Computing Science (Advanced Higher)

Understanding Standards Events 2019 —

Workshop Tasks (7, 8 and 9)

Contents

Workshop 7

Computing Science Advanced Higher

This workshop focussed on the sample course delivery plan on the following pages.

Discuss the following:

- Would this delivery plan work for you?
- What changes would you make?
- How do you plan to integrate course content?

Term	Area	Content	~hr	Notes
June (~4 weeks)	SDD/WDD design	Pseudocode design for server-side processes	1	Begin with the course ethos of integration by providing worked examples of SDD/DDD and WDD/DDD links. Use this to show similarities between the two (link to database, execute query, format results) through design. Keep integration implementation simple: small program, single web page, single-table database. Web/database server setup will be introduced during this work. During the web example a simple media query will be added to the web page to create a tablet or app view of the page. The use of hexadecimal as a computing abbreviation will be discussed in the context of web colours. Practice hex to denary conversion. (H)
	SDD implementation	Connecting a programming language to database	3	
	DDD implementation	Connecting web page to a database	11	
	WDD implementation	HTML – Forms (action, method, name, value)	2	
	WDD implementation	Media queries	2	
	Computer Systems	Hexadecimal	1	
August – October (~8 weeks)	Computer Systems	Risks of SQL code injections	1	Provide a worked example of analysis in the context of software development. Deliver initial OO theory and provide worked example of a UML class diagram. Complete design tasks for other scenarios (H).
	SDD analysis	Feasibility studies, user surveys, planning and use-case diagram	2	
	SDD design	UML class diagrams and user interface design	5	

	SDD Implementation	OO Programming	17	Work through a variety of OO programming tasks. Deliver theory and problem solving exercises using the three standard algorithms. (H) Code each algorithm using procedural and OO examples.
	SDD Implementation	Standard algorithms	5	
	DBDD analysis	Inputs, process, output, use-case diagram	4	Provide a worked example of problem analysis in a database context followed by a few similar tasks to complete. (H) Deliver new ERD theory followed by ERD creation tasks include identifying relationship types from EOD. (H)
	DBDD design	ERD, EOD, data dictionary	3	
	Project	Introduction	1	Discuss project requirements, timing including deadlines, assessment arrangements.
		Analysis and Design	2	
October – December (~8 weeks)	Project	Analysis and Design	2	Complete analysis and design tasks for selected project.
	DBDD implementation	SQL CREATE with constraints	3	Provide data dictionaries for a four-table relational database. Implement CREATE statements in database server admin view to create the tables and constraints.
	DBDD design & implementation & testing	Query design and implementation of SQL HAVING, sub-queries, logical operators	20	Provide a large, populated relational database, ERD and data dictionaries. Design, implement and test a variety of Queries requiring AH SQL. (H)
	Project	Implementation	15	Begin implementation of project design. Note evidence for ongoing testing report.
January – April	SDD Implementation	OO Programming	15	Complete OO programming.

(~10 weeks)	SDD testing	OO Programming	6	Discuss the range of testing that programs are subjected to. Create and implement test plans, including personas, for a couple of completed programs. (H)
	Project	Implementation	13	Complete implementation of project design. Note evidence for ongoing testing report. Write report on ongoing testing.
	Project	Testing	7	Formulate test plan. Implement test plan and gather evidence of testing.
	SDD, DDD, WDD Evaluation	fitness for purpose, maintainability, robustness	3	Discuss the link between requirements identified at analysis and fitness for purpose. Provide a list of requirements and a working solution to a problem. Set a task to evaluate the solution's fitness for purpose. (H) Discuss the maintainability of a project in terms of further changes to the solution (perfective, corrective, adaptive). Evaluate the maintainability of the above solution.
	Project	Evaluation	2	Write evaluation report for project and submit completed project.
	Computer Systems	fetch-execute cycle, binary addition, flags and pipe-lining	3	Delivery theory of fetch-execute cycle and the improved efficiency of pipe-lining instructions. Discuss the low-level machine and the primary function of adding two binary values. Practice binary addition and the use of flags in low-level programming. (H)
	Computer Systems	Environmental impact	1	Deliver theory on the environmental impact of data centres.

Throughout the course exam-style questions will be set as regular homework tasks. These will cover the full curriculum but with particular focus on problem design, reading code and writing code.

Workshop 8

Computing Science Advanced Higher

This workshop focussed on the object-oriented content of the course.

