Pearson Edexcel

Mark Scheme

Specimen Set 1

Pearson Edexcel GCSE In Computer Science (1CP2)
Paper 01: Principles of Computer Science

Edexcel and BTEC Qualifications

Edexcel and BTEC qualifications are awarded by Pearson, the UK's largest awarding body. We provide a wide range of qualifications including academic, vocational, occupational and specific programmes for employers. For further information visit our qualifications websites at www.edexcel.com or www.btec.co.uk. Alternatively, you can get in touch with us using the details on our contact us page at www.edexcel.com/contactus.

Pearson: helping people progress, everywhere

Pearson aspires to be the world's leading learning company. Our aim is to help everyone progress in their lives through education. We believe in every kind of learning, for all kinds of people, wherever they are in the world. We've been involved in education for over 150 years, and by working across 70 countries, in 100 languages, we have built an international reputation for our commitment to high standards and raising achievement through innovation in education. Find out more about how we can help you and your students at: www.pearson.com/uk

Paper 2 Mark Scheme

Question number	Answer	Additional guidance	Mark
1	Award marks as shown. • pLocation (1) • = (1) • boxes / theString (1) • print / int / input (1) • isValid (1) • MAX_BOXES (1) • valid / theString (1)	• The subprogram 'showBox' is a procedure, as defined in 6.6.2, because it does not return a result. The subprogram 'isValid' returns a value, so meets the definition of a function as defined in 6.6.2. • Do not penalise spelling or capitalisation, as long as meaning is clear	**(7)**

```
# ---------------------------------------------------------------------
# Constants
# ---------------------------------------------------------------------
MAX_BOXES = 5

# ---------------------------------------------------------------------
# Global variables
# ---------------------------------------------------------------------
boxes = ["circle", "square", "triangle", "hexagon", "octagon"]

# ---------------------------------------------------------------------
# Subprograms
# ---------------------------------------------------------------------
def isValid (pLocation):
    valid = False

    if (pLocation - 1 >= 0) and (pLocation - 1 < MAX_BOXES):
        valid = True
    return (valid)

def showBox (pLocation):
    theString = ""

    theString = "The box is " + boxes[pLocation - 1] + " shaped."
    print (theString)

```

```
27  # ---------------------------------------------------------------------
28  # Main program
29  # ---------------------------------------------------------------------
30  choice = int (input ("Welcome to the game.  Choose a number (1 to 5)"))
31  if (isValid (choice)):
32      showBox (choice)
33  else:
34      print ("Invalid choice")
35
36  # ---------------------------------------------------------------------
37  # =====> Write your answers here in the right hand column
38  # Left                                          # Right
39  # ---------------------------------------------------------------------
40  # Give the name of a parameter                  # pLocation
41  # Give the symbol used to show assignment       # =
42  # Give the name of a structured data type implemented as a list
43                                                  # boxes
44  # Give the name of a built-in subprogram        # print
45  # Give the name of a user-devised function      # isValid
46  # Give the name of a constant                   # MAX_BOXES
47  # Give the name of a local variable             # valid / theString
```

Question number	Answer	Additional guidance	Mark
2	Award marks as shown. • Fix the syntax error by changing [`inport turtle`] to [**`import`** `turtle`] (1) • Fix the syntax error by adding a bracket to change [`tomasz.pencolor "red")`] to [`tomasz.pencolor ("red")`] (1) • Fix the 'NameError' by changing [`tomasz == turtle.Turtle ()`] to [`tomasz = turtle.Turtle ()`] (1) • Calculate 'angle' by [`angle = 360 / numSides`] `angle =` (1) 360 / numSides (1) • Correct logic error [`tomasz.left (70)`] to [`tomasz.left (angle)`] (1) • Changing the variable name 'n' to a more meaningful name (1) such as 'numSides' throughout the code • Use of int(input("<prompt>")) to take input from the user (1) • Storing value in variable 'length' (1) • Comment indicating that string to integer conversion is required / input() returns a string and must be converted to integer / input is string and sides must be a number (1)		**(10)**

```
# ------------------------------------------------------------------
# Import libraries
# ------------------------------------------------------------------
# ===> Fix the syntax error
import turtle

# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
# ===> Fix the NameError
tomasz = turtle.Turtle ()                # Create a turtle
angle = 0
length = 20
key = ""

# ===> Change the name of the variable 'n' to be more meaningful
numSides = 0

