Cambridge Assessment International Education
Cambridge International Advanced Subsidiary and Advanced Level

COMPUTER SCIENCE **9608/22**

Paper 2 Written Paper **October/November 2018**

MARK SCHEME

Maximum Mark: 75

Published

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 descriptors 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.

Question	Answer	Marks
1(a)(i)	<table><tr><th>Statement</th><th>Selection</th><th>Repetition (Iteration)</th><th>Assignment</th></tr><tr><td>`Index ← Index + 5`</td><td></td><td></td><td>✓</td></tr><tr><td>`FOR Count ← 1 TO 100`</td><td></td><td>✓</td><td>(✓)</td></tr><tr><td>`TempValue[Index] ← ReadValue(SensorID)`</td><td></td><td></td><td>✓</td></tr><tr><td>`IF Index < 30`</td><td>✓</td><td></td><td></td></tr><tr><td>`UNTIL DayNumber > 7`</td><td></td><td>✓</td><td></td></tr><tr><td>`OTHERWISE OUTPUT "ERROR"`</td><td>✓</td><td></td><td></td></tr></table> 1 mark per correct row Ignore any tick in assignment column for second statement	**6**
1(b)(i)	<table><tr><th>Statement</th><th>Data type</th></tr><tr><td>`Revision ← 'B'`</td><td>CHAR</td></tr><tr><td>`MaxValue ← 13.3`</td><td>REAL</td></tr><tr><td>`ArrayFull ← TRUE`</td><td>BOOLEAN</td></tr><tr><td>`Activity ← "Design"`</td><td>STRING</td></tr><tr><td>`NumberOfEdits ← 270`</td><td>INTEGER</td></tr></table> 1 mark per correct row	**5**
1(b)(ii)	<table><tr><th>Expression</th><th>Evaluates to</th></tr><tr><td>`MID(Activity, 3, 4) & "ature"`</td><td>"signature"</td></tr><tr><td>`INT(MaxValue * 2)`</td><td>26</td></tr><tr><td>`ArrayFull AND NumberOfEdits < 300`</td><td>TRUE</td></tr><tr><td>`ASC(Revision + 1)`</td><td>ERROR</td></tr><tr><td>`Activity = "Testing" OR Revision = 'A'`</td><td>FALSE</td></tr></table> 1 mark per correct row	**5**

Question	Answer	Marks

2(a)(i)	
	```
	FUNCTION CalcPoints(CardNum: STRING, Total: REAL)
	                                         RETURNS INTEGER
	   DECLARE OldPoints : INTEGER
	   DECLARE NewPoints : INTEGER

	   IF Total > 100
	      THEN
	         OldPoints ← GetPoints(CardNum)
	         IF OldPoints > 2000
	            THEN
	               NewPoints ← INT(Total * 1.2)
	            ELSE
	               NewPoints ← INT(Total * 1.1)

	         ENDIF
	      ELSE
	         NewPoints ← 0
	   ENDIF

	   RETURN NewPoints

	ENDFUNCTION
	```
	1 mark for each of the following:
	
	1 Correct `FUNCTION` heading (as given) and end
	2 Declaring local variables for `OldPoints` and `NewPoints`
	3 `IF...THEN...ELSE...ENDIF with Total > 100`
	4 …Newpoints set to zero if Total <= 100
	5 Nested `IF...THEN...ELSE...ENDIF` with `OldPoints > 2000`
	6 … with correct assignments of NewPoints
	7 Return `NewPoints` for all cases
		7

Question	Answer	Marks

2(a)(ii)	`FUNCTION GetTotal() RETURNS REAL`
	
	   `DECLARE Valid : BOOLEAN`
	   `DECLARE Amount : REAL`
	   `Valid ← FALSE`
	
	   `REPEAT`
	      `OUTPUT "Enter the amount"`
	      `INPUT Amount`
	      `IF Amount > 0 AND Amount < 10000`
	         `THEN`
	            `Valid ← TRUE`
	      `ENDIF`
	   `UNTIL Valid = TRUE`
	
	   `RETURN Amount`
	
	`ENDFUNCTION`
	
	Note that the pseudocode shown is only an example. The use of an explicit flag and `IF` structure are not essential provided the functionality is provided.
	
