CAMBRIDGE INTERNATIONAL EXAMINATIONS

Cambridge International Advanced Subsidiary and Advanced Level

MARK SCHEME for the October/November 2015 series

9691 COMPUTING

9691/23 Paper 2 (Written Paper), maximum raw mark 75

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 will not enter into discussions about these mark schemes.

Cambridge is publishing the mark schemes for the October/November 2015 series for most Cambridge IGCSE®, Cambridge International A and AS Level components and some Cambridge O Level components.

CAMBRIDGE
International Examinations

1 (a)

Field	Identifier	Data type	Example of input data	Field size (in bytes)	marks
course code	CourseCode	STRING (not text/alphanumeric)	015110217	10 (approx.) (accept a range)	1
title	Title	STRING (not text/alphanumeric)	Programming for Beginners	30 (approx.) (accept a range)	1
tutor (3-letter initials)	Tutor	STRING (not text/alphanumeric)	PGL	3/6	
day of week	Day	BYTE / INTEGER Accept CHAR/STRING(1)	2	1 / 4 1 / 2	1
lab based?	IsLabBased	BOOLEAN	TRUE	1 / 2	1
session duration in hours	SessionHours	REAL/FLOAT/SINGLE	2.5	4 / 8	1
fee ($)	CourseFee	CURRENCY/FLOAT/DECIMAL SINGLE/REAL/DOUBLE	25.50	8 / 16	1
date course starts	StartDate	DATE/REAL (STRING)	03/11/2015	8 (10)	1
date course ends	EndDate	DATE/REAL (STRING)	03/12/2015	8 (10)	

[max 5]

(b) *Mark as follows:*
1 mark for correct record header
1 mark for correct definition terminator
1 mark for the first 5 fields defined correctly for language
1 mark for the remaining 4 fields defined correctly for language

Do not accept pseudocode
Field names must be as given, but ignore capitalisation/spaces
Declared program language must match code given
Ignore field sizes and data type
If misused DIM in VB, penalise once
If statement separators missing, penalise once

Example Pascal

```
TYPE CourseRecordType = RECORD
   CourseCode: STRING[10];
   Title: STRING[30];
   Tutor: STRING[3];
   Day: BYTE;
   IsLabBased: Boolean;
   SessionHours: REAL;
   CourseFee: Currency;
   StartDate: TDateTime;
   EndDate: TDateTime;
END;
```
[4]

(c) Note that some candidates may already have done this in part (b). In that case, give marks according to part (b).

```
VAR Course : ARRAY[1..50] OF CourseRecordType
```
[2]

```
VAR DummyRecord : CourseRecordType

WITH DummyRecord DO
   BEGIN
      CourseCode :='';
      Title := '';
      Tutor := '';
      Day := 0;
      IsLabBased := FALSE;
      SessionHours := 0;
      CourseFee := 0;
      StartDate:= 01/01/2010
      EndDate := 01/01/2010
   END;
FOR i := 1 to 50 DO
   Course[i] := DummyRecord;
```
[1] [1] [1] [1] [1]

Alternative:

```
VAR Course : ARRAY[1..50] OF CourseRecordType                    [2]
FOR i := 1 to 50 DO                                              [1]
   BEGIN
      Course[i].CourseCode :='';
      Course[i].Title := '';                                     [1]
      Course[i].Tutor := '';
      Course[i].Day := 0;
      Course[i].IsLabBased := FALSE;
      Course[i].SessionHours := 0;                               [1]
      Course[i].Course Fee := 0;
      Course[i].StartDate:= 0;
      Course[i].EndDate := 0;                                    [1]
   END;
         [1]
```

[max 6]

Do not penalise again for incorrect data type

(d) (i) – `EOF()` returns TRUE or FALSE
– depending on whether it found the marker at the end of the file [2]

(ii) Mark as follows:
– Open file CourseData.DAT
– … for reading/input

– loop while not end of file CourseData.DAT
– read record from file
– assign to array element
– correctly initialised and incremented index
– Close file CourseData.DAT

Example pseudocode:

```
OPENFILE CourseData.DAT for READING // for INPUT
i ← 1
WHILE NOT EOF(CourseData.DAT)
   READ record FROM FILE
   Course[i] ← record
   i ← i + 1
ENDWHILE
CLOSEFILE CourseData.DAT
```

[max 6]

(e) Mark as follows:
- change outer FOR loop to a REPEAT/WHILE loop
- decrementing the iterations of the FOR loop
- introduce a Boolean variable `NoSwaps` (or similar)
- initialise Boolean variable correctly (inside outer loop and outside inner loop)
- terminate REPEAT loop when no swaps made
- leave comparison and swapping code the same
- change upper limit of inner loop to `NumberOfCourses` – x (instead of 49)

