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.
<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Marks</th>
</tr>
</thead>
</table>
| **1 (a) (i)** | TYPE LinkedList

(DECLARE) Surname : STRING
(DECLARE) Ptr : INTEGER
ENDTYPE

Accept:
LinkedIn : RECORD
Surname : STRING
Ptr : INTEGER
ENDRECORD

Accept:
TYPE LinkedList = RECORD
Surname : STRING
Ptr : INTEGER
ENDTYPE / ENDRECORD

Accept:
STRUCTURE LinkedList

(DECLARE) Surname : STRING
(DECLARE) Ptr : INTEGER
ENDSTRUCTURE

Accept AS / OF instead of :

(ii) (DECLARE) **SurnameList[1:5000]** : LinkedList

Accept AS / OF instead of :
Accept () instead of []
Accept without lower bound
Index separator can be , : ...

(b) (i) Wu
Accept with quotes

(ii) 6

(c) (i) IsFound + relevant description
BOOLEAN

© Cambridge International Examinations 2016
## Question (ii)

Accept () instead of []

```plaintext
01 Current ← StartPtr
02 IF Current = 0
03 THEN
04 OUTPUT "Empty List" (or similar message) (accept without quotes) Reject “Error”
05 ELSE
06 IsFound ← FALSE
07 INPUT ThisSurname
08 REPEAT
09 IF SurnameList[Current].Surname = ThisSurname
10 THEN
11 IsFound ← TRUE
12 OUTPUT "Surname found at position ", Current
13 ELSE
14 // move to the next list item
15 Current ← SurnameList[Current].Ptr
16 ENDIF
17 UNTIL IsFound = TRUE OR Current = 0
18 IF IsFound = FALSE
19 THEN
20 OUTPUT "Not Found"
21 ENDIF
22 ENDIF
```

Accept = for assignment

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>(ii)</td>
<td>Accept () instead of []</td>
<td>6</td>
</tr>
<tr>
<td>(i)</td>
<td>A procedure which is defined in terms of itself // A procedure which makes a call to itself // A procedure that calls itself</td>
<td>1</td>
</tr>
<tr>
<td>(ii)</td>
<td>08 // 8</td>
<td>1</td>
</tr>
<tr>
<td>Question</td>
<td>Answer</td>
<td>Marks</td>
</tr>
<tr>
<td>----------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td><strong>(b) (i)</strong></td>
<td>![Table and Diagram]</td>
<td><strong>4</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Index</strong></td>
<td><strong>Item</strong></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3 5 8 9 13 16 27 0 0 0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4 13</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5 16</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>6 27</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>7 0</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

Note: Final mark only if no additional entries in table
Accept last row to show all final values

| (ii) | Any one from: Deletes/removes parameter value/ Item (from the array **MyList**) // Deletes the first entry (in **MyList**) that equals or is bigger than **Item** Overwrites **Item** by moving subsequent items up/down/ across/ left R right | 1 |

© Cambridge International Examinations 2016
Mark as follows:
Label F_TRAILER 1
Label TRANS 1
Customer box (Accept label Customer) 1
Hire box (Accept label Hire) 1
Customer fields: Customer Name, CustomerID/IDnumber 1
Hire fields: Car Reg 1
Hire fields: Hire start date, Number of days hired 1
accept level 5 fields in any order
Ignore parent
### Question (b)

<table>
<thead>
<tr>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAR_REPORT</td>
</tr>
<tr>
<td>*</td>
</tr>
<tr>
<td>CAR</td>
</tr>
<tr>
<td>No hires</td>
</tr>
<tr>
<td>One or more hires</td>
</tr>
<tr>
<td>NO_HIRE</td>
</tr>
<tr>
<td>CAR_HIRE</td>
</tr>
<tr>
<td>Car registration</td>
</tr>
<tr>
<td>HIRE_LIST</td>
</tr>
<tr>
<td>Car total</td>
</tr>
<tr>
<td>*</td>
</tr>
<tr>
<td>HIRE</td>
</tr>
<tr>
<td>Start date</td>
</tr>
<tr>
<td>Number of days</td>
</tr>
</tbody>
</table>

