CAMBRIDGE
International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/23**

Paper 2 Fundamental Problem-solving and Programming Skills **October/November 2024**

MARK SCHEME

Maximum Mark: 75

Published

This mark scheme is published as an aid to teachers and candidates, to indicate the requirements of the examination. It shows the basis on which Examiners were instructed to award marks. It does not indicate the details of the discussions that took place at an Examiners' meeting before marking began, which would have considered the acceptability of alternative answers.

Mark schemes should be read in conjunction with the question paper and the Principal Examiner Report for Teachers.

Cambridge International will not enter into discussions about these mark schemes.

Cambridge International is publishing the mark schemes for the October/November 2024 series for most Cambridge IGCSE, Cambridge International A and AS Level components, and some Cambridge O Level components.

Generic Marking Principles

These general marking principles must be applied by all examiners when marking candidate answers. They should be applied alongside the specific content of the mark scheme or generic level descriptions for a question. Each question paper and mark scheme will also comply with these marking principles.

GENERIC MARKING PRINCIPLE 1:

Marks must be awarded in line with:

- the specific content of the mark scheme or the generic level descriptors for the question
- the specific skills defined in the mark scheme or in the generic level descriptors for the question
- the standard of response required by a candidate as exemplified by the standardisation scripts.

GENERIC MARKING PRINCIPLE 2:

Marks awarded are always **whole marks** (not half marks, or other fractions).

GENERIC MARKING PRINCIPLE 3:

Marks must be awarded **positively**:

- marks are awarded for correct/valid answers, as defined in the mark scheme. However, credit is given for valid answers which go beyond the scope of the syllabus and mark scheme, referring to your Team Leader as appropriate
- marks are awarded when candidates clearly demonstrate what they know and can do
- marks are not deducted for errors
- marks are not deducted for omissions
- answers should only be judged on the quality of spelling, punctuation and grammar when these features are specifically assessed by the question as indicated by the mark scheme. The meaning, however, should be unambiguous.

GENERIC MARKING PRINCIPLE 4:

Rules must be applied consistently, e.g. in situations where candidates have not followed instructions or in the application of generic level descriptors.

GENERIC MARKING PRINCIPLE 5:

Marks should be awarded using the full range of marks defined in the mark scheme for the question (however; the use of the full mark range may be limited according to the quality of the candidate responses seen).

GENERIC MARKING PRINCIPLE 6:

Marks awarded are based solely on the requirements as defined in the mark scheme. Marks should not be awarded with grade thresholds or grade descriptors in mind.

Mark scheme abbreviations

/	separates alternative words / phrases within a marking point
//	separates alternative answers within a marking point
underline	actual word given must be used by candidate (grammatical variants accepted)
max	indicates the maximum number of marks that can be awarded
( )	the word / phrase in brackets is not required, but sets the context

Note: No marks are awarded for using brand names of software packages or hardware.

