1 1b	Demonstrate understanding of the key concepts and principles of computer science.
AO2	Apply knowledge and understanding of key concepts and principles of computer science.
AO2 1a	Apply knowledge of key concepts and principles of computer science.
AO2 1b	Apply understanding of key concepts and principles of computer science.
AO3	Analyse problems in computational terms: • to make reasoned judgements • to design, program, evaluate and refine solutions.
AO3 1	To make reasoned judgements (this strand is a single element).
AO3 2a	Design solutions.
AO3 2b	Program solutions.
AO3 2c	Evaluate and refine solutions.

COMPONENT 2 SECTION B SYNTAX GUIDANCE

In Section B, certain questions require candidates to answer in either the OCR Exam Reference Language or the high-level programming language they are familiar with. The information in this section provides generic guidelines in relation to the marking of these questions.

Where a response requires an answer in OCR Exam Reference Language or a high-level programming language, a candidate's level of precision will be assessed. These questions are designed to test both a candidate's programming logic and understanding of core programming structures.

Marks will be given for correctly using syntax to represent core programming constructs which are common across all programming languages. The construct must be present in a recognisable format in a candidate's answer.

Where the response requires a candidate to respond using the OCR Exam Reference Language or a high-level programming language, answers written in pseudocode, natural English or bullet points **must not** be awarded marks.

The guidance below covers the elements of each core construct. As guidance, several examples are provided for each. These examples are not exclusive but do present a variety of acceptable ways taken from a range of different languages.

Concept		Examiner Guidance
Commenting		

`//`	`//This function squares a number`
	`function squared(number)`
	`    squared = number^2`
	`    return squared`
	`endfunction`
	`//End of function`
		• Other examples allowable, e.g.:
		  ○ # this is a comment
		  ○ /* this is another comment */

Variables		

`=`
`const`
`global`
	`x = 3`
	`name = "Louise"`
	`const vat = 0.2`
	`global userID = "Cust001"`
		• Variables and constants are assigned using the = operator
		• Constants are assigned using the `const` keyword (or similar)
		• Identifiers should **not** have clear spaces within them or start with numbers
		• String values must use quotation marks (or equivalent)
		• Assignment must use =, :=, ← (or a suitable alternative)
		• variable identifier must be on the left when using OCR Exam Reference Language and the value to be assigned on the right
		• Some languages allow the value on the left- and the identifier on the right-hand side
		• Variables and constants are declared the first time a value is assigned. They assume the data type of the value they are given
		• Variables and constants that are declared inside a function or procedure are local to that subroutine
		• Variables in the main program can be made global with the keyword global
		• For input, a suitable command word for input and a variable identifier to assign data to (if required)
		e.g.
		`INPUT identifier`
		`identifier = INPUT`

Input/Output		

`input(…)`
 	
`print(…)`
	`myName = input("Please enter a name")`
	
	`print("My name is Noni")`
	`print(myArray[2,3])`
		• For output, a command word for output (e.g. output, print, cout)
		• Data to be output. If this is a string then quotation marks (or equivalent) are required
		• If multiple items are to output, a suitable symbol for concatenation such as +, &.

Casting		

`str()`
 	
`int()`
 	
`float()`
 	
`real()`
 	
`bool()`
	`str(345)`
	
	`int("3")`
	
	`float("4.52")`
	
	`real("4.52")`
	
	`bool("True")`
		• Variables can be typecast using the int str and float functions

Iteration		

`for … to …`
 	
`next …`
 	
`for … to … step …`
 	
`next …`
	`for i=0 to 9`
	`  print("Loop")`
	`next i`
	
	`for i=2 to 10 step 2`
	`  print(i)`
	`next i`
	
	`for i=10 to 0 step -1`
		• `for` keyword
		• …with counter variable
		• Identification of number of times to iterate
		• Clear identification of which section of code will be repeated (e.g. using indentation, next keyword or equivalent, {braces})