**Mark as follows:**

- Selection symbol x 2 (Car-hire / No car-hire) 1
- Labelling for CAR_HIRE/NO_HIRE (accept similar labels*) 1
- Labelling for Car registration and Car total / Total hires 1
- Iteration symbol for HIRE (accept in HIRE_LIST as a BOD) 1
- Labelling for start date and number of days (as per diagram) 1

* For CAR_HIRE label:
Accept: Hires / hired / Car data / hire data / hire record / one or more hires

---

© Cambridge International Examinations 2016
### Question 4

(a) (i) 

- `a03, b07, a23`  
  accept in any order, must be lower case  
  1 mark

(ii) 

- The car must pass (both) brake test and tyres test  
  1 mark

(b) 

- `retestAllowed(ThisCar)`  
  1 mark
- `If (testBrakes(ThisCar, pass) and testTyres(ThisCar, fail)) or (testBrakes(ThisCar, fail) and testTyres(ThisCar, pass))`  
  1 mark

(one mark per bold underlined all correct) 

accept another variable instead of `ThisCar`, but must be same throughout.  

(c) (i) 

- `a07`  
  `[p03]`  
  must be [], must be lower case, but don’t penalise twice, so follow through from part(b)  
  2 marks

(ii) 

- `[p05,m04]`  
  1 mark

(iii) 

- `[ ]`  
  1 mark

(d) 

- `[ ]`  
  1 mark

### Question 5

(a) (i) 

<table>
<thead>
<tr>
<th>Mark</th>
<th>Description</th>
<th>Expected result (Grade)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>FAIL/PASS/MERIT/DISTINCTION</td>
<td></td>
</tr>
<tr>
<td>Abnormal</td>
<td>Error</td>
<td></td>
</tr>
<tr>
<td>Extreme/Boundary</td>
<td>FAIL/PASS/MERIT/DISTINCTION</td>
<td></td>
</tr>
</tbody>
</table>

3 × (mark + matching grade)  

for abnormal data accept negative values, non-integer values, Expected Result: Error  

0 and marks above 100 are still acceptable values  

Do not accept FAIL in expected result column for Abnormal data  

(ii) 

- (The programmer is) concerned only with the input (i.e. the mark) to the function and monitoring the expected output (i.e. the grade)  
  // can compare expected result and actual result  
  1 mark

(b) 

- Exception:  
  1. situation causing a crash / run-time error / fatal error  
  1 mark

Exception handling:  

- 2. code which is called when a run-time error occurs  
  1 mark

- 3. … to avoid the program terminating/crashing  
  1 mark
### Question (c)

<table>
<thead>
<tr>
<th>Answer</th>
<th>Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Open a non-existent file</td>
<td>Max 3</td>
</tr>
<tr>
<td>2. Directory path does not exist</td>
<td></td>
</tr>
<tr>
<td>3. Attempt to read past the end of the file // attempt to read an empty file</td>
<td></td>
</tr>
<tr>
<td>4. Array subscript is out of range</td>
<td></td>
</tr>
<tr>
<td>5. Non-integer value / corrupt data read</td>
<td></td>
</tr>
<tr>
<td>6. File already open in a different mode // wrong file permissions</td>
<td></td>
</tr>
</tbody>
</table>

### Question (d) (i)

<table>
<thead>
<tr>
<th>Answer</th>
<th>Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>09. // 9</td>
<td>1</td>
</tr>
</tbody>
</table>

### Question (d) (ii)

<table>
<thead>
<tr>
<th>Answer</th>
<th>Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Line 11 catches exceptions (only) between lines 05 and 10</td>
<td>1</td>
</tr>
<tr>
<td>2. Line 11 stops the program from crashing</td>
<td>1</td>
</tr>
<tr>
<td>3. Different exception types recognised</td>
<td>1</td>
</tr>
<tr>
<td>4. Each exception type has an appropriate message output</td>
<td>1</td>
</tr>
<tr>
<td>5. The program language has an (object) type EXCEPTION</td>
<td>1</td>
</tr>
<tr>
<td>6. ThisException is the instance of EXCEPTION which has been raised</td>
<td>1</td>
</tr>
<tr>
<td>7. EXCEPTION objects have a ‘Message’ property // the message property for ThisException is “Arithmetic operation resulted in an overflow”</td>
<td>1</td>
</tr>
</tbody>
</table>

### Question 6 (a)

Max 3 marks if extra states/transitions added.