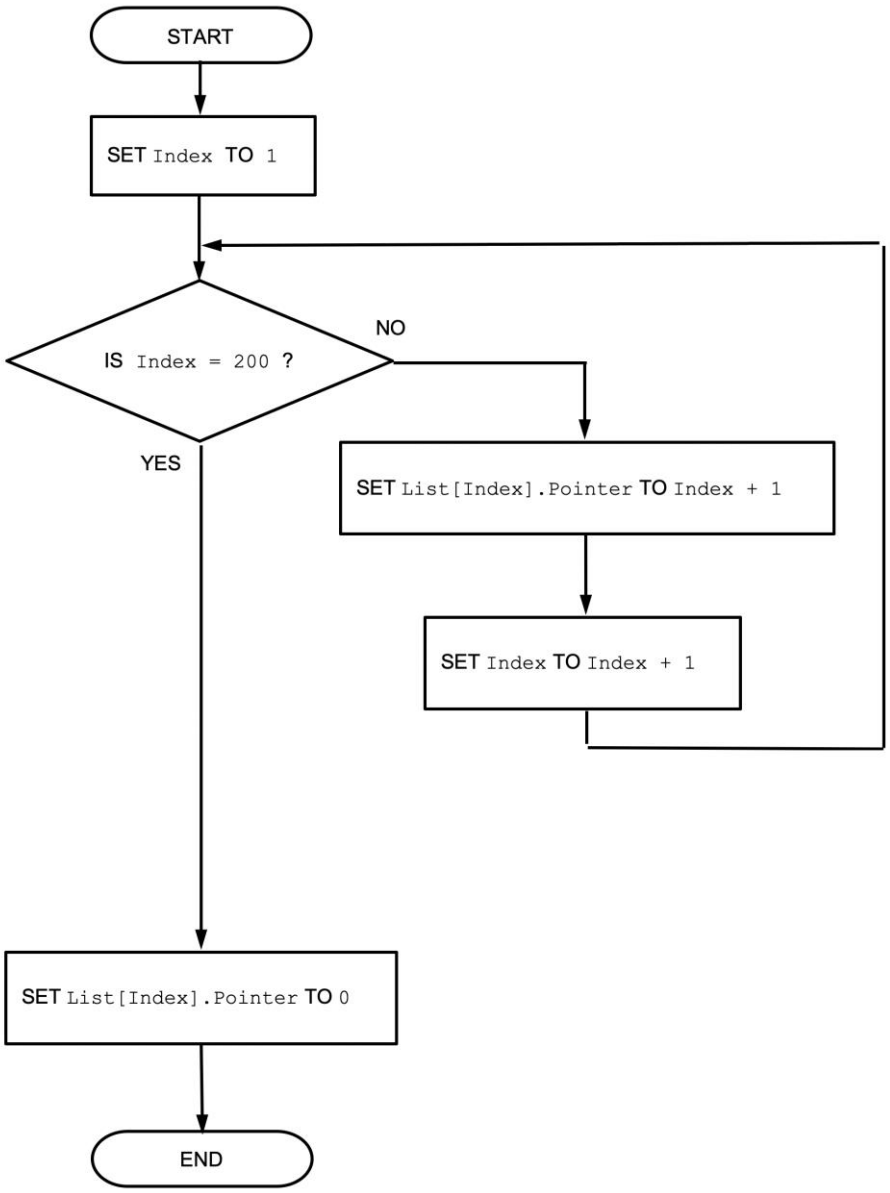
<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>1(a)</td><td>
<table>
<tr><th>Pseudocode example</th><th>Selection</th><th>Iteration</th><th>Subroutine</th></tr>
<tr><td><pre>FOR Index ← 1 TO 3
   IF Safe[Index] = TRUE THEN
      Flap[Index] ← 0
   ENDIF
NEXT Index</pre></td><td>✓</td><td>✓</td><td></td></tr>
<tr><td><pre>CASE OF Compound(3)</pre></td><td>✓</td><td></td><td>✓</td></tr>
<tr><td><pre>REPEAT UNTIL AllDone() = TRUE</pre></td><td></td><td>✓</td><td>✓</td></tr>
<tr><td><pre>WHILE Result[3] <> FALSE</pre></td><td></td><td>✓</td><td></td></tr>
</table>
One mark per row</td><td>4</td></tr>
<tr><td>1(b)</td><td>
<table>
<tr><th>Variable</th><th>Example data value</th><th>Data type</th></tr>
<tr><td><pre>Available</pre></td><td><pre>TRUE</pre></td><td>BOOLEAN</td></tr>
<tr><td><pre>Received</pre></td><td><pre>"18/04/2021"</pre></td><td>STRING</td></tr>
<tr><td><pre>Index</pre></td><td><pre>100</pre></td><td>INTEGER</td></tr>
</table>
One mark per row</td><td>3</td></tr>
<tr><td>1(c)</td><td>
<table>
<tr><th>Expression</th><th>Evaluates to</th></tr>
<tr><td><pre>Available AND NOT(Index > 100)</pre></td><td>TRUE</td></tr>
<tr><td><pre>Index MOD 30</pre></td><td>10</td></tr>
<tr><td><pre>NUM_TO_STR(Index + "33")</pre></td><td>ERROR</td></tr>
</table>
One mark per row</td><td>3</td></tr>
</table>

Question	Answer	Marks

2(a)	`DECLARE Count, Total, NextNumber : INTEGER`
	
	`Total ← 0`
	`FOR Count ← 1 TO 100`
	`  OUTPUT "Input an integer value"`
	`  INPUT NextNumber`
	`  IF NextNumber > 0 THEN`
	`  Total ← Total + NextNumber`
	`  ENDIF`
	`NEXT Count`
	