Discuss the following:

- extend Question 1 to include a question that requires students to make use of the find maximum algorithm
- extend Question 2 to include a question that requires students to make use of the linear search algorithm

Question 1: Vehicles

An object-oriented program makes use of a `Vehicle` class to store details of a company's vehicles. .

The `Van` class is a subclass of the `Vehicle` class with additional instance variables:

- `capacity` that represents the maximum load space measured in litres
- `tailLift` that represents whether the van has a tail lift or not

The class definition for the `Vehicle` class has been provided below. You should note that each of the three instance variables are private variables.

```
CLASS Vehicle IS { STRING regNumber, STRING make, STRING colour }

METHODS

      PROCEDURE Vehicle(STRING reg, STRING mke, STRING col)
             DECLARE THIS.regNumber INITIALLY reg
             DECLARE THIS.make INITIALLY mke
             DECLARE THIS.colour INITIALLY col
      END PROCEDURE


      PROCEDURE setColour(STRING col)
             SET THIS.colour TO col
      END PROCEDURE


      PROCEDURE purpose()
             SEND "I carry passengers" TO DISPLAY
      END FUNCTION


      FUNCTION getMake() RETURNS STRING
             RETURN THIS.make
      END FUNCTION

END CLASS
```

a) i) Use the OO terms instantiation and inheritance to explain the purpose of the statement:

```
DECLARE vehicle1 AS Van INITIALLY ("ABC 123D", "Ford",
"white", 50, false)
```

2

ii) The statement `vehicle1.purpose()` should produce the message "I carry cargo". Explain how polymorphism would apply in this situation. **2**

iii) Use a programming language of your choice to write the class definition for the `Van` class. In addition to the its constructor and `purpose()` methods, this class should provide getter methods that enable external code to access the values stored in its two instance variables. **4**

iv) Use appropriate object-oriented terminology to explain why the following statement is invalid. **2**

```
SET vehicle1.regNumber TO "XYZ 987W"
```

b) Draw a UML class diagram to represent the structure of the `Vehicle` and `Van` classes. **3**

c) An array of `Van` objects called `vanDetails` is used to store details of the 25 vans in the company fleet. The following statement in the main program is used to activate the function `count()`.

```
SET numberFordVans TO count(vanDetails)
```

This function is used to calculate and return the number of Ford vans in the company fleet.

Use a programming language of your choice to write the code for this `count()` function. **3**

Marking Instructions

a) i) A new object called `vehicle1` has been instantiated. This object belongs to the `Van` class. Since `Van` is a subclass of the `Vehicle` class, `vehicle1` inherits each instance variable and method belonging to the `Vehicle` superclass. The values provided in the `DECLARE` statement are assigned, in the sequence listed, to the 3 instance variables inherited from the `Vehicle` class and the additional two instance variables belonging to the `Van` class.

Award 1 mark for explanation of instantiation that makes reference to the code provided.
Award 1 mark for explanation of encapsulation that makes reference to the code provided.

ii) The `vehicle1` object inherits the `purpose`() method from the `Vehicle` superclass. Since the output from this inherited method differs from the output that is required, polymorphism must be used to redefine the `purpose`() method for the `Van` subclass. In this way, the `purpose`() method for the `Van` subclass overrides the inherited method thereby allowing all `Van` object to respond differently.

Award 1 mark for explanation of inherited method.
Award 1 mark for explanation of the need to use polymorphism to override the inherited method in order to alter its behaviour.

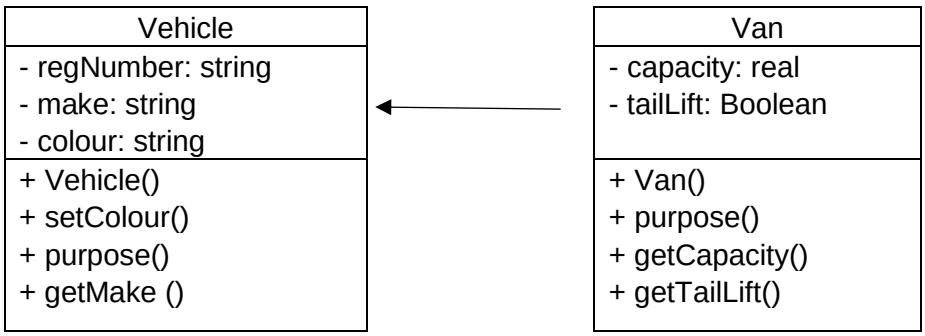
iii)
```
CLASS Van INHERITS Vehicle WITH { REAL capacity, BOOLEAN
tailLift}
METHODS
    PROCEDURE Van(REAL cap, BOOLEAN tail)
            DECLARE THIS.capcity INITIALLY cap
            DECLARE THIS.tailLift INITIALLY tail
    END PROCEDURE

    OVERRIDE PROCEDURE purpose()
            SEND "I carry cargo" TO DISPLAY
    END FUNCTION

    FUNCTION getCapacity() RETURNS REAL
            RETURN THIS.capacity
    END FUNCTION

    FUNCTION getTailLift() RETURNS BOOLEAN
            RETURN THIS.tailLift
    END FUNCTION
END CLASS
```