![State Transition Diagram]

Winning move

WHITE WINS

WINNING MOVE

NO MOVE POSSIBLE

STATEMATE

BLACK WINS

Winning move

WHITE'S TURN

BLACK'S TURN

BLACK moves

WHITE moves

Max 3 marks if extra states/transitions added.
### Question (b) (i)
Mark as follows:
1. Declaration for array (character or string data type)
2. FOR loop for x going from 1 to 8, generating column index used in array
3. FOR loop for y going from 1–2, 3–6, 7–8 (Accept all squares being set to 'E' and then overwritten with 'B', 'W' respectively)
4. Setting squares to 'B', 'E', 'W' (must be in quotes, accept single or double)

**Marks:** 4

### Question (b) (ii)
Mark as follows:
1. Procedure heading and declaration of 2 local variables
2. Establishing the stopper colour – opposite to the mover
3. Test for piece in column 1 (x>1) // column 8 (x<8)
4. Test for 'E'
5. Correct method for moving left // for moving right
6. until edge of board reached
7. until other colour (stopper colour) encountered
8. until own colour encountered (PieceColour)
9. Correct output for cell indexes (accept for moving in 1 direction only)
10. including the ‘REMOVE’ message

**Note:** must use given parameter identifiers: PieceColour, xCurrent, yCurrent

**Max:** 5

### Question (c) (i)
Classes could be designed for:
- the board
- a piece

Containment (Board contains Pieces)
The pieces are instances/objects (of the Piece class)

**Max:** 2
<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>(ii)</td>
<td>Accept any reasonable answer, for example:</td>
<td>Max 2</td>
</tr>
</tbody>
</table>

**BOARD class:**

Properties:
- Number of squares / size / dimensions
- Current state of all squares

Methods: –
- Set the starting board
- Capture the finishing state of the board
- Display the state of the board after each move

**PIECE class:**

Properties:
- Starting x position
- Starting y position
- Current x position
- Current y position
- Colour
- State / Removed / Active

Methods:
- Move piece
- Remove piece

Mark as follows:
two correct responses are worth 1 mark

Accept other classes: Game, Player
Programming code

6 (b) (i)

**VB.NET**
Dim Board(8, 8) As Char
Dim Row, Column As Integer
For Row = 1 To 2
    For Column = 1 To 8
        Board(Row, Column) = "B"
    Next
Next
For Row = 3 To 6
    For Column = 1 To 8
        Board(Row, Column) = "E"
    Next
Next
For Row = 7 To 8
    For Column = 1 To 8
        Board(Row, Column) = "W"
    Next
Next

**PASCAL**
var Row, Column : integer;
Board : array[1..8, 1..8] of char;
begin
    for Row := 1 to 2 do
        for Column := 1 to 8 do
            Board[Row, Column] := 'B';
    for Row := 3 to 6 do
        for Column := 1 to 8 do
            Board[Row, Column] := 'E';
    for Row := 7 to 8 do
        for Column := 1 to 8 do
            Board[Row, Column] := 'W';
end.
PYTHON
Board = [["" for j in range(9)] for i in range(9)]
for Row in range(1, 3):
    for Column in range(1, 9):
        Board[Row][Column] = "B"
for Row in range(3, 7):
    for Column in range(1, 9):
        Board[Row][Column] = "E"
for Row in range(7, 9):
    for Column in range(1, 9):
        Board[Row][Column] = "W"

Alternative declarations of Board array:

Board = ["" for i in range(9)]
for i in range(9):
    Board.append("")

Instead of initialising with empty string, could initialise with ‘E’. this would then only require ‘B’ and ‘W’ loops later.

