THIS IS A LEGACY SPECIFICATION

OCR
RECOGNISING ACHIEVEMENT

ADVANCED SUBSIDIARY GCE
COMPUTING 2507

Structured Practical Computing Tasks

Issued September 2008

Maximum Mark **120**

OPEN ON RECEIPT **JUNE 2009**

CUP/T64190

INSTRUCTIONS TO CANDIDATES

- You should attempt all tasks, working independently from other candidates.
- There are no time limitations on the tasks other than that they must be submitted by the appropriate internal deadline set by the Candidate's Centre. This deadline will reflect the need for the Centre to complete marking of the tasks and submit the marks to OCR by the required date.
- There are no restrictions on computing facilities, hardware or software, that may be used.
- You are strongly advised to keep all your working notes as these may be required by the moderator.
- Reasons for answers to tasks are expected to form part of the work submitted.
- Once your tasks have been marked by the Centre, they cannot be re-submitted for improvement.

INFORMATION FOR CANDIDATES

- This document consists of **12** pages. Any blank pages are indicated.

Notice to candidates

1 The work which you submit for assessment must be your own.

However, you may:

(a) quote from books or any other sources: if you do, you must state which ones you have used;

(b) receive guidance from someone other than your teacher: if so, you must tell your teacher, who will record the nature of the assistance given to you.

2 If you copy from someone else or allow another candidate to copy from you, or if you cheat in any other way, **you may be disqualified from at least the subject concerned**.

3 When you hand in your coursework for assessment, you will be required to sign that you have understood and followed the coursework and portfolio requirements for the subject.

ALWAYS REMEMBER – YOUR WORK MUST BE YOUR OWN

Task 1 [47 marks]

This is a software development task and an implementation task.

You are to create a database to keep track of orders to a book publishing company. The company receives orders from customers and each customer may send many orders but an order can only be from one customer.

An order can be for many books and a book can be on many orders.

An order must contain the address for delivery, the customer's order number and the date of the order. Each line on an order consists of a quantity, an ISBN number, the title of the book and the name(s) of the author(s). Note that several customers can use the same order number.

You need to create four tables called

CUSTOMER – to hold customer details
ORDER – for order details
BOOK – to hold the details of the books
ORDER_LINE – to hold the details of each line on the order

The publishing company uses electronic data interchange and will accept orders by telephone, post or email. It wants to be able to send invoices electronically.

(a) Create a table called CUSTOMER to hold the details of each customer.
- Give the reason for including each of the attributes.
- Give the data type of each attribute.
- Identify the key.
- Show how validation has been used in the creation of the table. **[7]**

(b) Create a table called ORDER to hold details of each order.
- Give the reason for including each of the attributes.
- Give the data type of each attribute.
- Identify the key.
- Show how validation has been used in the creation of the table. **[7]**

(c) Create a table called BOOK to hold details of each book.
- Give the reason for including each of the attributes.
- Give the data type of each attribute.
- Identify the key.
- Show how validation has been used in the creation of the table. **[6]**

(d) Create a table called ORDER_LINE to hold details of each line on an order.
- Give the reason for including each of the attributes.
- Give the data type of each attribute.
- Identify the key.
- Show how validation has been used in the creation of the table. **[7]**

(e) Create suitable data for each of your tables.

- Include at least 5 customers, 10 orders and 15 books.
- The ORDER_LINE table should have at least 30 entries.
- Choose your data to be suitable for producing sensible results in the remaining question in part **(f)**.
- Include hard copy of the whole of each of your tables.
  Screen shots are acceptable. **[10]**

(f) For the rest of this task you may assume that no part orders are dispatched.

Create a delivery note to accompany the books ordered when they are delivered.

The delivery note should contain

- name and address of the publisher;
- name and delivery address of the customer;
- customer's order number and date of order;
- publisher's order number and date of delivery;
- quantity, ID, title and author(s) of each book ordered;
- all the books on the order.

Include

- evidence that the order chosen has at least three books on it;
  a screen shot is acceptable.
- evidence that the user has input the order to be used;
  a screen shot is acceptable.
- evidence to show how your solution created the delivery note;
  a screen shot is acceptable.
- a printed copy of your delivery note. **[10]**

Task 2 [26 marks]

This is a white box test task. No implementation is required.

Read the algorithm shown below, which validates a number and determines whether it is an integer or a real number.

The function **LEN(aString)** returns the number of characters in **aString**. For example, if aString = +2.3461, LEN(aString) = 7.

The function **SUBSTRING(aString, n, m)** returns **m** characters of **aString** starting at the **n**th. For example, if aString = computer, SUBSTRING(aString, 3, 4) is mput.

The algorithm refers to the one dimensional array **err**, which has been dimensioned to ten cells.