	`    print(i)` `next i`	

`while …`
 	
`endwhile`
 	
`do`
 	
`until …`
	`while answer != "Correct"`
	`    answer = input("New answer")`
	`endwhile`
	
	`do`
	`    answer = input("New answer")`
	`until answer == "Correct"`
		• `While / do..until` key words or equivalent
		• …with logical comparison
		• clear identification of which section of code will be repeated (e.g. using indentation, endwhile/until keyword or equivalent, braces)

Selection		

`if … then`
`elseif … then`
`else`
`endif`
	`if answer == "Yes" then`
	`    print("Correct")`
	`elseif answer == "No" then`
	`    print("Wrong")`
	`else`
	`    print("Error")`
	`endif`
		• `if` key word followed by logical comparison
		• key word for `elseif` or equivalent followed by logical comparison
		• key word for `else` or equivalent with no comparison
		• clear identification of which section of code will be executed depending upon decision

`switch … :`
`    case … :`
`    case … :`
`    default:`
`endswitch`
	`switch day :`
	`    case "Sat":`
	`        print("Saturday")`
	`    case "Sun":`
	`        print("Sunday")`
	`    default:`
	`        print("Weekday")`
	`endswitch`
		• May be referred to differently in some languages. The format to the left will be used in all questions
		• `switch/select` key word or equivalent followed by variable/ value being checked
		• key word for each case followed by variable/ value to compare to
		• key word for default case (last option)
		• clear identification of which section of code will be executed depending upon decision

String handling/operations		
`.length`	`subject = "ComputerScience"` `subject.length` **gives** `the value 15`	• Suitable key word to indicate length and string identifier e.g. **`len(string)`**

`.substring(x , i)`
`.left(i)`
`.right(i)`
	`subject.substring(3,5)` **returns** `"puter"`
	`subject.left(4)` **returns** `"Comp"`
	`subject.right(3)` **returns** `"nce"`
		• Suitable string and characters required identified
		• Use of key words such as **`left, right, mid`**, etc, are all acceptable as long as these are precise
		• Treating a string as an array of characters is acceptable

`+` (concatenation)	`print(stringA + stringB)` `print("Hello, your name is: " + name)`	• Alternate symbol used indicate two strings or values are being concatenated is acceptable e.g. `stringA & stringB` or `stringA.stringB` • Use of comma e.g. `print(stringA, stringB)` is acceptable to output multiple values but examiners should be aware that this is not concatenation.
`.upper` `.lower`	`subject.upper` **gives** `"COMPUTERSCIENCE"` `subject.lower` **gives** `"computerscience"`	• Suitable key word to indicate string to be converted and whether this is to be converted to upper or lower case e.g. **`lower(stringname)`**
`ASC(…)` `CHR(…)`	`ASC ('A')` **returns** `65` (numerical) `CHR(97)` **returns** `'a'` (char)	• Suitable keyword to indicate conversion and whether this is to or from ASCII. Where converting from ASCII, an integer value must be given and where converting to ASCII, a single character must be given.

File handling		

`open(…)`
`.close()`
 	
`.readLine()`
 	
`.writeLine(…)`
 	
`.endOfFile()`
 	
`newFile()`
	`myFile = open("sample.txt")`
	`myFile.close()`
	
	`myFile.readLine()` **returns** the next line in the file
	
	`myFile.writeLine("Add new line")`
	
	`while NOT myFile.endOfFile()`
	`  print(myFile.readLine())`
	`endwhile`
	
	`newFile("myText.txt")`
		• `open` keyword (or equivalent)
		• `read` or `write` clearly identified
		• `write` or read keyword (or equivalent)
		• `close` file keyword (or equivalent)
		• `newFile` keyword (or equivalent)

Arrays		

`array colours[…]`
 	
`array gameboard[…,…] = …`
 	
`names[…] = …`
`gameboard[…,…] = …`
	`array colours[5]`
	
	`array colours = ["Blue", "Pink", "Green", "Yellow", "Red"]`
	
	`array gameboard[8,8]`
	
	`names[3] = "Noni"`
	`gameboard[1,0] = "Pawn"`
		• Array identifier
		• Index number to be accessed in square brackets, rounded brackets or curly braces (all acceptable)
		
