Cambridge International AS & A Level

Cambridge International Examinations
Cambridge International Advanced Subsidiary and Advanced Level

COMPUTING **9691/23**

Paper 2 Written Paper **May/June 2016**

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 will not enter into discussions about these mark schemes.

Cambridge is publishing the mark schemes for the May/June 2016 series for most Cambridge IGCSE®, Cambridge International A and AS Level components and some Cambridge O Level components.

1 (i) **[2]**

Identifier	Data type	Explanation
LowerTemp	INTEGER	Lower bound value of Fahrenheit temperatures
UpperTemp	INTEGER	Upper bound value of Fahrenheit temperatures
Interval	INTEGER	The interval between two Fahrenheit temperatures
Fahrenheit	INTEGER	The Fahrenheit to be converted
Result	REAL/FLOAT	Value of conversion before rounding
Celsius	INTEGER	The rounded Result

Mark as follows:

5 × INTEGER (1)

Result – REAL (1)

(ii) **[6]**

```
INPUT LowerTemp
INPUT UpperTemp
INPUT Interval
OUTPUT "Conversion Table"
OUTPUT "Fahrenheit Celsius"
Fahrenheit ← LowerTemp
REPEAT
Result ← (Fahrenheit – 32) * 5 / 9
Celsius ← ROUND(Result)
OUTPUT Fahrenheit,"", Celsius
Fahrenheit ← Fahrenheit + Interval
UNTIL Fahrenheit > UpperTemp
```

Mark as follows:

- Fahrenheit ← LowerTemp
- (Fahrenheit – 32)
- * 5 / 9
- Celsius ← ROUND(Result)
- Fahrenheit ← Fahrenheit + Interval
- UNTIL Fahrenheit > UpperTemp

2 (a) Example Pascal **Max [8]**

```
PROCEDURE VendingMachine;
VAR SnackType: CHAR;
VAR Price, CoinValue, Total: INTEGER;

BEGIN
   ReadLn(SnackType);

   CASE SnackType OF
        'A': Price: = 20;
        'B': Price: = 40;
        'C': Price: = 50;
        'D': Price: = 80;
   END;

   Total: = 0;
   REPEAT
      ReadLn(CoinValue);
      Total:= Total + CoinValue;
   UNTIL Total >= Price;

   ChangeDue:= Total - Price;
   IF ChangeDue > 0
      THEN
         OutputChange(ChangeDue)
      ELSE
         WriteLn("No Change");
END;
```

Mark as follows:

– Procedure heading & ending
– Local variables declared
– With correct data types
– Input SnackType
– Correct case statement
– Initialise Total
– REPEAT loop
– Input CoinValue and keep running total
– Calculate change due
– IF statement with procedure call to OutputChange()
– Output “No change”

(b) (i) **[4]**

Statement	Value	Explanation
`X ← ChangeDue DIV 50`	`X = 1`	`X` represents **the number of 50-cent coins**
`Y ← ChangeDue MOD 50`	`Y = 30`	`Y` represents **the remaining change due**

(ii) Example Pascal **[7]**

```
PROCEDURE OutputChange(ChangeDue: INTEGER);
VAR Coins, LeftOver : INTEGER;
BEGIN
   Coins50: = ChangeDue DIV 50;
   WriteLn("Number of 50c coins: ", Coins50);
   LeftOver: = ChangeDue MOD 50;
   Coins20: = LeftOver DIV 20;
   WriteLn("Number of 20c coins:", Coins20);
   LeftOver: = LeftOver MOD 20;
   Coins10: = LeftOver DIV 10;
   WriteLn("Number of 10c coins:", Coins10);
END;
```

Mark as follows:
- Procedure heading including parameter
- Number of 50-cent coins calculated
- Calculate 'leftovers' after 50 cents correctly
- Number of 20-cent coins calculated
- Calculate 'leftovers' after 20 cents correctly
- Number of 10-cent coins calculated
- Output all numbers of coins needed

3 (a) (i) **[7]**

```
PROCEDURE InputScoresAndCalcAverage(BYREF Average : REAL)
    DECLARE x, TotalScore: INTEGER
    TotalScore ← 0
    FOR x ← 1 TO 10
        OUTPUT "Score for", CompetitorName [x]
        // input scores into the Score array
        INPUT Score [x]
        TotalScore ← TotalScore + Score [x]
    ENDFOR
    Average ← TotalScore / 10
ENDPROCEDURE
```

(ii) **[8]**

```
PROCEDURE UpdatePointsTotals(Average : REAL)
    DECLARE i: INTEGER
    FOR i ← TO 10
       IF Score[i] > Average
          THEN // increase PointsTotal
             PointsTotal [i] ← PointsTotal [i] + 1
          ELSE // below average?
             IF Score [i] < Average
                THEN
                   PointsTotal [i] ← PointsTotal [i] – 1
             ENDIF
       ENDIF
    ENDFOR
ENDPROCEDURE
```

(iii) **[2]**

– the address of `Average` gets passed ……

– so that its value is returned to the calling program

(b) (i) **[9]**

x	NoMoreSwaps	PointsTotal[x] < PointsTotal[x+1]	Temp	PointsTotal									
				[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
				+5	+3	+4	+2	0	+1	−2	−1	−1	−3
	TRUE												
1		FALSE											
2	FALSE	TRUE	+3		+4	+3							
3		FALSE											
4		FALSE											
5		TRUE	0					+1	0				
6		FALSE											
7		TRUE	-2							-1	-2		
8		TRUE	-2								-1	-2	
9		FALSE											
	TRUE												
1		FALSE											
2		FALSE											
3		FALSE											
4		FALSE											
5		FALSE											
6		FALSE											
7		FALSE											
8		FALSE											
9		FALSE											

(ii) – when sorting the table **[2]**
– only swapped the points total, not the name or the score

(c) (i) Example Pascal **Max [5]**

```
TYPE CompetitorRecord = RECORD
            CompetitorName: STRING[20];
            CompetitorTelNumber: STRING[15];
            DateOfBirth: TDATETIME;
            GameScores: ARRAY[1..8] OF INTEGER;
            PointsTotal: INTEGER;
END;
```

Mark as follows:
- record header & ending
- `CompetitorName, CompetitorTelNumber`
- `DateOfBirth`
- `GameScores … INTEGER`
- `ARRAY[1..8]`
- `PointsTotal`

(ii) Example Pascal **[3]**

```
VAR CompetitorData: ARRAY[1..10] OF CompetitorRecord
```

Mark as follows:
- array name declaration
- array dimension
- data type

(d) **[7]**

```
FUNCTION FindCompetitorRank(SearchName: STRING) RETURNS INTEGER
DECLARE i : INTEGER
   i ← 0
   REPEAT
      i ← i + 1
   UNTIL CompetitorData [i].CompetitorName = SearchName
   RETURN i
ENDFUNCTION
```

(e) Example Pascal **Max [5]**

```
PROCEDURE SaveToFile;
BEGIN
   VAR CompFile: FILE OF CompetitorRecord;
   VAR i: INTEGER;
   ASSIGNFILE (CompFile, 'CompetitorFile.DAT';
   REWRITE (CompFile);
   FOR i: = 1 TO 10 DO
      WRITE(CompFile, CompetitorData[i]);
   CLOSEFILE(CompFile);
END;
```

Mark as follows:

– Procedure heading and ending
– Declaration of local variables
– Assigning a file name
– Open file for writing
– Nested loop to access each array element
– Write element out to file
– Close file