	1 mark for each of:
	
	1 declaration of local variable(s) used
	2 prompt followed by input
	3 conditional loop
	4 checking that input value > 0 and input value < 10000
	5 returning the value
		5

2(b)(i)	1 mark for name, 1 mark for description Accept by example for description Name: Run-time Description: The program executes an illegal instruction // performs an illegal operation that is trapped by the OS	**2**
2(b)(ii)	One **specific value** from within each of the following ranges: For example: • 73 (a value <= 100) • 145.3 (a value > 100) 1 mark for the value, plus one for a meaningful description.	**4**

Question	Answer	Marks
3(a)	You should be able to recognise / understand in another language: • Declaration / assignment / sequence / selection / repetition (iteration) / Subroutines / Parameters passed between modules / program structure / Input and Output Any 2 marks from the list. Max 1 mark if no explanation given	**2**
3(b)	Key points: • to increase the level of detail of the algorithm // break the problem into smaller steps • until steps are easier to solve // to be directly translated into lines of code	**2**
3(c)	Key points: • A <u>loop / repetition / iteration</u> to check every element • Compare the array element with the value being searched • Exit loop / stop search when value found or end of array reached • If value is found then output the index position, otherwise output “Not found”	**4**

Question	Answer	Marks
4(a)(i)	(see table below) 1 mark for each correct answer	**5**
4(a)(ii)	Example mark points, max 4 marks: • If `FileData` is not empty, start and continue a loop // while `FileData` is not empty... • Compare the Left 7 characters of `FileData` with `SearchString` • If they match, add `FileData` to the array / `ResultArray[]` • If they match, increment the array index variable • Increment `FileLine` • Read the next line from the file	**4**

4(a)(i):

The identifier name of a local variable	`FileData / FileLine`
The identifier name of a user-defined procedure	`ScanCompleted`
The identifier name of a user-defined function	`ReadFileLine / ScanFile`
The number of dimensions of `ResultArray`	`1`
The scope of `FileData`	`Local`

Question	Answer	Marks

4(b)	Pseudocode solution included here for development and clarification of mark scheme.
	Programming language solutions appear at the end of this mark scheme.
	
	`FUNCTION ScanFile(SearchString STRING) RETURNS INTEGER`
	
	   `DECLARE FileData : STRING`
	   `DECLARE FileLine : INTEGER`
	   `DECLARE NewData : STRING`
	   `DECLARE Size : INTEGER`
	
	   `NextArrayElement ← 1`
	   `FileLine ← 1`
	
	   `FileData ← ReadFileLine("DataFile.txt", FileLine)`
	
	   `WHILE FileData <> ""`
	      `IF LEFT(FileData, 7) = SearchString`
	         `THEN`
	            `Size ← LENGTH(FileData)`
	            `NewData ← RIGHT(FileData,Size-7)`
	            `ResultArray[NextArrayElement] ← NewData`
	            `NextArrayElement ← NextArrayElement + 1`
	      `ENDIF`
	      `FileLine ← FileLine + 1`
	      `FileData ← ReadFileLine("DataFile.txt", FileLine)`
	   `ENDWHILE`
	
	   `CALL ScanCompleted()`
	   `RETURN FileLine`
	
	`ENDFUNCTION`
	
	1 mark for each of the following:
	
	1 Function heading and ending including parameters **and** return statement
	2 Local variable declarations `for FileData` **and** `FileLine` but **NOT** declaration of `NextArrayElement`
	3 initialisation of `FileLine` and `NextArrayElement` (allow 0 or 1)
	4 `WHILE ... ENDWHILE` loop
	5 Comparing `SearchString` with first seven characters of `FileData`
	6 Use of substring function **and** subtraction of 7 from length
	7 Assign `NewData` to array `ResultArray` following attempt at MP6
	8 Increment `NextArrayElement` **and** `FileLine` (as above)
		8

4(c)(i)	Through the use of: Subroutines / Functions / Procedures / Parameters / Methods	**1**
4(c)(ii)	Reduces program complexity // easier to develop / test / debug // tasks may be re-used // tasks can be allocated to different programmers/teams (with different skills) // limited scope of local variables	**1**

Question	Answer	Marks

4(d)	Pseudocode solution included here for development and clarification of mark scheme.
	Programming language solutions appear at the end of this mark scheme.
	
	`DECLARE ResultArray : ARRAY [1:100] OF STRING`
	`DECLARE Index: INTEGER`
	
	`FOR Index ← 1 TO 100`
	`   ResultArray[Index] ← "NO DATA"`
	`ENDFOR`
	
	1 mark for each of the following:
	
	• `ResultArray` declaration / commented in Python
	• Loop for 100 elements
	• …assignment of string `"NO DATA"` to indexed array element
		3

