Cambridge International AS Level

Cambridge International Examinations
Cambridge International Advanced Subsidiary Level

COMPUTING **9691/21**

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 (a) Example Pascal: **[4]**

```
PROCEDURE PrintNameLine(Symbol : CHAR, Name : STRING)
BEGIN
   Write(Symbol, '  ');
   WRITE(Name);
   WriteLn('  ', Symbol);
END;
```

Mark as follows:
- correct procedure header & ending
- output symbol at either end of line
- output name
- output (2) spaces either side between name and symbol

(b) Example Pascal: **[6]**

```
PROCEDURE PrintSymbolLine(Symbol : CHAR, LabelWidth : INTEGER)
VAR i : INTEGER;
BEGIN
   FOR i := 1 TO LabelWidth DO
      Write(Symbol);
   WriteLn();
END;
```

Mark as follows:
- correct procedure headings and endings
- output line of symbols to correct length
- followed by newline

```
PROCEDURE PrintGapLine(Symbol : CHAR, LabelWidth : INTEGER)
VAR i : INTEGER;
BEGIN
   Write(Symbol);
   FOR i := 1 TO LabelWidth - 2 DO
      Write(' ');
   WriteLn(Symbol);
END;
```

Mark as follows:
- output a symbol at either end
- output (`LabelWidth - 2`) spaces
- followed by newline

(c) (i) Example Pascal: **[7]**

```
PROCEDURE PrintNameLine(Symbol : CHAR, Name : STRING,
                                                LabelWidth : INTEGER)
VAR NumberOfSpaces, i : INTEGER;
   BEGIN                  // remember to allow for symbol at edges
   NumberOfSpaces := (LabelWidth – Length(Name) - 2) DIV 2
   Write(Symbol);
   FOR i := 1 TO NumberOfSpaces DO Write(' ');
   WRITE(Name);
   IF (LabelWidth - Length(Name)) MOD 2 > 0
       THEN NumberOFSpaces := NumberOFSpaces + 1;
   FOR i := 1 TO NumberOfSpaces DO Write(' ');
   WriteLn(Symbol);
END;
```

Mark as follows:
- correct procedure header & ending
- output symbol at either end of line, and name in middle
- calculate the number of spaces required
- output half number of spaces before name
- output half number of spaces after name
- check for odd number of spaces
- add extra space at front/rear

(ii) – `IF LENGTH(Name) > LabelWidth – 2` **[2]**

1 mark for suggesting length of name too long
1 mark for giving exact amount of excessive length

(iii) – ask user to input a shorter name // validate length of name **[2]**
– loop around INPUT `Name` until `LENGTH(Name) <= LabelWidth – 2`

(d) – reusing modules **[2]**
– easier to amend/maintain

(e) – constant declaration **Max [3]**
– meaningful identifiers/variable names
– modules // procedure calls
– use of parameters
– indentation
– capitalised variable names/identifiers // upper case key words

2 (a) Example Pascal: **[2]**

```
VAR Interval : ARRAY [1..4] OF INTEGER;
Interval[1] := 4;
Interval[2] := 2;
Interval[3] := 1;
Interval[4] := 3;
```

Mark as follows:
– declare array
– initialising

(b) Mark the reason (data must match the reason) **Max [4]**

– travelling 1 interval
– travelling 2 intervals
– travelling 3 intervals
– travelling in opposite direction should give same result
– using stations in the middle

(c) Example Pascal: **[10]**

```
PROCEDURE TicketCalculator;
VAR Origin, Destination, Temp, Distance, i : INTEGER;
VAR TicketPrice : Currency;
   BEGIN
      ReadLn(Origin);
      ReadLn(Destination);
      Distance := 0;
      IF Origin > Destination
         THEN
            BEGIN
               Temp := Origin;
               Origin := Destination;
               Destination := Temp
            END;
      FOR i := Origin TO Destination - 1 DO
         Distance := Distance + Interval[i];
      TicketPrice := Distance * 0.25;
      WriteLn(TicketPrice);
   END;
```