```

```
# ------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------
# ===> Fix the syntax error
tomasz.pencolor ("red")
tomasz.pensize (5)

# ===> Add a comment to describe why int() is needed
# input() always returns a string so needs to be converted to integer
numSides = int (input ("Enter number of sides, at least 3: "))

# ===> Calculate angle by dividing 360 by the number of sides and
#      store the value in the variable angle
angle = 360 / numSides

# ===> Ask the user for the length of each side and store the value
#      in the variable length
length = int (input ("Enter the length of each side: "))

for sides in range (numSides):
  tomasz.forward (length)
  # ===> Fix the logic error
  tomasz.left (angle)

key = input ("Press any key")
```

Question number	Answer	Additional guidance	Mark

3	Award marks as shown.
	
	• Order of arithmetic required in creation of constants:
	  ○ RETIRED creation must come after creation of STUDENT (1)
	  ○ CONCESSION creation must come after RETIRED creation (1)
	• Choice of appropriate instruction statement:
	  ○ layout = "Your fare is {:4.2f}" (1)
	  ○ while (not validChoice) (1)
	  ○ if ((userChoice != "Q") and … (1)
	  ○ validChoice = True (1)
	• Calculated fares match category of test:
	  ○ Concession fare must go with 'else' (1)
	  ○ All four fares match tests (1)
	• Ordering of lines in main program with correct indentation, one each for a maximum of 7
	  ○ while (choice != "Q"): (1)
	  ○ showMenu() (1)
	  ○ choice = getUserChoice() (1)
	  ○ if (choice != "Q"): (1)
	  ○ fare = calcFare (choice) (1)
	  ○ print (layout.format(fare)) (1)
	  ○ print ("Goodbye") (1)
			(15)

```
# ---------------------------------------------------------------------
# Import libraries
# ---------------------------------------------------------------------

# ---------------------------------------------------------------------
# Constants
# ---------------------------------------------------------------------
# ===> Order the jumbled lines
FULL_FARE = 1.50
STUDENT = 0.25
CHILD = 0.0
RETIRED = 2 * STUDENT
CONCESSION = RETIRED * 2 / 10 * 4
# Finished jumbled lines

# ---------------------------------------------------------------------
# Global variables
# ---------------------------------------------------------------------
# ===> Choose one line
layout = "Your fare is {:4.2f}"
# layout = "Your fare is {:4.4f}"

choice = ""
fare = 0.0

```

```
26  # -----------------------------------------------------------------------
27  # Subprograms
28  # -----------------------------------------------------------------------
29  def showMenu ():
30      print ("----- Fares -----")
31      print ("A - Full fare")
32      print ("B - Student fare (8 - 16 years)")
33      print ("C - Child fare (0 - 7 years)")
34      print ("D - Retired (65+ years)")
35      print ("E - Other concession")
36      print ("Q - Quit")
37
```

```
38  def getUserChoice ():
39      userChoice = ""
40      validChoice = False
41
42      # ===> Choose one line
43      # while (validChoice):
44      while (not validChoice):
45          userChoice = input ("Please choose a fare: ")
46          userChoice = userChoice.upper ()
47          #===> Choose one long statement of 3 lines
48          if ((userChoice != "Q") and (userChoice != "A") and
49                  (userChoice != "B") and (userChoice != "C") and
50                  (userChoice != "D") and (userChoice != "E")):
51          #if ((userChoice == "Q") or (userChoice == "A") or
52          #        (userChoice == "B") or (userChoice == "C") or
53          #        (userChoice == "D") or (userChoice == "E")):
54              print ("Invalid entry")
55          else:
56              # Choose one line
57              # validChoice = False
58              validChoice = True
59
60      return (userChoice)
61
```

```
62  def calcFare (pCategory):
63      theFare = 0.0
64
65      # ===> Order and indent the jumbled lines
66      if (pCategory == "A"):
67          theFare = FULL_FARE
68      elif (pCategory == "B"):
69          theFare = STUDENT * FULL_FARE
70      elif (pCategory == "C"):
71          theFare = CHILD * FULL_FARE
72      elif (pCategory == "D"):
73          theFare = RETIRED * FULL_FARE
74      else:
75          theFare = CONCESSION * FULL_FARE
76      # Finished jumbled lines
77
78      return (theFare)
79
```

```
# ----------------------------------------------------------------------
# Main program
# ----------------------------------------------------------------------

# ===> Here is a group of jumbled lines
# ===> Order the lines by moving them into their correct location
# ===> The number of lines to place is shown in brackets after the arrows
# ===> Indentation levels are shown for you with commented arrows

# ===> (1 line)
while (choice != "Q"):
    # ===> (3 lines)
    showMenu ()
    choice = getUserChoice ()
    if (choice != "Q"):
        # ===> (2 lines)
        fare = calcFare (choice)
        print (layout.format (fare))
# ===> (1 line)
print ("Goodbye")
# ----- End of the file -----
```