Question	Answer	Marks

5	
	```
	PROCEDURE LineNumber(FileName: STRING,
	              StartNumber: INTEGER, StepNumber: INTEGER)

	   DECLARE FileData : STRING
	   DECLARE Count : INTEGER
	   DECLARE Reply : CHAR
	   DECLARE Continue : BOOLEAN

	   Count ← 0
	   Continue ← TRUE

	   OPENFILE FileName FOR READ

	   WHILE NOT EOF(FileName) AND Continue = TRUE

	      READFILE FileName, FileData
	      FileData ← NUM_TO_STRING(StartNumber) & ": " &
	                                                FileData
	      OUTPUT FileData
	      Count ← Count + 1

	      IF Count = 20 THEN
	         OUTPUT "Do you wish to continue?"
	         INPUT Reply
	         IF Reply = 'N'
	            THEN
	               Continue ← FALSE
	            ELSE
	               Count ← 0
	         ENDIF
	      ENDIF
	      StartNumber ← StartNumber + StepNumber

	   ENDWHILE

	   CLOSEFILE FileName
	ENDPROCEDURE
	```
		11

Question	Answer	Marks
5	1 mark for each of the following to max 11: 1 Procedure heading including parameters 2 Declare an integer variable for the count 3 Open file in `READ` mode 4 Initialise `Count` variable before loop **and** increment in the loop (or other mechanism) 5 Loop including until `EOF(FileName)` (see note) 6 Call `READFILE()` **in a loop** 7 Convert `StartNumber` to string **in a loop** 8 Output concatenated string **in a loop** 9 Check if count = 20 10 Prompt and Input (inside an IF) 11 If user input = 'N' terminate loop 12 Add `StepNumber` to `StartNumber` 13 Close file	

*** End of Mark Scheme – program code solutions follow ***

Program Code Solutions

Q4 (b): Visual Basic

```
Function ScanFile(SearchString As String) As Integer

   Dim FileData As String
   Dim FileLine As Integer
   Dim NewData As String
   Dim Size As Integer

   NextArrayElement = 1
   FileLine = 1

   FileData = ReadFileLine("DataFile.txt", FileLine)

   Do While FileData <> ""
      If Left(FileData, 7) = SearchString Then
         Size = Len(FileData)
         NewData = Right(FileData,Size-7) // NewData = Mid(FileData, 8,
                                                                  Size-7)
         ResultArray(NextArrayElement) = NewData
         NextArrayElement = NextArrayElement + 1
      End If
      FileLine = FileLine + 1
      FileData = ReadFileLine("DataFile.txt", FileLine)
   Loop

   Call ScanCompleted() ' Keyword "Call" OK but not required
   Return FileLine

End Function
```

Q4 (b): Pascal

```
function ScanFile(SearchString : string) : integer;
   var
      FileData : string;
      FileLine : integer;
      NewData : string;
      Size : integer;

   begin
      NextArrayElement := 1;
      FileLine := 1;

      FileData  := ReadFileLine('DataFile.txt', FileLine);

      while FileData <> '' do
      begin
         if leftstr(FileData, 7) = SearchString then
            begin
               Size := Length(FileData);
               NewData := rightstr(FileData,Size-7);
               ResultArray[NextArrayElement] := NewData;
               NextArrayElement := NextArrayElement + 1;
            end;
         FileLine := FileLine + 1;
         FileData := ReadFileLine("DataFile.txt", FileLine);
      end;

   ScanCompleted(); // Keyword "Call" not valid
   Return FileLine; // ScanFile := FileLine;
   end;
```

Q4 (b): Python

```
def scanfile(searchstring):

   # filedata : string
   # fileline : integer
   # newdata : string

   nextarrayelement = 1
   fileline = 1

   filedata = readfileline("datafile.txt", fileline)

   while filedata != "" :
      if filedata[:7] == searchstring:
         newdata = filedata[7:]
         resultarray[nextarrayelement] = newdata
         nextarrayelement = nextarrayelement + 1
      fileline = fileline + 1
      filedata = readfileline("datafile.txt", fileline)

   scancompleted() # keyword "call" not valid ???
   return fileline
```

Q4 (d): Visual Basic

```
Dim ResultArray(99) As String
Dim Index As Integer
For Index = 0 To 99
   ResultArray(Index) = "NO DATA"
Next Index
```

Q4 (d): Pascal

```
var
   ResultArray : array [1..100] of string;
   Index : integer;

begin
for Index := 1 to 100 do
   ResultArray[Index] := 'NO DATA';

end.
```

Q4 (d): Python – alternative 1 of n

```
#ResultArray[] as STRING

Resultarray = ["NO DATA" for index in range(100)]
```

Q4 (d): Python – alternative 2 of n

```
#ResultArray[] as STRING

ResultArray = []
For Index in range(100):
   ResultArray.append("NO DATA")
```

Q4 (d): Python – alternative 3 of n

```
# ResultArray[99] As String

ResultArray = ["NO DATA"]*100
```