	`OUTPUT Total`
	
	Mark as follows:
	MP1 Declarations of all variables used
	MP2 Loop for 100 iterations
	MP3 Prompt **and** input a value **in a loop and**
	MP4 Test for value > 0 // >=1 **in a loop**
	MP5 Sum the Total **in a loop**
	MP6 Output of Total after the **loop**
	
	Max 5 marks
		5

2(b)	**MP1** Construct: Iteration / Repetition **MP2** Use: To loop through all 100 inputs // To loop 100 times ALTERNATIVE: **MP1** Construct: Selection **MP2** Use: To test whether the value input is positive	**2**

Question	Answer	Marks
3(a)(i)	0 is not a valid array / `List` index value // No 0^{th} element in the array	**1**
3(a)(ii)	`0 to 200`	**1**

Question	Answer	Marks
3(b)	 Mark as follows: **MP1** Use of variable as array index **and** initialisation to 1 / to first element of array **MP2** Loop for 199 / 200 iterations **MP3** Assign Index+1 to each pointer field in a loop **MP4** Assign 0 to 200th pointer field	**4**

Question	Answer	Marks
3(c)	One mark per step: **MP1** Assign `HeadPointer` to `ThisPointer` / Current Pointer // Identify first node using headpointer **MP2** Loop the following until `ThisPointer` value is 0 (zero) / null pointer reached **MP3** … Output data (field) from array element at index `ThisPointer` // Output data (field) of the current node / array element **MP4** … Assign pointer field from current array element / index / to `ThisPointer` / Current Pointer // Assign pointer field from current node to `ThisPointer` / Current Pointer	**4**

4(a)	One mark per highlighted part:
	
	`FOR Index ← 1 TO 5`
	`  Lower ← GB[Index] - 2`
	`  Upper ← GB[Index] + 2  // Lower + 4`
	`  IF Mark >= Lower AND Mark <= Upper THEN`
	`//IF Mark <= Upper AND Mark >= Lower THEN`
	`  OUTPUT "Check this paper"`
	`  ENDIF`
	`NEXT Index`
		4

4(b)(i)	**MP1** Use `Mark` as the index to the `Check` array // to specify an array element **MP2** If value of indexed element is `TRUE`, then the paper will need to be checked	**2**

Question	Answer	Marks

4(b)(ii)	Example:solution
	
	`PROCEDURE GBInitialise()`
	`  DECLARE GBIndex, GBMark, ThisIndex : INTEGER`
	
	`  FOR ThisIndex ← 0 TO 75`
	`  Check[ThisIndex] ← FALSE // initialise all values`
	`  NEXT ThisIndex`
	
	`  FOR GBIndex ← 1 TO 5`
	`  GBMark ← GB[GBIndex]`
	`  FOR ThisIndex ← GBMark - 2 TO GBMark + 2`
	`  Check[ThisIndex] ← TRUE //Set GBMark ± 2 to TRUE`
	`  NEXT ThisIndex`
	`  NEXT GBIndex`
	`ENDPROCEDURE`
	
	Mark as follows:
	MP1 Procedure heading and ending
	MP2 Initialise `Check` array to `FALSE`
	MP3 Loop through `GB` array
	MP4 Extract the `GB` mark
	MP5 Loop for 5 elements across `GBMark` ± 2 // Check if within `GBMark` ± 2
	MP6 Assign each element of `Check` array to `TRUE`
	
	ALTERNATIVE – Example using selection Version 1
	
	`FOR ThisIndex ← 0 TO 75`
	`  CASE ThisIndex`
	`  GB[1] – 2 TO GB[1] + 2 : Check[ThisIndex] ← TRUE`
	`  GB[2] – 2 TO GB[2] + 2 : Check[ThisIndex] ← TRUE`
	`  GB[3] – 2 TO GB[3] + 2 : Check[ThisIndex] ← TRUE`
	`  GB[4] – 2 TO GB[4] + 2 : Check[ThisIndex] ← TRUE`
	`  GB[5] – 2 TO GB[5] + 2 : Check[ThisIndex] ← TRUE`
	`  OTHERWISE: Check[ThisIndex] ← FALSE`
	`  ENDCASE`
	`NEXT ThisIndex`
		6

