Cambridge International AS & A Level

Cambridge Assessment International Education
Cambridge International Advanced Subsidiary and Advanced Level

COMPUTER SCIENCE **9608/21**

Paper 2 Written Paper **October/November 2019**

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.

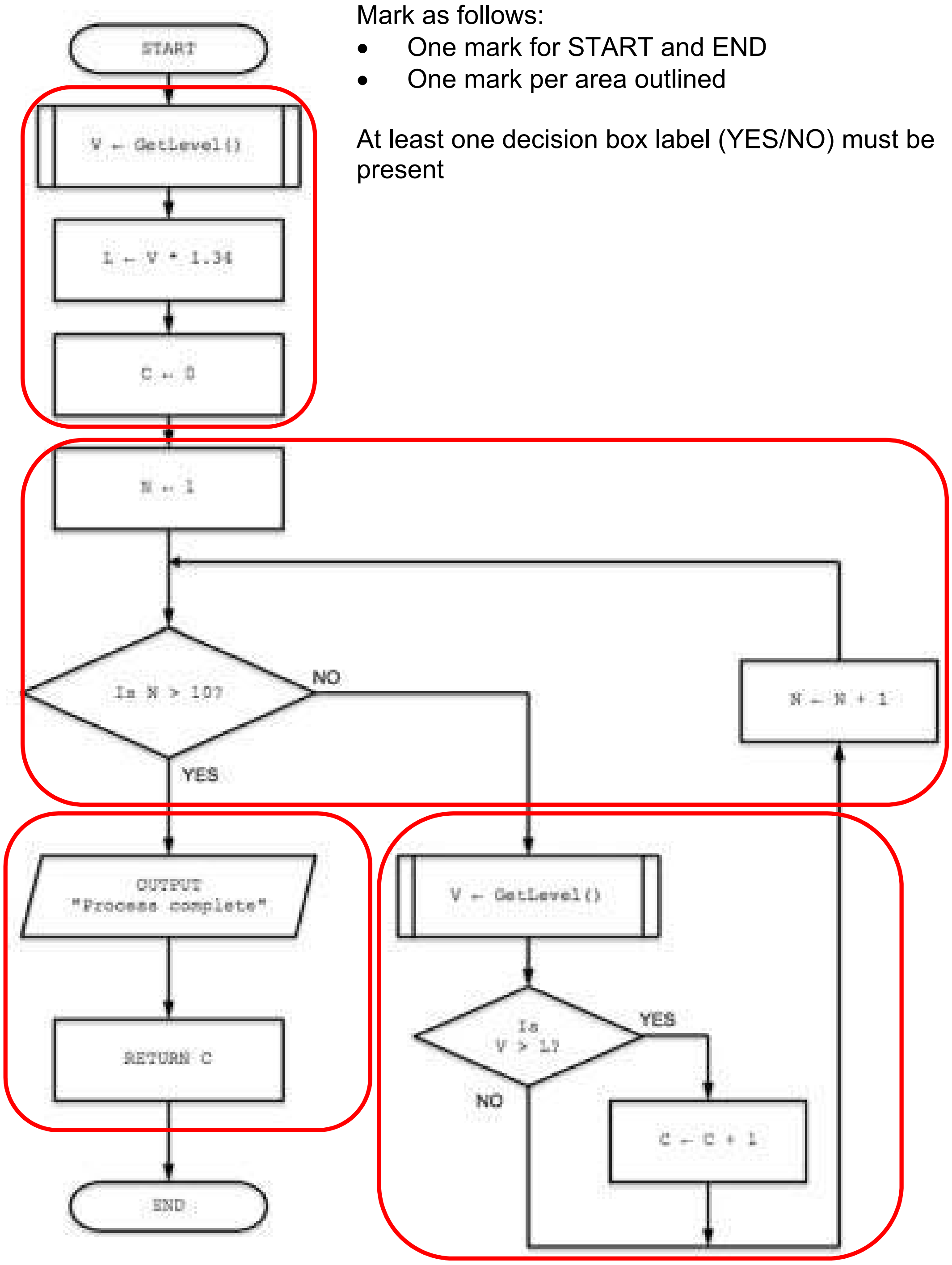
GENERIC MARKING PRINCIPLE 5:

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)	One mark for each feature: 1. meaningful / sensible identifier names // use of Camel case for identifier names // use of constants 2. blank lines / white space 3. comments	**3**

Question	Answer	Marks
1(a)(ii)	 START V ← GetLevel() L ← V * 1.34 C ← 0 N ← 1 Is N > 10? NO YES N ← N + 1 OUTPUT "Process complete" RETURN C V ← GetLevel() Is V > L? YES NO C ← C + 1 END Mark as follows: • One mark for START and END • One mark per area outlined At least one decision box label (YES/NO) must be present	5

Question	Answer	Marks
1(b)(i)	One mark per row **Example value** | **Data type** "NOT TRUE" | **STRING** −4.5 | **REAL** NOT FALSE | **BOOLEAN** 132 | **INTEGER**	**4**
1(b)(ii)	One mark per row **Expression** | **Evaluates to** `LEFT("Start", 3) & RIGHT("Apple", 3)` | **"Staple"** `MID("sample", 3, 5)` | **ERROR** `NUM_TO_STRING(12.3 * 2)` | **"24.6"** `INT(STRING_TO_NUM("53.4")) + 7` | **60**	**4**