Each output starts on a new line.

```
1       INPUT aString
2       len = LEN(aString); count = 0
3       IF len = 0 THEN
4           OUTPUT "Null string not allowed"
5           STOP
6       END IF
7       wrong = FALSE
8       FOR i = 1 TO len
9          ch(i) = SUBSTRING(aString, i, 1)
10         err(i) = FALSE
11      NEXT i
12      IF len = 1 THEN
13          IF ch(1) < "0" OR ch(1) > "9" THEN
14              wrong = TRUE
15              err(1) = TRUE
16          END IF
17      ELSE
18          IF len = 2 THEN
19             IF ch(1) = "+" OR ch(1) = "-" OR ch(1) = "." THEN
20                  IF ch(1) = "." THEN
21                      count = count + 1
22                  END IF
23                  IF ch(2) = "." THEN
24                      count = count + 1
25                      IF count > 1 THEN wrong = TRUE ENDIF
26                  ELSE
27                     IF ch(2) <"0" OR ch(2) > "9" THEN
28                          wrong = TRUE
29                          err(2) = TRUE
30                     END IF
31                  END IF
32              ELSE
33                  IF ch(1) < "0" OR ch(1) > "9" THEN
34                      wrong = TRUE
35                      err(1) = TRUE
36                  ELSE
37                      IF ch(2) = "." THEN
```

```
                        count = count + 1
                        IF count > 1 THEN wrong = TRUE ENDIF
                    ELSE
                        IF ch(2) < "0" OR ch(2) > "9" THEN
                            wrong = TRUE
                            err(2) = TRUE
                        END IF
                    END IF
                END IF
            END IF
        ELSE
            IF ch(1) = "." THEN
                count = count + 1
                IF count > 1 THEN wrong = TRUE ENDIF
            ELSE
               IF ch(1) <> "+" AND ch(1) <> "-" THEN
                   IF ch(1) < "0" OR ch(1) > "9" THEN
                      wrong = TRUE
                      err(1) = TRUE
                   END IF
               END IF
            END IF
            FOR i = 2 To len
                IF ch(i) = "." THEN
                    count = count + 1
                    IF count > 1 THEN wrong = TRUE ENDIF
                ELSE
                    IF ch(i) < "0" OR ch(i) > "9" THEN
                        wrong = TRUE
                        err(i) = TRUE
                    END IF
                END IF
            NEXT i
        END IF
    END IF
    IF wrong THEN
        IF count > 1 THEN
            OUTPUT "Too many decimal points"
        END IF
        FOR i = 1 To len
            IF err(i) THEN
                OUTPUT "Error in ", i, "th character"
            END IF
        NEXT i
    ELSE
        IF count = 0 THEN
            OUTPUT "Valid integer number"
        ELSE
            OUTPUT "Valid real number"
        END IF
    END IF
```

(a) Write down the outputs from this algorithm when the following codes are input at line 1.

(i) 1.2+3

(ii) *14

(iii) .5.6 **[3]**

(b) During the tracing of the algorithm in **(a)**, changes will have occurred to the contents of the array **err** and the variable **wrong**.

For each of the three cases from **(a)**, state the contents of the array **err** and the variable **wrong** as they are at the end of the algorithm. **[6]**

(c) When the algorithm has the string +3 input at line 1, the lines

1, 2, 3, 6, 7, 8, 9, 10, 8, 9, 10, 11, 12, 17, 18, 19, 20, 23, 26, 27, 30, 31, 47, 48, 71, 72, 73, 82, 83, 84, 85, 87, 88

are executed and the output is

Valid integer string

This example of a test can be described more fully as shown in the table.

Type of test	Valid integer exactly 2 characters long
Test data	+3
Path	1, 2, 3(False), 6, 7, 8, 9, 10, 8, 9, 10, 11, 12(False), 17(True), 18(True), 19(True), 20(False), 23(False), 26(True), 27(False), 30, 31, 32(False) 47, 48(False), 71, 72, 73(False), 82(True), 83(True), 84, 85(False), 87, 88
Expected output	Valid integer string

Note:

- When the path is described, each IF and ELSE line must show whether the IF or ELSE test led to a (True) or (False) result.
- When the path is described, FOR-NEXT loops should be repeated the appropriate number of times.

Using tables in the same form as the table above, write down

- the lines that are executed
- the expected output

when the following codes are input to the algorithm.

(i) . (decimal point) **[5]**

(ii) +.– (plus sign, decimal point, minus sign) **[9]**

(d) In the algorithm, lines 77 to 81 contain a loop that outputs which, if any, of the characters are invalid. The phrase OUTPUT is unacceptable for the first three characters, but not for the rest of the characters.

Write an amended version of lines 77 to 81 to remedy this problem. **[3]**

Task 3 [47 marks]

This is a software development task and an implementation task.

Fig. 3.1 shows a simple maze. The starting position is in cell (1, 1) and the finishing position is in cell (6, 6) using the usual notation (row, column).