For example:

Board = [['E'] * 9 for i in range(9)]  // Board =[['E']*9]*9
for Row in range(1, 3):
    for Column in range(1, 9):
        Board[Row][Column] = "B"
for Row in range(7, 9):
    for Column in range(1, 9):
        Board[Row][Column] = "W"

Board = []
for i in range(9):
    Board.append(['E']*9)
6 (b) (ii)

**VB.NET**

```vbnet
Sub ValidMoves(ByVal PieceColour As Char, ByVal xCurrent As Integer, ByVal yCurrent As Integer)
    Dim i As Integer
    Dim StopperColour As Char
    Dim NoFurther As Boolean
    If PieceColour = "B" Then
        StopperColour = "W"
    Else
        StopperColour = "B"
    End If
    Console.WriteLine("Possible moves are : ")
    If xCurrent <> 1 Then
        Console.WriteLine("Moving LEFT . . .")
        i = xCurrent – 1
        NoFurther = False
        Do
            If Board(i, yCurrent) = "E" Then
                Console.WriteLine(i & " " & yCurrent)
            End If
            If Board(i, yCurrent) = StopperColour Then
                Console.WriteLine(i & " " & yCurrent & " REMOVE PIECE")
                NoFurther = True
            End If
            i = i – 1
        Loop Until i = 0 Or NoFurther = True
    End If
    If xCurrent <> 8 Then
        Console.WriteLine("Moving RIGHT . . .")
        i = xCurrent + 1
        NoFurther = False
        Do
            If Board(i, yCurrent) = "E" Then
                Console.WriteLine(i & " " & yCurrent)
            End If
            If Board(i, yCurrent) = StopperColour Then
                Console.WriteLine(i & " " & yCurrent & " REMOVE PIECE")
                NoFurther = True
            End If
            i = i + 1
        Loop Until i = 9 Or NoFurther = True
    End If
End Sub
```
PASCAL
procedure ValidMoves(PieceColour : char; xCurrent, yCurrent : integer);
var StopperColour : char;
i : integer;
NoFurther : boolean;
begin
  if (PieceColour = 'B') then
    StopperColour := 'W'
  else
    StopperColour := 'B';
  writeln('Possible moves are : ');
  if (xCurrent <> 1) then
    begin
      writeln('Moving LEFT . . . ');
      i := xCurrent – 1;
      NoFurther := false;
      repeat
        if (Board[i, yCurrent] = 'E') then
          writeln(intToStr(i) + ' ' + intToStr(yCurrent));
        if (Board[i, yCurrent] = StopperColour) then
          begin
            writeln(intToStr(i) + ' ' + intToStr(yCurrent) + ' REMOVE PIECE');
            NoFurther := true;
          end;
        i := i – 1;
      until ((i = 0) or (NoFurther = true));
    end;
  if (xCurrent <> 8) then
    begin
      writeln('Moving RIGHT . . . ');
      i := xCurrent + 1;
      NoFurther := false;
      repeat
        if (Board[i, yCurrent] = 'E') then
          writeln(intToStr(i) + ' ' + intToStr(yCurrent));
        if (Board[i, yCurrent] = StopperColour) then
          begin
            writeln(intToStr(i) + ' ' + intToStr(yCurrent) + ' REMOVE PIECE');
            NoFurther := true;
          end;
        i := i + 1;
      until ((i = 9) or (NoFurther = true));
    end;
end;
def ValidMoves(PieceColour, xCurrent, yCurrent) :
    if PieceColour == "B" :
        StopperColour = "W"
    else :
        StopperColour = "B"
    print("Possible moves are : ")
    if xCurrent != 1 :
        print("Moving LEFT . . .")
        i = xCurrent – 1
        NoFurther = False
        while i > 0 and NoFurther == False :
            if Board[i][yCurrent] == "E" :
                print(str(i) + " " + str(yCurrent))
            if Board[i][yCurrent] == StopperColour :
                print(str(i) + " " + str(yCurrent) + " REMOVE PIECE")
                NoFurther = True
            i = i – 1
    if xCurrent != 8 :
        print("Moving RIGHT . . .")
        i = xCurrent + 1
        NoFurther = False
        while i < 9 and NoFurther == False :
            if Board[i][yCurrent] == "E" :
                print(str(i) + " " + str(yCurrent))
            if Board[i][yCurrent] == StopperColour :
                print(str(i) + " " + str(yCurrent) + " REMOVE PIECE")
                NoFurther = True
            i = i + 1