		• Array identifier assigned to initial values in one step
		
		• For 2D arrays, the two indices should be given in one bracket separated by a comma **or** in two separate brackets, e.g.
		`gameboard[4,6]`
		`gameboard[4][6]`
		
		Where 2D arrays are represented by tables in a question, candidates are expected to use the **same** row/column or column/row format as given in the question. **This will always be given.**

Sub programs		

`procedure` *`name`* `(…)`
 	
`endprocedure`
	`procedure agePass()`
	`    print("You are old enough to ride")`
	`endprocedure`
	
	`procedure printName(name)`
	`    print(name)`
	`endprocedure`
	
	`procedure multiply(num1, num2)`
	`    print(num1 * num2)`
	`endprocedure`
		• `function` or `procedure` key word (or equivalent)
		• … followed by identifier
		• Any parameters passed in are contained within brackets and come after identifier name
		• Clear identification of which section of code is contained within the subroutine (e.g. indentation, **`endsub`** key word, braces)

`procedure(`*`parameters`*`)`	`agePass()`
	
	`printName(parameter)`
	
	`multiply(parameter1, parameter2)`	

`function` *`name`* `(…)`
`    …`
`    return …`
`endfunction`
	`function squared(number)`
	`    squared = number^2`
	`    return squared`
	`endfunction`
		• **functions only**: a suitable method of returning a value (e.g. `return` keyword or assignment of value to function identifier)
		
		e.g.
		`def newfunction(x,y)`
		`    total = x + y`
		`    newfunction = total`

`function(`*`parameters`*`)`	`print(squared(4))`
	
	`newValue = squared(4)`	

Random numbers		

`random(…,…)`	`myVariable = random(1,6)`
	
	`myVariable = random(-1.0,10.0)`
		• **`random`** key word (or equivalent)
		• identification of either smallest and largest number to be chosen **or** just largest number
		
		e.g.
		`randnumber(10)`
		`rand(1,6)`

Comparison operators			
`==`	Equal to	`<=`	Less than or equal to
`!=`	Not equal to	`>`	Greater than
`<`	Less than	`>=`	Greater than or equal to

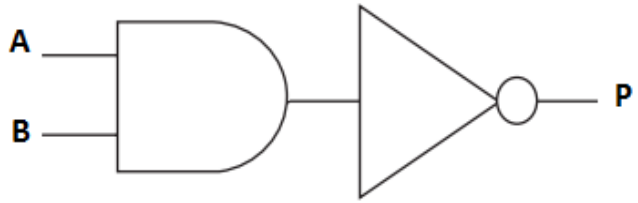
Boolean operators	
`AND`	Logical AND
`OR`	Logical OR
`NOT`	Logical NOT

Arithmetic operators	
`+`	Addition
`-`	Subtraction
`*`	Multiplication
`^`	Exponent
`/`	Division
`MOD`	Modulo
`DIV`	Quotient

- = or == are both acceptable for equal to.
- <> is acceptable for not equal to.
- Care must be taken by candidates to ensure that > and < are not mixed up.
- Candidates must understand that < and > are non-inclusive, so that <9 does not include 9. This is different than <=9 which is inclusive and therefore does include 9.
- Alternative symbols for arithmetic operators are acceptable where these appear in other high-level languages (such as % for MOD or ** for exponentiation).

- 6 x 5 is not an acceptable alternative for multiplication.
- Alternative logical operators are acceptable where these appear in other high-level languages (such as **`&&`** for **`AND`**).
- Alternative Arithmetic Operators may be used as well (such as % for modulus).
- Candidates must be aware that logical operators must be used correctly:

`if x > 0 AND x < 10` is logically correct.
`if x > 0 AND < 10` is **not** logically correct.