Question number	Answer	Additional guidance	Mark
4	Award marks as shown. • Add new value to total (1) • 'if' statement and test for maximum (1) • Set new maximum value if test is True (1) • 'if' statement and test for minimum (1) • Set new minimum value if test is True (1) • Calculate mean (1) • Use of 'while' loop for 'count > 0' (1) • Maximum set to a float value <= 0.0 (1) • Minimum set to a float value >= 99999.0 (1) • Take number of values from keyboard and convert to integer (1) • Take individual values from keyboard and convert to float (1) • Display all three (minimum, maximum, mean) to user (1) Levels-based mark scheme to a maximum of 3, from: • Functionality (3)	• No validation of input values required • Setting maximum and minimum with the first value entered is acceptable for bullet 8 and bullet 9 Considerations for levels-based mark scheme: • [6.5.2] Accurate use of relational operators (>, <) in appropriate selection or sequence statements • [6.5.1] Accurate use of arithmetic operators (+, -, ÷) in appropriate selection or sequence statements • [6.1.3] Program should produce the correct output for any predictable input	**(15)**

Functionality (levels-based mark scheme)

0	1	2	3	Max.
No rewardable material	**Functionality (when the code is run)** • The component parts of the program are incorrect or incomplete, providing a program of limited functionality that meets some of the given requirements. • Program outputs are of limited accuracy and/or provide limited information. • Program responds predictably to some of the anticipated input. • Solution is not robust and may crash on anticipated or provided input.	**Functionality (when the code is run)** • The component parts of the program are complete, providing a functional program that meets most of the stated requirements. • Program outputs are mostly accurate and informative. • Program responds predictably to most of the anticipated input. • Solution may not be robust within the constraints of the problem.	**Functionality (when the code is run)** • The component parts of the program are complete, providing a functional program that fully meets the given requirements. • Program outputs are accurate, informative, and suitable for the user. • Program responds predictably to anticipated input. • Solution is robust within the constraints of the problem.	**3**

```
# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
count = 0
numValues = 0
inValue = 0.0
total = 0.0
theMax = 0.0
theMin = 99999.0
theMean = 0.0

# ------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------

numValues = int (input ("How many values do you want to enter? "))
count = numValues
while (count > 0):
    inValue = float (input ("Enter a value: "))
    total = total + inValue
    if (inValue > theMax):
        theMax = inValue
    if (inValue < theMin):
        theMin = inValue
    count = count - 1
theMean = total / numValues
print (theMin, theMax, theMean)
```