Question	Answer	Marks
2(a)	One mark for each feature: 1. Module <u>hierarchy</u> 2. The <u>parameters</u> that are passed (between modules) // the module <u>interface</u> 3. Selection / Decisions (which modules are executed) 4. Iteration / Repetition	**4**
2(b)	One mark for name and one mark for explanation. Example: • PrettyPrint // Colour coding • Colour coding of command words / key words • Expand and collapse code blocks • Allows programmer to focus on a section of code // allows quicker navigation of the code • Auto(matic) indentation • Allows the programmer to clearly see the different code sections / easier to see the code structure Accept suitable alternatives	**2**
2(c)	One mark for identification, one mark for description: • By reference / ref • The <u>address</u> of / <u>pointer</u> to the parameter is passed to the subroutine // if the parameter value is changed in the subroutine this changes the original value	**2**
2(d)	One mark per bullet point: • Changes made to // Updating // Editing a program / algorithm / data structure / software / system • ...as a result of changes to requirements / specification / legislation / available technology	**2**

Question	Answer	Marks
3	One mark per row:	**7**

	Answer
The number of the line containing a variable being	**`24 / 26 / 28`**
The range of line numbers containing a pre-condition loop	**`20 – 30`**
The number of initialisation statements	**`3`**
The number of the line containing a logical operator	**`20`**
The range of line numbers containing a selection statement	**`22 - 27 / 32 - 37`**
The name of a built-in function	**`MID / LENGTH`**
The name of a parameter	**`InString / Index`**

Question	Answer	Marks
4(a)	One mark for process name, **max 3** for structured English. **Process:** • Stepwise Refinement / Top-down design **Structured English:** • Check that character is between 'A' and 'Z' • Produce unique array index for this character • Increment this array element	**4**

4(b)	`DECLARE Index : INTEGER`
	`DECLARE Count : INTEGER`
	
	`FOR Count ← 1 TO LENGTH(InString)`
	`   NextChar ← UCASE(MID(InString, Count, 1))`
	`   IF NextChar >= 'A' AND NextChar <= 'Z'`
	`      THEN`
	`         Index ← ASC(NextChar) - 64`
	`         Result[Index] ← Result[Index] + 1`
	`   ENDIF`
	`ENDFOR`
	
	`FOR Index ← 1 TO 26`
	
	`   OUTPUT "Letter " & CHR(Index + 64) & " : "`
	`                    & NUM_TO_STRING(Result[Index])`
	`ENDFOR`
	
	One mark for each of the following (**max 7**):
	
	1 First loop from 1 to length of `InString`:
	2 Extract each character in turn **in a loop**
	3 Check that character is alphabetic (must cater for lower & upper case) **in a loop**
	4 Obtain array index using `ASC() - 64` **in a loop**
	5 Increment element of `Result` array **in a loop**
	6 Second loop from 1 to 26:
	7 Attempt to `OUTPUT` character A to Z **and** corresponding count **in a loop**
	8 Fully complete `OUTPUT` including any necessary type conversion **in a loop**
		7

Question	Answer	Marks
5(a)	One mark for each point. A valid string must contain: • At least two // more than one upper case character(s) • At least five // more than four lower case character(s) • More digit characters than 'other' characters	**3**

Question	Answer	Marks
5(b)(i)	One mark for each area as outlined:	5

Index	NextChar	Upper	Lower	Digit	Other
		0	0	0	0
1	'J'	1			
2	'i'		1		
3	'm'		2		
4	'+'				1
5	'S'	2			
6	'm'		3		
7	'i'		4		
8	't'		5		
9	'h'		6		
10	'*'				2
11	'9'			1	
12	'9'			2	

Question	Answer	Marks
5(b)(ii)	One mark per bullet point: • Returned value is `FALSE` • `Digit – Other` is not greater than zero // Number of `Digit` same as `Other`	2

Question	Answer	Marks
6(a)	To retain data when the computer is shut down / turned off // after the program ends Accept equivalent answer.	1

6(b)	'Pseudocode' solution included here for development and clarification of mark scheme.
	Programming language example solutions appear in the Appendix.
	