SECTION A					
Question			**Answer**	**Marks**	**Guidance**
1	**a**		<table><tr><th>A</th><th>B</th><th>P</th></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td>1</td></tr><tr><td></td><td></td><td>1</td></tr><tr><td></td><td></td><td></td></tr></table>	**2 (AO1 1b)**	1 mark for each correct answer in table 'True' or 'T' are also credit worthy.
	b			**1 (AO1 1b)**	Correct Answer Only
2	**a**		• `input("enter first number")` • `if` • **`num2`** • `print (num1)` • `print (num2)`	**5 (AO3 2b)**	Allow equivalent pseudocode expressions Variables must not have speech marks around them
	b		• use of condition controlled loop (while or do/until)… • …checking condition of number larger than or equal to 0 • Input number from user within loop (FT if no loop) • multiply number input by 2… • ….output value in number	**5 (AO3 2b)**	e.g. 1 store 10 in number while number is greater than or equal to 0 do the following: Take input from the user, store in number Multiply number by 2 Output number e.g. 2 while number >= 0 number = input() output(number * 2) Ignore non-initialisation of value used in condition for loop.

SECTION A					
Question			**Answer**	**Marks**	**Guidance**
3			• SELECT **StudentName, Subject, Grade** • FROM **Results** • WHERE **Subject = "Art"**	**1 (AO1 1b)** **2 (AO3 2a)**	Correct Answer Only Accept SELECT *****
4	**a**		• RebEl	**1 (AO2 1b)**	Correct Answer Only (allow any case)
	b	**i**	• uitFr	**1 (AO2 1b)**	Correct Answer Only (allow any case)
		ii	• Taking firstname, surname and teacher or student as input • Checking IF role is teacher/student (using appropriate selection) • For teacher ...Generating last 3 letters of surname using appropriate string manipulation • ...Generating first 2 of letters of firstname and adding to previous • For student.... correctly calculating as before • Correct concatenation **and** output e.g. Ask the user to input the data, store in variables firstname, surname and role. Check whether the role entered is teacher. If it is, join the right 3 most letters in surname with the left 2 letters in firstname. Store this in username. If it is not teacher, join the left 3 letters from firstname with the left 2 letters from surname. Store this in username. Output the value in username.	**6 (AO3 2b)**	1 mark for each correct bullet to a maximum of 6. If used, a flowchart should represent the bulleted steps in the answer column.

<table>
<tr><td colspan="3">SECTION A</td><td></td><td></td></tr>
<tr><td colspan="3">Question</td><td>Answer</td><td>Marks</td><td>Guidance</td></tr>
<tr><td>5</td><td>a</td><td></td><td>• To convert it to binary/machine code
• The processor can only understand machine code</td><td>1
(AO1 1a)</td><td>Maximum 1 mark</td></tr>
<tr><td></td><td>b</td><td></td><td>• Compiler translates all the code in one go…
• …whereas an interpreter translates one line at a time
• Compiler creates an executable…
• …whereas an interpreter does not/executes one line at a time
• Compiler reports errors at the end…
• …whereas an interpreter stops when it finds an error</td><td>4
(AO1 1b)</td><td>1 mark to be awarded for the correct identification and one for a valid description up to a maximum of 4 marks. No more than 2 marks for answers relating only to interpreters and no more than 2 marks for answers only relating to compilers.</td></tr>
<tr><td>6</td><td>a</td><td></td><td><table>
<tr><td>crime</td><td>bait</td><td>fright</td><td>victory</td><td>nibble</td><td>loose</td></tr>
<tr><td>bait</td><td>crime</td><td>fright</td><td>victory</td><td>nibble</td><td>loose</td></tr>
<tr><td>bait</td><td>crime</td><td>fright</td><td>nibble</td><td>victory</td><td>loose</td></tr>
<tr><td>bait</td><td>crime</td><td>fright</td><td>nibble</td><td>loose</td><td>victory</td></tr>
<tr><td>bait</td><td>crime</td><td>fright</td><td>loose</td><td>nibble</td><td>victory</td></tr>
</table></td><td>4
(AO2 1b)</td><td>1 mark for each row from rows 2–5. Allow multiple swaps in one stage, where it is clear that a bubble sort has been applied.</td></tr>
</table>