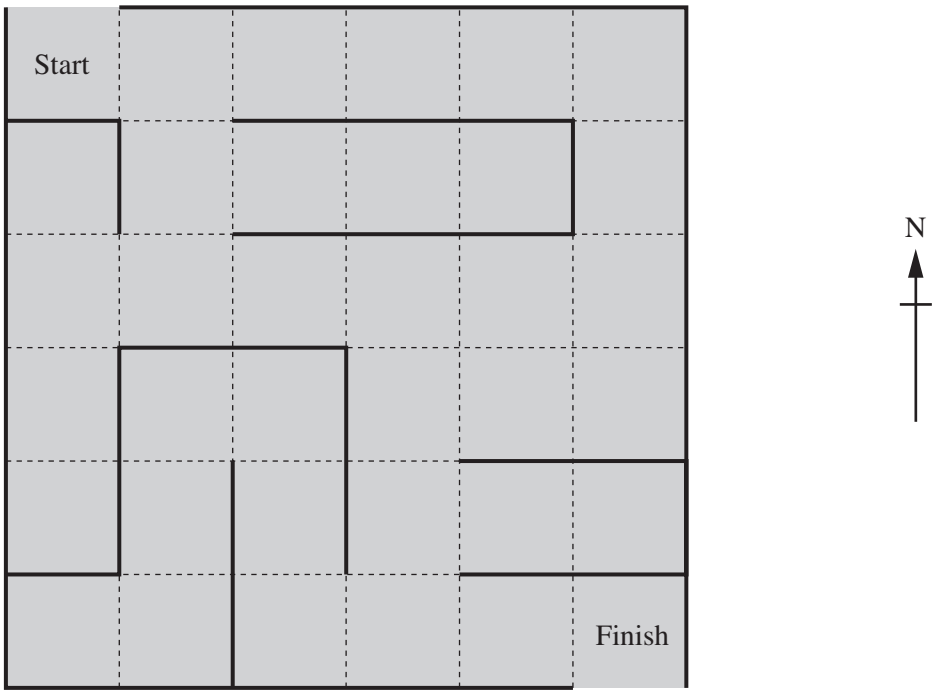

Fig. 3.1

You will need to represent the maze by a 2-D array. For the purpose of this task you should use the following rules.

- Represent each square of the maze by a 4-digit binary number.
- It is possible to move **west** if the **first** digit is 1.
- It is possible to move **south** if the **second** digit is 1.
- It is possible to move **east** if the **third** digit is 1.
- It is possible to move **north** if the **fourth** digit is 1.

You will also need to know which way you are facing in each cell of the maze. For the purpose of this task you should use the following rules.

- Facing **west** is represented by **3**.
- Facing **south** is represented by **2**.
- Facing **east** is represented by **1**.
- Facing **north** is represented by **0**.

In the maze in Fig. 3.1, cell (4, 2) can be represented by the 4-digit binary string 0 1 1 0.

If Face is a variable holding the direction in which you are facing, then 3 would indicate that you are facing west.

Now number the digits in the 4-digit string 0 to 3 from the RIGHT.

Now, if Face holds the number 3 and digit 3 is a 1, you can move west. If digit 3 is a zero, you cannot move west because there is a wall in the way.

One algorithm for finding a way out of a maze is to place your left hand on the wall and move through the maze without taking your hand off the wall.

In Fig. 3.1 this would mean visiting cells (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 6), (3, 6), (4, 6), (4, 5), (4, 4), (5, 4), (5, 5), (5, 6), (5, 5), (5, 4), (6, 4), (6, 5), (6, 6).

This Task is to be solved using this method.

(a) Create a text file to hold the following data for the maze in Fig. 3.1.

- The number of rows in the maze.
- The number of columns in the maze.
- The coordinates of the starting cell.
- The coordinates of the finishing cell.
- The direction in which you are facing at the start.
- The 4-digit binary strings representing the cells.

Provide hard copy evidence that you have created this file.
A screen shot is acceptable. **[8]**

(b) Write program code that will input the data in the file created in **(a)** ready to solve the maze problem. For this task you are to assume that all the data are correct.

Provide hard copy evidence of your code, which should be fully annotated. Use meaningful names for all variables, labels, buttons etc. **[6]**

(c) Create code to solve the maze problem for any maze up to 10 rows by 10 columns.

Your code should output

- a list of all the cells on the route from the start cell to the finish cell;
- the number of cells on the route.

Provide hard copy evidence of your code, which should be fully annotated. Use meaningful names for all variables, labels, buttons etc. **[9]**

(d) Create a rectangular maze with at least 5 rows and at least 6 columns.

The path through the maze should involve passing through at least 15 cells.

Provide a diagram of your maze (this may be hand drawn) and give the expected path through the maze. **[5]**

(e) Create a file to hold the data for your maze designed in part **(d)**.

Provide hard copy evidence that you have created this file.
A screen shot is acceptable.

Run your program using this data and provide hard copy evidence of the result. **[10]**

(f) Develop your program code so that all data are validated.

Test your code using a suitable file.

Hand in a copy of the new code and the results of your tests. **[9]**

BLANK PAGE