	`FUNCTION SearchFileNtoZ(AccNum : STRING) RETURNS BOOLEAN`
	`  DECLARE FileData : STRING`
	`  DECLARE Found : BOOLEAN`
	`  CONSTANT SearchFile = "UserListNtoZ.txt"`
	`  Found ← FALSE`
	
	`  OPENFILE SearchFile FOR READ`
	
	`  WHILE NOT EOF(SearchFile) AND NOT Found`
	
	`    READFILE SearchFile, FileData`
	`    IF AccNum & '*' = LEFT(FileData, LENGTH(AccNum)+ 1)`
	`       THEN`
	`          Found ← TRUE`
	`    ENDIF`
	
	`  ENDWHILE`
	
	`  CLOSEFILE SearchFile`
	
	`  RETURN Found`
	
	`ENDFUNCTION`
	
	One mark for each of the following:
	
	1. Function heading **and** ending, (ignore parameter) and returned `BOOLEAN`
	2. File `OPEN UserListNtoZ.txt` in `READ` mode **and** `CLOSE`
	3. Conditional loop repeating until `EOF()` **or** '`Found`'
	4. Read a line from the file **in a loop**
	5. Compare the correct number of characters with `AccNum` **in a loop**
	6. Set termination logic if found **in a loop**
	7. Return Boolean value
		7

Question	Answer	Marks

6(c)	
	```
	PROCEDURE FindDuplicates()

	  DECLARE Index : INTEGER
	  DECLARE FileData : STRING
	  DECLARE Continue : BOOLEAN
	  DECLARE AccNum : STRING

	  Index ← 1 // assuming array is [1:100]
	  Continue ← TRUE
	  OPENFILE "UserListAtoM.txt" FOR READ

	  WHILE NOT EOF("UserListAtoM.txt") AND Continue = TRUE

	    READFILE "UserListAtoM.txt", FileData

	    IF MID(FileData, 7, 1) = '*' // six character reference
	      THEN
	        AccNum ← LEFT(FileData, 6)
	      ELSE
	        AccNum ← LEFT(FileData, 9)
	    ENDIF

	    IF SearchFileNtoZ(AccNum) = TRUE
	      THEN
	        IF Index = 101 // is the array already full?
	          THEN
	            OUTPUT "Error - Array Full"
	            Continue ← FALSE
	          ELSE
	            Duplicates[Index] ← AccNum
	            Index ← Index + 1
	        ENDIF
	    ENDIF

	  ENDWHILE

	  CLOSEFILE "UserListAtoM.txt"

	ENDPROCEDURE
	```
	One mark for each of the following (**max 8**):
	
	1. Declaration **and** Initialisation of `Index` **and** used to index array `Duplicates`
	2. `OPEN` file `UserListAtoM.txt` in `READ` mode **and** `CLOSE`
	3. Pre-Condition loop to go through the file until `EOF()` **and** early termination if array full
	4. Read line from file **and** extract account number (`AccNum`) **in a loop**
	5. Call `SearchFileNtoZ` (with `AccNum`) following an attempt at MP4 **in a loop**
	6. Check if return value is TRUE and if so: **in a loop**
	7. store AccNum in correct array element
	8. increment array `index` following an attempt at MP7
	9. If array overflow `OUTPUT` error message
		8

Question	Answer	Marks

6(d)(i)	
	```
	PROCEDURE ClearArray(BYREF ThisArray : ARRAY,
	                    NumElements : INTEGER, InitVal : STRING)

	  DECLARE Index : INTEGER
	  FOR Index ← 1 TO NumElements
	     ThisArray[Index] ← InitVal
	  ENDFOR

	ENDPROCEDURE
	```
	Mark as follows:
	• Procedure header
	• Loop
	• Assignment within loop
		3

6(d)(ii)	'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix. <u>`CALL ClearArray(Duplicates, 100, "Empty")`</u> Mark as follows: • Procedure call • Parameter list (in brackets)	**2**

Program Code Example Solutions

Question 6(b): Visual Basic

```
Function SearchFileNtoZ(ByVal SearchString As String) As Boolean
  Dim FileData As String
  Dim Found As Boolean

  Found = FALSE

  FileOpen(1, "UserListNtoZ.txt", OpenMode.Input)

  While Not EOF(1) And Not Found

    Filedata = LineInput(1)
    If SearchString & '*' = Left(FileData, Len(SearchString)+1) Then
      Found = TRUE
    End If

  End While

  FileClose(1)

  Return Found

End Function
```

Question 6(b): Pascal

```
function SearchFileNtoZ (SearchString : string): boolean;
  var
     FileData : string;
     Found : boolean;
     MyFile : text;

  begin

     Found := FALSE;

     assign(MyFile, "UserListNtoZ.txt");
     reset (Myfile);

     while Not EOF(MyFile) And Not Found do
     begin
        readLn(MyFile, FileData);
        if SearchString + '*' = LeftStr(FileData, length(SearchString)+1)
then
           Found := TRUE;

     end;

    close(MyFile);

    result := Found; // SearchFileB := Found;

  end;
```

Question 6(b): Python

```
def SearchFileNtoZ(SearchString):
   ## FileData : String
   ## Found : Boolean
   ## MyFile : Text

     Found = False

     MyFile = open("UserListNtoZ.txt", 'r')
     FileData = MyFile.readline()
     while Filedata !=  "" and not Found :
        if SearchString + '*' == FileData[0: len(SearchString)+1]:
           Found = True

        FileData = MyFile.readline()

     MyFile.close

    return(Found)
```

Question 6(d)(ii): Visual Basic

```
Call ClearArray(Duplicates, 100, "Empty") 'Call optional
```

Question 6(d)(ii): Pascal

```
ClearArray(Duplicates, 100, 'Empty');
```

Question 6(d)(ii): Python

```
ClearArray(Duplicates, 100, "Empty")
```