Question number	Answer	Additional guidance	Mark
5	Award marks as shown. • Use of 'for' or 'while' loop to process every tree (1) • Use of string concatenation to build output strings (1) • Output string contains 3 items in order (name, count, score) (1) • Output string fields separated by commas (1) • Line feed added to end of output string (1) • Use of same or equivalent index variable for tree names, count, and score. (1) • Use of white space and comments aids readability (1) Levels-based mark scheme to a maximum of 6, from: • Solution design (3) • Functionality (3)	• Fixing error with odd numbers can be done in several different ways (see examples) • Award any accurate tests for validation range Considerations for levels-based mark scheme: • [6.4.2] Each tree's information is on a separate line in the output file • [6.2.2] Use of 'for' loop in preference to a 'while' loop for iteration across an entire data structure • [6.4.2] No comma on last field; no "\n" on last record; no additional spaces after commas; i.e. meets requirement of text file • [6.3.1] Same index variable used for all data structures, rather than three different ones. Minimum of one index value is required. • [6.4.2] File opened only for writing • [6.4.2] Close file to match open	**(13)**

Solution design (levels-based mark scheme)

0	1	2	3	Max.
No rewardable material	• There has been little attempt to decompose the problem. • Some of the component parts of the problem can be seen in the solution, although this will not be complete. • Some parts of the logic are clear and appropriate to the problem. • The use of variables and data structures, appropriate to the problem, is limited. • The choice of programming constructs, appropriate to the problem, is limited.	• There has been some attempt to decompose the problem. • Most of the component parts of the problem can be seen in the solution. • Most parts of the logic are clear and appropriate to the problem. • The use of variables and data structures is mostly appropriate. • The choice of programming constructs is mostly appropriate to the problem.	• The problem has been decomposed clearly into component parts. • The component parts of the problem can be seen clearly in the solution. • The logic is clear and appropriate to the problem. • The choice of variables and data structures is appropriate to the problem. • The choice of programming constructs is accurate and appropriate to the problem.	**3**

Functionality (levels-based mark scheme)

0	1	2	3	Max.
No rewardable material	**Functionality (when the code is run)** • The component parts of the program are incorrect or incomplete, providing a program of limited functionality that meets some of the given requirements. • Program outputs are of limited accuracy and/or provide limited information. • Program responds predictably to some of the anticipated input. • Solution is not robust and may crash on anticipated or provided input.	**Functionality (when the code is run)** • The component parts of the program are complete, providing a functional program that meets most of the stated requirements. • Program outputs are mostly accurate and informative. • Program responds predictably to most of the anticipated input. • Solution may not be robust within the constraints of the problem.	**Functionality (when the code is run)** • The component parts of the program are complete, providing a functional program that fully meets the given requirements. • Program outputs are accurate, informative, and suitable for the user. • Program responds predictably to anticipated input. • Solution is robust within the constraints of the problem.	**3**

Example 1:

```
# ---------------------------------------------------------------------
# Constants
# ---------------------------------------------------------------------
OUTFILE = "Trees.txt"
LF = "\n"

# ---------------------------------------------------------------------
# Global variables
# ---------------------------------------------------------------------
treeNames = ["Maple", "Ash", "Sycamore", "Birch", "Hazel", "Willow", "Oak"]
treeCounts = [1357, 8421, 9287, 1043, 3743, 2948, 10826]
treeScore = [0.15, 0.18, 0.73, 0.38, 0.92, 0.24, 0.78]
index = 0

```

```
# ---------------------------------------------------------------------
# Main program
# ---------------------------------------------------------------------
# ===> Open the output file for writing
theFile = open (OUTFILE, "w")                # Open for writing

# ===> Use iteration to process each item in treeNames
for tree in treeNames:

    # ===> Create a comma separated value output line
    outString = ""
    outString = outString + tree + ","      # Separate with commas
    outString = outString + str (treeCounts[index]) + ","
    outString = outString + str (treeScore[index])   # No comma

    index = index + 1                           # Point to next one

    if (index < len (treeNames)):               # Not for last line
        outString = outString + LF              # Add line feed

    # ===> Write the line to the file
    theFile.write (outString)                # Write the line

# ===> Close the output file
theFile.close ()                             # Close the file
```