Question	Answer	Marks

4(b)(ii)	ALTERNATIVE – Example using selection Version 2
	
	`DECLARE ThisIndex, GBIndex : INTEGER`
	`DECLARE Lower, Higher : INTEGER`
	
	`FOR ThisIndex ← 0 TO 75`
	`  For GBIndex ← 1 T0 5`
	`  Lower ← GB[GBIndex] - 2`
	`  Higher ← GB[GBIndex] + 2`
	`  IF ThisIndex >= Lower AND ThisIndex <= Higher THEN`
	`  Check[ThisIndex] ← TRUE`
	`  ELSE`
	`  Check[ThisIndex] ← FALSE`
	`  ENDIF`
	`  NEXT Index`
	`NEXT ThisIndex`
	
	Mark as follows:
	MP1 Procedure heading and ending
	MP2 Loop through `Check` array// loop from 0 to 75
	MP3 Attempt at using CASE / Selection structure with five clauses/ **five** checks for range
	MP4 Extract GB grade
	MP5 Compare `ThisIndex` with each element in GB array ±2
	MP6 Assign each element of `Check` array to `TRUE` if in range **or** `FALSE` if not in range	

Question	Answer	Marks
5(a)	<table><tr><th>Activity</th><th>Name of life cycle stage</th></tr><tr><td>A compiler is be used.</td><td>Coding</td></tr><tr><td>A program that has been released for general use is modified</td><td>Maintenance</td></tr><tr><td>The dry run method is used.</td><td>Testing</td></tr><tr><td>The program structure is specified.</td><td>Design</td></tr></table> One mark per row	**4**
5(b)	One mark for method – two marks for the description **MP1** Stub testing **MP2** The modules `Test_A()` and `Test_B()`are replaced by dummy modules **MP3** ... which return a known result // An output statement is displayed when called (to check it works) // gives expected output	**3**

Question	Answer	Marks

6	Loop Solution
	Example solution:
	
	`FUNCTION AdjustClock(ThisYear : INTEGER) RETURNS INTEGER`
	`  DECLARE ThisDayNumber, SundayCount : INTEGER`
	`  DECLARE ThisDate : DATE`
	
	`  ThisDayNumber ← 0`
	`  SundayCount ← 0`
	`  REPEAT`
	`  ThisDayNumber ← ThisDayNumber + 1`
	`  ThisDate ← SETDATE(ThisDayNumber, 3, ThisYear)`
	`  IF DAYINDEX(ThisDate) = 1 THEN`
	`  SundayCount ← SundayCount + 1`
	`  ENDIF`
	`  UNTIL SundayCount = 3`
	`  RETURN ThisDayNumber`
	`ENDFUNCTION`
	
	Mark as follows:
	MP1 Function heading, parameter, ending and return type
	MP2 Declare local integer variable that is used to create a date
	MP3 Loop until 3rd Sunday found
	MP4 Attempt to use both `SETDATE()` **and** `DAYINDEX()` **in a loop**
	MP5 Correctly generate value of type `DATE` using `SETDATE()` **in a loop**
	MP6 Test if value represents a Sunday using `DAYINDEX()` **in a loop**
	MP7 Increment Sunday count **in a loop and** initialised correctly before loop
	MP8 Return day number of **third** Sunday
	
	Max 7 marks
	
	Non-loop solution
	Example solution:
	
	`FUNCTION AdjustClock(ThisYear : INTEGER) RETURNS INTEGER`
	`  DECLARE FirstDayIndex: INTEGER`
	`  DECLARE ThisDate : DATE`
	
	`  ThisDate ← SETDATE(1, 3, ThisYear)`
	`  FirstDayIndex ← DAYINDEX(ThisDate)`
	`  IF FirstDayIndex = 1 THEN`
	`  ThirdSunday ← FirstDayIndex + 14 //First day is Sunday`
	`  ELSE`
	`  ThirdSunday ← 23 – FirstDayIndex // Other days`
	`  ENDIF`
	`  RETURN ThirdSunday`
	`ENDFUNCTION`
		7