Example pseudocode:

```
PROCEDURE SortData(NumberOfCourses)
   x ← 0
   // NoSwaps ← FALSE       (required for WHILE loop)
   REPEAT  // WHILE NoSwaps = FALSE
      x ← x + 1
      NoSwaps ← TRUE
      FOR y ← 1 TO NumberOfCourses - x
         IF Course[y].CourseFee > Course[y + 1].CourseFee
            THEN
               NoSwaps ← FALSE
               TempRecord ← Course[y]
               Course[y] ← Course[y + 1]
               Course[y + 1] ← TempRecord
         ENDIF
      ENDFOR
   UNTIL NoSwaps = TRUE   // ENDWHILE
ENDPROCEDURE
```

[max 6]

2 (a) (i) *Mark as follows:*

- *parameter*
- *Return data type*
- *Correctly formed CASE statement (including the end)*
  - *with all cases present (characters in quotes)*
  - *ELSE clause*
- *Return of value (implied)*

Example PASCAL

```
FUNCTION DenaryDigit(Letter : CHAR) : INTEGER;
   BEGIN
      CASE Letter OF
         'K': DenaryDigit := 0;
         'D': DenaryDigit := 1;
         'L': DenaryDigit := 2;
         'C': DenaryDigit := 3;
         'F': DenaryDigit := 4;
         'H': DenaryDigit := 5;
         'B': DenaryDigit := 6;
         'G': DenaryDigit := 7;
         'E': DenaryDigit := 8;
         'A': DenaryDigit := 9;
      ELSE
         DenaryDigit:= -1
      END;
   END;
```

[max 5]

(ii)

`Letter`	**Expected result**	**Type of data** (normal, borderline or invalid)
`'1'`	–1	Invalid (digit)
`'X'`	–1	Invalid (letter)
`'G'`	7	normal

1 mark per row [3]

(b) (i) Mark as follows:
1 mark per column (2 to 6)
if zero marks then mark by row

CodedNumber	Denary	i	ThisChar	ThisNumber	OUTPUT
"LED"	0				
	20	1	L	2	
	100	2	E	8	
	110	3	D	1	110

[5]

(ii) line number 08
`Denary ← Denary * 10 + ThisNumber` [2]

Do not accept concatenation of separate digits (unless they are CHAR/STRING)

(iii) logic error [1]

(iv) Second and third mark dependent on first mark.

When and how interchangeable
Type: – syntax error
When: – during compilation of program // in IDE environment // running an interpreted program
How: – reported by the translator diagnostics // highlights/stops at the statement with the syntax error // compiler/interpreter checks against syntax rules / rules of the language
Type: – run-time error
When: – during testing/execution
How: – program will 'crash' e.g. attempted 'divide **by zero**' error [6]

(c) (i)
- Meaningful variable names
- Capitalisation of keywords
- Empty lines
- Use of indentation
- Initialisation (of variables)
- Use of (library/built-in) functions [max 3]

Do not accept white space / camel case on its own.

(ii) Comments/annotations/remarks [1]

(iii) 1 mark per line of pseudocode correctly written in the high-level language chosen.
1 mark for declarations:
Example Pascal:

```
PROCEDURE ConvertToDenary(CodedNumber: STRING);              //  [1]
VAR Denary, ThisNumber, i : INTEGER;
   ThisChar : CHAR;                                          //  [1]

BEGIN
   Denary := 0;                                              //  [1]

   FOR i := 1 TO LENGTH(CodedNumber)DO                       //  [1]
      BEGIN
         ThisChar := MIDSTR(CodedNumber, i, 1);              //  [1]
         ThisNumber := DenaryDigit(ThisChar);                //  [1]
         Denary := Denary + ThisNumber * 10;                 //  [1]
                          Accept 'corrected version'
      END;                                                   //  [1]

   WriteLn(Denary);                                          //  [1]
END;                                                         //  [1]
```

[10]

(iv)
- IF `ThisNumber` = –1 THEN
- output statement giving the error message
- instead of OUTPUT Denary
- exit from the loop [3]

3 **(a)** *Give credit for answers to "why" rather than "how".*

(i) Set a breakpoint in the program code
Execution will pause at this point [2]

(ii) Stepping allows one statement to be executed at a time
The program execution pauses after each statement
Often used from a set breakpoint
Can use variable watch at each step
Stepping over to skip statements [Max 2]

(iii) Variable watch allows tester to see the current contents of a variable
// Used to see how variable contents change when stepping through program
Tester chooses variables to watch [2]

(b) White-box [1]