Award 1 mark for code that indicates inheritance from the `Vehicle` class with two additional instance variables.
Award 1 mark each for constructor and `purpose()` methods.
Award 1 mark for both getter methods.

iv) The instance variable `regNumber` is private, meaning that it is encapsulated. To access the value stored in the variable and edit its contents, a method must be used: the instance variable `regNumber` cannot be edited directly.

Award 1 mark for explanation that refers to encapsulation of the variable.
Award 1 mark for explanation that refers to the need to use a method.

b)



Award 1 mark for correct instance variables and methods of `Vehicle` class
Award 1 mark for correct instance variables and methods of `Van` class
Award 1 mark for correct indication of inheritance

c)
```
FUNCTION count (ARRAY OF Van vanDetails) RETURNS INTEGER
   SET total TO 0
   FOR index FROM 0 TO 24 DO
        IF vanDetails[index].getMake() ="Ford" THEN
            SET total TO total + 1
        END IF
   END FOR
   RETURN total
END FUNCTION
```

Award 1 mark for correct use of `vanDetails` **array**
Award 1 mark for correct use of `getMake()` **method**
Award 1 mark for correct processing of array total

Question 2: Players

A `Player` object is defined by the `Player` class shown below.

```
CLASS Player { STRING name, INTEGER score, STRING location }

METHODS

    CONSTRUCTOR Player(STRING nme, INTEGER scr, STRING loc)
        DECLARE THIS.name INITIALLY nme
        DECLARE THIS.score INITIALLY scr
        DECLARE THIS.location INITIALLY loc
    END CONSTRUCTOR

    FUNCTION getName() RETURNS STRING
        RETURN THIS.name
    END FUNCTION

    FUNCTION getScote() RETURNS INTEGER
        RETURN THIS.score
    END FUNCTION

END CLASS
```

a) Describe the purpose of the constructor method shown in the class definition code above. **2**

An array of `Player` objects called `topTen` contains the names and scores of the 10 highest scoring players in an online computer game. These details are stored in descending order of score.

b) The first three members of the array `topTen` are shown below.

`topTen`

[0]			[1]			[2]		
Jo	964	Ayr	Pete	900	York	Sofia	840	Rome

State the value returned by:

i) `topTen[1].getName()`

ii) `topTen[2].getScore ()` **2**

c) At the end of each game, a new `Player` object called `newPlayer` is created.

The method `compare()` receives the `newPlayer` object containing the player's details and the `topTen` array of `Player` objects. The method compares the new player's score with those in the `topTen` array and returns the position in which the new score should be inserted in the `topTen` array, or the value –1 if the new score is not high enough to be included.

The method `compare()` has been started below:

```
FUNCTION compare(Player newPlayer, ARRAY OF Player
topTen) RETURNS INTEGER
   ## lines of code missing


END FUNCTION
```

Write the missing code for this `compare()` method. **4**

d) A method `calculateAverage()` is used to calculate and return the average score of the scores stored in the `topTen` array. This function is activated in the main program using the statement:

```
SET averageScote TO calculateAverage(topTen)
```

Write the code for this `calculateAverage()` method. **3**

Marking Instructions

a) Whenever it is invoked, this constructor method is used to instantiate a new object that that belongs to the `Player` class. The values provided by the user are assigned to this new object's three instance variables `name`, `score` and `location`.

Award 1 mark for explanation that refers to instantiation of an object
Award 1 mark for explanation that refers to assignment of values to instance variables

b) i) Pete
ii) 840

Award 1 mark each

c)

```
FUNCTION compare (Player newPlayer, ARRAY OF Player topTen)
RETURNS INTEGER
    SET index TO 0
    SET include TO false
    SET position TO -1
    REPEAT UNTIL include = true OR index = 10
       IF newPlayer.getScore() > topTen[index].getScore() THEN
          SET include TO true
          SET position TO index
       END IF
    END REPEAT
    SET index TO index + 1
    RETURN position
END FUNCTION
```