SECTION A					
Question			**Answer**	**Marks**	**Guidance**
6	**b**		• Comparing zebra to orange • Greater, so split and take right side • Further comparison (1 or 2 depending on choices made) • Correct identification of zebra using methodology above e.g. compare zebra to orange greater, split right compare to wind greater, split right compare to zebra	**4 (AO2 1b)**	1 mark per bullet (multiple ways through, marks awarded for appropriate comparison and creation of sub groups).
7	**a**		1 mark for naming the example and 1 mark for an example related to that method E.g • Comments/annotation… • …E.g. any relevant example, such as line 4 checks the input is valid • Indentation… • …E.g. indenting within IF statement • Using constants… • …E.g. π	**4 (AO2 1b)**	

SECTION A					
Question			**Answer**	**Marks**	**Guidance**
7	**b**		• radius • area	**2** **(AO1 1b)**	1 mark per bullet up to a maximum of 2 marks.
	c	**i**	• 3.142 • 2 • 1 • 30	**1** **(AO2 1a)**	1 mark for one correct identification.
	c	**ii**	• The number does not need to be changed while the program is running • The number can be updated once and it updates throughout	**1** **(AO1 1a)**	Maximum of 1 mark.
	d		• HAS been used • HAS been used • HAS NOT been used	**3** **AO2 1b**	
	e		• Error diagnostics (any example) • Run-time environment • Editor (any feature such as auto-correct, auto-indent) • Translator • Version control • Break point • Stepping	**2** **(AO1 1a)**	1 mark per bullet to a maximum of 2 marks. Only 1 example per bullet, e.g. auto-correct and auto-indent would only gain 1 mark.

SECTION B					
Question			**Answer**	**Marks**	**Guidance**
8	**a**		Integer (1)… • …number of seconds not important (1) • … level of accuracy not needed so round to nearest minute (1) • …using a decimal to store seconds (0-60) is not appropriate (1) Real (1)… • … number of seconds may be important (1) • … allows parts/fractions to be stored over integers (1)	**1 (AO3 2a)** **1 (AO3 1)**	One mark for appropriate data type identified. One mark for appropriate justification linked to the data type chosen.
8	**b**	**i**	• `or` • `>300 // >= 301` • `print`	**3 (AO3 2b)**	**<u>High-level programming language / OCR Exam Reference Language response required</u>** Do not accept pseudocode / natural English. MP2 do not accept 'greater than', must use the HLL syntax > or >= MP3 must be a suitable output command word that could be found in a HLL e.g. `print` (Python), `console.writeline` (VB), `cout` (C++)
	b	**ii**	• Suitable invalid test data (i.e. > 300, e.g. 350) • "Value accepted" or equivalent	**2 (AO3 2c)**	

SECTION B					
Question			**Answer**	**Marks**	**Guidance**
8	**c**		`print (minsPlayed[0,4])`	**1 (AO3 2b)**	**High-level programming language / OCR Exam Reference Language response required** Do not accept pseudocode / natural English. `print` may be a suitable output command word that could be found in a HLL e.g. `print` (Python), `console.writeline` (VB), `cout` (C++) The array elements may be accessed together `[0,4]` (VB.NET) or separately `[0][4]` (Python)

8	**d**		• Initialises total as 0 **and** prints out total the end (as per original program)
			• Uses iteration, e.g. FOR, WHILE
			• …that repeats 5 times
			• …correctly adds up values using loop index
			
			e.g.
			`total = 0`
			`for x = 0 to 4`
			`  total = total + hoursplayed[2, x]`
			`next x`
			`console.writeline(total)`
			
			e.g.
			`total = 0`
			`for x in range (0, 4)`
			`  total += hoursplayed[2][x]`
			`next x`
			`print (total)`
				4 (AO3 2c)	**High-level programming language / OCR Exam Reference Language response required**
					
					Do not accept pseudocode / natural English.
					