Example 2:

```
# ------------------------------------------------------------------
# Constants
# ------------------------------------------------------------------
OUTFILE = "Trees.txt"

# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
treeNames = ["Maple", "Ash", "Sycamore", "Birch", "Hazel", "Willow", "Oak"]
treeCounts = [1357, 8421, 9287, 1043, 3743, 2948, 10826]
treeScore = [0.15, 0.18, 0.73, 0.38, 0.92, 0.24, 0.78]
# ===> Write your code here
index = 0
nextRecord = "" # Will be used to store a line to be written to the file
# ------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------
# ===> Open the output file for writing

treeFile = open (OUTFILE, "w")

```

```
22  # ===> Use iteration to process each item in treeNames
23  for index in range (0, len (treeNames)): # Iterates through the arraytreeNames and picks up
24                                           # corresponding values stored in same index position
25                                           # in the other 2 parallel arrays
26
27      # ===> Create a comma separated value output line
28      # Commas used to separate the values, and a new line code added at the end
29      # so that each record is written to a separate line of the file
30      nextRecord = treeNames[index] + "," + \
31                   str (treeCounts[index]) + "," + \
32                   str (treeScore[index])
33
34      # Add linefeed for all lines except last in the file,
35      # otherwise, there is a blank line at the end of the file
36      if (index != len (treeNames) - 1):
37          nextRecord = nextRecord + "\n"
38
39      # ===> Write the line to the file
40      treeFile.write (nextRecord)
41
42  # ===> Close the output file
43  treeFile.close ()
```

Question number	Answer	Additional guidance	Mark
6	Award marks as shown. Points-based mark scheme: **Inputs** • Accepts four inputs, disregard conversion (1) • Validation with range check using relational operators 1 <= month <= 12 (1) **Process** • Correct subprogram call (1) to generate identifier or to do linear search • Builds and appends a correctly formatted record to members list (1) • Use of indexing (1) to access fields in members list **Outputs** • [6.4.1] Display of at least one appropriate message (1) Levels-based mark scheme to a maximum of 9, from: • Solution design (3) • Good programming practices (3) • Functionality (3)	• Range checks may be constructed in different ways, e.g. 0 < month < 13. • Subprogram call must match subprogram header, i.e. order and type of arguments must match order and type of parameters expected. • Record appended to members table must follow format of table entries, i.e. fields must be in the right order. Considerations for levels-based mark scheme: • [6.1.2] Write in a high-level language • [6.2.2] Main program code is laid out in clear sections; white space is used to show different parts of the solution/functionality; variable names are meaningful; comments are provided and are helpful; • [6.2.2] Use sequencing and selection components • [6.4.1] Messages match result of range tests, result of linear search, or result of adding a new member • [6.1.6] Functions correctly for all of the cases, e.g. month too small,	**(15)**

		month too large, year too small, year too large, ID already in members, ID needs to be added • [6.5.3] Use of Boolean operators (AND, OR) to create compound tests for range validations / use 'not in range' • [6.1.1] Use decomposition and abstraction to analyse a problem (inputs, outputs, processing, initialisation, design) • [6.6.1] Decompose into subproblems • [6.2.2] Use of 'for' loop to iterate over a data structure, rather than a 'while' loop is acceptable, although the more efficient and effective solution is to use a 'while' and stop the loop if duplicate found. • [6.3.1] Conversion of input types to those required by program, e.g. two strings and two integers	

Solution design (levels-based mark scheme)

0	1	2	3	Max.
No rewardable material	• There has been little attempt to decompose the problem. • Some of the component parts of the problem can be seen in the solution, although this will not be complete. • Some parts of the logic are clear and appropriate to the problem. • The use of variables and data structures, appropriate to the problem, is limited. • The choice of programming constructs, appropriate to the problem, is limited.	• There has been some attempt to decompose the problem. • Most of the component parts of the problem can be seen in the solution. • Most parts of the logic are clear and appropriate to the problem. • The use of variables and data structures is mostly appropriate. • The choice of programming constructs is mostly appropriate to the problem.	• The problem has been decomposed clearly into component parts. • The component parts of the problem can be seen clearly in the solution. • The logic is clear and appropriate to the problem. • The choice of variables and data structures is appropriate to the problem. • The choice of programming constructs is accurate and appropriate to the problem.	**3**