Question	Answer	Marks
6	Mark as follows: **MP1** Function heading, parameter, ending and return type **MP2** Declare local integer variable that is used to hold a day number **MP4** Attempt to use both `SETDATE()` **and** `DAYINDEX()` **MP5** Correctly generate value of type `DATE` using `SETDATE()` for first of March / specific day in March **MP6** Test if date represents a Sunday / specific day using `DAYINDEX()` and calculate third Sunday // Correctly calculate third Sunday for a value returned by `DAYINDEX()`for one day **MP7** Calculate third Sunday for other six days **MP8** Return day number of **third** Sunday **Max 7 marks**	

Question	Answer	Marks
7(a)	One mark per answer: **MP1** Analysis **MP2** **Named** document, examples include: • A requirement specification • Definition of System **Objectives** • List of problems with existing system • Survey results • Feasibility study • Interview notes **Max 2 marks**	**2**

Question	Answer	Marks
7(b)	One mark for name **and** use Mark as follows: Examples include: Sub-module: `ScanCard()/ GetID()` Use: Read the barcode from the loyalty card / Get the customer ID from the barcode Sub-module: `GetPoints()` Use: Get the number of points the customer has Sub-module: `ExchangePoints()` / `UpdateCard()` Use: Reduce the number of loyalty points Sub-module: `GetDiscount()` / `CalculateDiscount()` Use: Calculate the discount Sub-module: `CalculatePoints()` Use: Calculate points (following a purchase) Sub-module: `UpdatePoints()` Use: Update total points a customer has (following a purchase) Sub-module: `ShowDiscount()` Use: Display the discount **Max 4 marks**	**4**

Question	Answer	Marks

8(a)	Example solution
	
	`PROCEDURE Count(ThisPlayer : INTEGER, ThisRole : STRING)`
	`  DECLARE Index, Num, Total : INTEGER`
	
	`  Num ← 0`
	`  Total ← 0`
	
	`  FOR Index ← 1 TO 45`
	`  IF Character[Index].Player = ThisPlayer AND __`
	`  Character[Index].Role = ThisRole THEN`
	`  Num ← Num + 1`
	`  Total ← Total + Character[Index].SkillLevel`
	`  NEXT Index`
	
	`  IF Num > 0 THEN`
	`  OUTPUT "Player ", ThisPlayer, __`
	`  " has ", Num, " characters with the role of`
	`  ", ThisRole, " and the total skill level is`
	`  ", Total`
	`  ELSE`
	`  OUTPUT "No characters with that role are`
	`  assigned to this player"`
	`  ENDIF`
	
	`ENDPROCEDURE`
	
	MP1 Initialisation of local integers for `Num` and `Total`
	MP2 Loop through 45 elements
	
	MP3 Attempt to check Player and Role fields **in a loop**
	MP4 Correctly compare `Player` field with parameter **in a loop**
	MP5 Correctly compare `Role` field with parameter **in a loop**
	
	MP6 ... if player and role found, increment `Num` **and** sum Skill Total **in a loop**
	MP7 Test for any matches **after the loop**
	MP8 **Both** possible OUTPUT statements correctly formed following an attempt at MP6 but outputting one only
	
	Max 7 marks
		7

Question	Answer	Marks

8(b)	Example solution:
	
	`PROCEDURE Restore()`
	`  DECLARE Index : INTEGER`
	`  DECLARE Line  : STRING`
	
	`  OPENFILE "SaveFile.txt" FOR READ`
	`  FOR Index ← 1 TO 45`
	`  READFILE "SaveFile.txt", Line`
	`  Character[Index].Player ← STR_TO_NUM(Extract(Line, 1))`
	`  Character[Index].Role ← Extract(Line, 2)`
	`  Character[Index].Name ← Extract(Line, 3)`
	`  Character[Index].Skill ← STR_TO_NUM(Extract(Line, 4))`
	`  NEXT Index`
	
	`  CLOSEFILE "SaveFile.txt"`
	
	`ENDPROCEDURE`
	
	Mark as follows:
	MP1 Open the file in read mode and subsequently close
	MP2 Loop through 45 elements
	MP3 Read a line from the file **in a loop**
	MP4 Attempt to use `Extract()` **in a loop**
	MP5 Correct use of `Extract()` for all fields **in a loop**
	MP6 Use of `STR_TO_NUM()` on Player and Skill **in a loop**
	MP7 Completely correct extraction and assignment of all fields **in a loop**
		7

8(c)	**MP1** The `Character` array / character data can be saved before the program is closed **MP2** Allowing the **game to continue** using the same data / from the point it was saved	**2**