Award 1 mark for correct use of `topTen` **array**
Award 1 mark for correct use of `getScore()` **method**
Award 1 mark for traversing `topTen` **array**
Award 1 mark for correcting recording insertion position

d)

```
FUNCTION calculateAverage (ARRAY OF Player topTen) RETURNS
REAL
    SET result TO 0.0
    FOR index FROM 0 TO 9
       SET result TO result + topTen[index].getScore()
    END FOR
    SET result TO result/10
    RETURN result
END FUNCTION
```

Award 1 mark for correct use of `topTen` **array**
Award 1 mark for correct use of `getScore()` **method**
Award 1 mark for correct processing of array average

Workshop 9

Computing Science Advanced Higher

This workshop focussed on the 2-D array content of the course.

Discuss the following:

- extend Question 3 to include a question that requires students to make use of a sort algorithm
- extend Question 4 to include a question that requires students to make use of the count occurrences algorithm

Question 3: Store Cards

A retail store employs ten sales staff. The store keeps a record of the number of new store cards issued by its sales staff over the first six months of the year. Sample data is shown in the table below.

	Jan	Feb	Mar	Apr	May	Jun
Adams	12	12	6	8	3	2
Burns	12	17	7	4	5	9
Cook	2	12	0	12	0	3
Davies	4	10	7	4	8	9
East	5	0	0	0	0	0
Faass	6	1	4	6	7	18
Gray	12	19	12	16	17	7
Hill	13	9	7	3	4	5
Iozzi	12	8	4	4	5	4
Jian	14	11	12	4	5	6

The sales data is to be stored in a 2-dimensional array called `storeCards` with each row of the array representing the sales for one salesperson and each column representing a month.

Two separate 1-dimensional arrays called `person` and `month` will be used to store the name of each salesperson and the names of the first six months of the year.

The sample data in the table above would be stored in the three arrays as shown below.

`storeCards`

12	12	6	8	3	2
12	17	7	4	5	9
2	12	0	12	0	3
4	10	7	4	8	9
5	0	0	0	0	0
6	1	4	6	7	18
12	19	12	16	17	7
13	9	7	3	4	5
12	8	4	4	5	4
14	11	12	4	5	6

`person`

Adams
Burns
Cook
Davies
East
Faass
Gray
Hill
Iozzi
Jian

`month`

Jan
Feb
Mar
Apr
May
Jun

a) i) Use a programming language of your choice to write a declaration statement for the two-dimensional array `storeCards`. **2**

ii) Write the statement used to assign the value of the April sales for salesperson Gray. **1**

b) Describe the purpose of the following section of code. **2**

```
SEND "Enter a number 1 to 10 to represent one salesperson" TO DISPLAY
RECEIVE salesPersonNumber (INTEGER) FROM KEYBOARD
SET personTotal TO 0
FOR index FROM 0 TO 5 DO
  SET personTotal TO personTotal + storeCards[salesPersonNumber-1,
index]
END FOR
SEND "The result is " & personTotal TO DISPLAY
```

c) The one-dimensional array called `monthlyTotals` will store the total number of new cards issued each month. Use a programming language of your choice to write the code for a procedure that can used to work out each monthly total and assign them to the array called `monthlyTotals`. **3**

d) Any member of staff who issues 16 or more cards in any month is due to receive a bonus. Details of qualifying staff are to be stored in a database called `StaffBonus` in a table called `Bonus`.

Use pseudocode to design an algorithm to identify qualifying staff and store the relevant details in the database table called `Bonus`.

Structure of the `Bonus` table

field	key	validation
id	PK	auto number
salesPersonID		required
monthID		required
cardsInMonth		required

5

e) Write the code for a procedure called `display` that can be activated in the main program and used to display the name of the salesperson who issued the highest number of store cards in any one month in the first half of the year. **4**

Marking Instructions

a) i) `DECLARE storeCards AS ARRAY OF ARRAY OF INTEGER < 2D array with 10 rows and 6 columns, all elements set initially to zero >`

Award 1 mark for correct dimensions
Award 1 mark for correct data type

ii) `SET storeCards [6] [3] TO 16`

Award 1 mark for correct assignment

b) This section of code allows the user to enter a value 1–10 that represents one of the company sales staff and acts as the row index for the 2D array. The code then totals the values in all six columns of the selected row of the 2D array.

The total sales for the required sales person for the first six months of the year is displayed on the screen.