Good programming practices (levels-based mark scheme)

0	1	2	3	Max.
No rewardable material	• There has been little attempt to lay out the code into identifiable sections to aid readability. • Some use of meaningful variable names. • Limited or excessive commenting. • Parts of the code are clear, with limited use of appropriate spacing and indentation.	• There has been some attempt to lay out the code to aid readability, although sections may still be mixed. • Uses mostly meaningful variable names. • Some use of appropriate commenting, although may be excessive. • Code is mostly clear, with some use of appropriate white space to aid readability.	• Layout of code is effective in separating sections, e.g. putting all variables together, putting all subprograms together as appropriate. • Meaningful variable names and subprogram interfaces are used where appropriate. • Effective commenting is used to explain logic of code blocks. • Code is clear, with good use of white space to aid readability.	**3**

Functionality (levels-based mark scheme)

0	1	2	3	Max.
No rewardable material	**Functionality (when the code is run)** • The component parts of the program are incorrect or incomplete, providing a program of limited functionality that meets some of the given requirements. • Program outputs are of limited accuracy and/or provide limited information. • Program responds predictably to some of the anticipated input. • Solution is not robust and may crash on anticipated or provided input.	**Functionality (when the code is run)** • The component parts of the program are complete, providing a functional program that meets most of the stated requirements. • Program outputs are mostly accurate and informative. • Program responds predictably to most of the anticipated input. • Solution may not be robust within the constraints of the problem.	**Functionality (when the code is run)** • The component parts of the program are complete, providing a functional program that fully meets the given requirements. • Program outputs are accurate, informative, and suitable for the user. • Program responds predictably to anticipated input. • Solution is robust within the constraints of the problem.	**3**

```python
# ---------------------------------------------------------------------
# Global variables
# ---------------------------------------------------------------------
membersTable = [[1996, "Adams", "Amanda", 1, 1996],
                [2002, "Brown", "Brenda", 6, 2000],
                [2000, "Collins", "Christopher", 3, 1999],
                [2010, "Dawson", "Denise", 12, 2001],
                [2016, "Everett", "Edward", 8, 2012]]

# ===> Write your code here
lastName = ""
firstName = ""
birthMonth = 0
birthYear = 0
idNum = 0
newRecord = []

# ---------------------------------------------------------------------
# Subprograms
# ---------------------------------------------------------------------
def isFreeKey (pKey):
    isFree = True                       # Assume it's free
    index = 0

    # ===> Write your code here
    while ((isFree) and (index < len (membersTable))):
        if (membersTable[index][0] == pKey):
            isFree = False
        index = index + 1

    return (isFree)

```

```
33  def genKey (pYear, pMonth):
34      # ===> Write your code here
35      newKey = pYear + (pYear % pMonth)
36
37      return (newKey)
38
39  # ---------------------------------------------------------------------
40  # Main program
41  # ---------------------------------------------------------------------
42  # ===> Write your code here
43
44  # Get the data from the user
45  lastName = input ("Enter your last name: ")
46  firstName = input ("Enter your first name: ")
47  birthMonth = int (input ("Enter your birth month (1 is January) "))
48  birthYear = int (input ("Enter your birth year in four digits "))
49
```

```
50      # Validate input data
51      if (birthMonth < 1) or (birthMonth > 12):
52          print ("Invalid month")
53      elif (birthYear < 1900) or (birthYear > 9999):
54          print ("Invalid year")
55      else:
56          # All input is good.  Try to make a member.
57          idNum = genKey (birthYear, birthMonth)
58          if (isFreeKey (idNum)):
59              # Make record and add to members table
60              newRecord = [idNum, lastName, firstName, birthMonth, birthYear]
61              membersTable.append (newRecord)
62              print ("The new member is: ", newRecord)
63          else:
64              # Member number already used
65              print ("ID already used ")
66
67      print ("Goodbye")
```