Mark as follows:
– declare local variables
– procedure heading and ending
– input origin and destination
– initialise distance
– swap origin and destination
– use of a temporary variable for swapping
– loop correct number of times (accept `REPEAT` or `WHILE` loops)
– add correct interval* to distance
– calculate ticket price
– output ticket price

Note: *Need to see answer to part (a)

3 (a) **[11]**

```
PROCEDURE TakeBooking
    DECLARE NumberOfCustomers, TableNumber : INTEGER          (1)
    DECLARE Found : BOOLEAN                                   (1)
    INPUT NumberOfCustomers
    // initialise search for a suitable table
    Found ← FALSE
    TableNumber = 0                                           (1)
    REPEAT        // find a table with enough seats
       TableNumber ← TableNumber + 1
       IF TableSeats[TableNumber] >= NumberOfCustomers        (1+1)
             AND Booked[TableNumber] = FALSE                  (1)
          THEN
          Found ← TRUE
       ENDIF
    UNTIL Found = TRUE OR TableNumber = 12                    (1+1)
    IF Found = FALSE
       THEN    // no tables left with enough seats
          OUTPUT "Sorry no tables with enough seats"          (1)
       ELSE...// make the booking
          Booked[TableNumber] ← TRUE                          (1)
          GroupSize[TableNumber] ← NumberOFCustomers          (1)
       OUTPUT "Table number booked: ", TableNumber
    ENDIF
ENDPROCEDURE
```

(b) **[4]**

```
PROCEDURE CancelBooking
    DECLARE TableNumber : INTEGER                             (1)
    INPUT TableNumber
    IF Booked[TableNumber] = FALSE                            (1)
       THEN
          OUTPUT "Error – this table is not booked"
       ELSE     // cancel booking
          Booked[TableNumber] ← FALSE                         (1)
          GroupSize[TableNumber] ← 0                          (1)
          OUTPUT "Booking cancelled"
    ENDIF
ENDPROCEDURE
```

(c) **[4]**

```
PROCEDURE AvailableTablesReport
   DECLARE TableNumber : INTEGER
   FOR TableNumber ← 1 TO 12
      IF Booked[TableNumber] = FALSE
         THEN
            OUTPUT "Table Number: ", TableNumber,
               " available seats: ", TableSeats[TableNumber]
      ENDIF
   ENDFOR
ENDPROCEDURE
```

Mark as follows:
- loop to access every table
- check if table not booked
- output table number
- output seats available at this table

(d) (i) Example Pascal: **[5]**

```
TYPE BookingType = RECORD
   TableSeats           : INTEGER;
   Booked               : BOOLEAN;
   GroupSize            : INTEGER;
   CustomerName         : STRING[20];
   CustomerTelNumber    : STRING[15];
   AmountDepositPaid    : CURRENCY;
END;
```

Mark as follows:
- record header & ending
- TableSeats, GroupSize correctly declared
- Booked correctly declared
- CustomerName, CustomerTelNumber correctly declared
- AmountDepositPaid correctly declared

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

```
VAR TableBookings : ARRAY[1..12] OF BookingType
```

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

(e) Example Pascal: **Max [6]**

```
PROCEDURE SaveToFile;                                    (1)
BEGIN
   VAR BookingFile : FILE OF BookingType;                (1)
   VAR i : INTEGER;                                      (1)
   ASSIGNFILE (BookingFile, 'TableBookings.DAT');        (1)
   REWRITE (BookingFile);                                (1)
   FOR i := 1 TO 12 DO                                   (1)
      WRITE(BookingFile, TableBookings[i]);              (1)
   CLOSEFILE(BookingFile);
END;
```

Mark as follows:
– Procedure heading and ending
– Declare local variable
– Assign file name
– Open file for writing
– Close file
– Loop 12 times
– Write record to file