					MP1 must have appropriate identifier, = and then the numeric 0
					MP2 must have `for` or `while`
					MP3 must have the for stopping condition 4/5
					MP4 must have the same identifier for MP1 and equal and + to add the data in the array (using either `[x,y]` or `[x][y]`. This could be `total = total +` …. Or `total +=` ….

<table>
<tr><td colspan="3">SECTION B</td><td></td><td></td></tr>
<tr><td colspan="3">Question</td><td>Answer</td><td>Marks</td><td>Guidance</td></tr>
<tr><td></td><td></td><td></td><td>e.g.
<pre>total = 0;
for (int x = 0; x <= 4; x++){
    total = total + hoursplayed[2][x];
}
System.out.println (total);</pre></td><td></td><td></td></tr>
<tr><td>8</td><td>e</td><td></td><td>
<table>
<tr><th></th><th>x</th><th>y</th><th>output</th></tr>
<tr><td>MP1</td><td>15</td><td>0</td><td></td></tr>
<tr><td rowspan="2">MP2</td><td>14</td><td>1</td><td></td></tr>
<tr><td>12</td><td>2</td><td></td></tr>
<tr><td rowspan="3">MP3</td><td>9</td><td>3</td><td></td></tr>
<tr><td>5</td><td>4</td><td></td></tr>
<tr><td>0</td><td>5</td><td></td></tr>
<tr><td>MP4</td><td></td><td></td><td>5</td></tr>
</table>
</td><td>4
(AO3 2c)</td><td>one mark for first row

one mark for row 2 and 3

one mark for rows 4, 5, and 6

one mark for the correct output (the only value in the output column, in any position)</td></tr>
<tr><td>8</td><td>f</td><td></td><td>1 mark per bullet
• Test data either 0 or less characters, or 20 or more characters
• Stating correct output

• Test data between 1 and 19 characters (inc)
• Stating correct output</td><td>4
(AO3 2c)</td><td>Mark test data first, both must meet different criteria. Then mark output for each.</td></tr>
</table>

SECTION B					
Question			**Answer**	**Marks**	**Guidance**
8	**g**	**i**	**Input** • Number of hours and minutes **Output** • Number of minutes	**2** **(AO3 2a)**	

	g	**ii**	• Program calls function correctly using hours and minutes variables
			• Parameters used appropriately
			• Calculation is computed accurately
			• Final total is returned suitably
				4
				(AO3 2a)
					`hours = input("Please enter number of hours played")`
					
					`minutes = input("Please enter number of minutes played")`
					
					`finalTotal = totalMins(hours, minutes)`
					
					`print (finalTotal)`
					
					`function totalMins(hours,minutes)`
					
					`    total = (hours * 60) + mins`
					
					`    return total`
					
					`endfunction`
					
					1. Parameters named in function must be used within the function itself
					2. Does not matter if function uses different names to those declared in main program
					3. Return must be included with the correct local variable for total

SECTION B					
Question			**Answer**	**Marks**	**Guidance**

8	**g**	**iii**	• Takes input from the user
			• Compares if input is larger than 120…
			• …if true, outputs "`You played games for too long!`"
			• …if false, outputs "`You are under your time limit!`"
				4 (AO3 2b)	**<u>High-level programming language / OCR Exam Reference Language response required</u>**
					
					Do not accept pseudocode / natural English.
					
					Example algorithm given below
					
					`minutes = input("Enter minutes played")`
					`if minutes > 120`
					`  print "You played games for too long!"`
					`else`
					`  print "You are under your time limit!"`
					`endif`
					
					Accept alternative (but suitable) output messages.
					
					Accept logical comparison of input less than or equal to 120 and appropriate True/False statements.