Award 1 mark for description that refers to user selection used as row index
Award 1 mark for description that refers to totalling of the sales for the required sales person

c)
```
PROCEDURE calculate (ARRAY OF ARRAY OF INTEGER storeCards, ARRAY
OF INTEGER monthlyTotals)
   FOR row FROM 0 TO 9 DO
      FOR column FROM 0 TO 5 DO
          SET monthlyTotals[column] TO monthlyTotals[column} +
          storeCards[row][column]
      END FOR
   END FOR
END PROCEDURE
```

Award 1 mark for correct use of nested loop
Award 1 mark for correct use of column index to process `monthlyTotals` **array**
Award 1 mark for correct use of row and column indices to process `storeCards` **array**

d)
1. open connection with StaffBonus database on the secure database server
2. start loop for each row from 0 to 9
3. start loop for each column from 0 to 5
4. if storeCards[row][column] >= 16 then
5. create SQL INSERT query to add the salesperson's id, month id and number of store cards issued to the Bonus table
6. execute SQL INSERT query
7. end if
8. end column loop
9. end row loop
10. close connection with database server

Award 1 mark for open and close connection to the database
Award 1 mark for use of nested loop
Award 1 mark for correct use of row and column indices to process `storeCards` **array**
Award 1 mark for correct generation of `INSERT` **query to add salesperson's details to the database**
Award 1 mark for execution of the `INSERT` **query**

e)
```
PROCEDURE display (ARRAY OF ARRAY OF INTEGER storeCards, ARRAY OF
STRING person)
   SET maxIssued TO -1
   SET bestPerson TO -1
   FOR row FROM 0 TO 9 DO
      FOR column FROM 0 TO 5 DO
          IF storeCards[row][column] > maxIssued THEN
             SET maxIssued TO storeCards[row][column]
             SET bestPerson TO row
          END IF
      END FOR
   END FOR
   SEND "The salesperson who issued the most store cards in the
   first half of the year is " & person[bestPerson] TO DISPLAY
END PROCEDURE
```

Award 1 mark for nested loop
Award 1 mark for correct use of `storeCards` **array using row and column indices**
Award 1 mark for correct use of find max algorithm to determine correct sales person
Award 1 mark for correct use of row index to process `person` **array**

Question 4: Bingo Cards

A printing company uses a computer program to randomly generate and print bingo tickets.

Each bingo ticket has a grid with three rows and nine columns; each row on the ticket contains five numbers and four blank spaces.

4			32	45		68		82
9		26			51	62		88
		24		47	55	65	71	

a) Use a programming language of your choice to define a two-dimensional array called `ticket` to store the numbers selected for an individual bingo ticket. **2**

b) To generate the tickets, the program first fills in the columns with random integers as specified in the table below.

Column	Highest Possible Random Integer	Lowest Possible Random Integer
1	10	1
2	20	11
3	30	21
4	40	31
5	50	41
6	60	51
7	70	61
8	80	71
9	90	81

Numbers selected can only appear once on each bingo ticket.

Use pseudocode to write an algorithm to fill <u>each</u> element of the array called `ticket` with random numbers according to the rules specified above.

Note: there is no need to sort the numbers in each column of the bingo ticket. **5**

c) After filling the array with randomly selected numbers, the program replaces four separate positions on each row with the number 0. The bingo ticket is then printed using the following rules.

- If the value of the array cell is 0 then display a space
- Otherwise, display the value stored in the array cell

Use pseudocode to describe an algorithm that could be used to print the numbers in the array onto a ticket. **4**

Marking Instructions

a)

```
DECLARE ticket AS ARRAY OF ARRAY OF INTEGER < 2D array with 3
rows and 9 columns, all elements set initially to zero >
```

Award 1 mark for correct dimensions
Award 1 mark for correct data type

b)

```
1.  set all 90 elements in chosen array to false
2.  set selected to false
3.  start loop for each row from 0 to 2
4.      start loop for each column from 0 to 8
5.          repeat until selected = true
6.              select random number between column*10+1 and (column+1)*10
7.              if chosen[random number] = false then
8.                  set chosen[random number] to true
9.                  set tickets[row][column] to random number
10.                 set selected to true
11.             end if
12.         end repeat
13.         set selected to false
14.     end column loop
15.  end row loop
```

Award 1 mark for correct use of nested loop
Award 1 mark for generation of random numbers
Award 1 mark for checking that number selected has not already been used
Award 1 mark for use of conditional loop to select three unique numbers for each column
Award 1 mark for assignment of random number to 2D array

c)

```
1.  start loop for each row from 0 to 2
2.      set all 9 elements in successfulChoices array to false
3.      start loop for choices from 1 to 4
4.          set success to false
5.          repeat until success = true
6.              select random column between 0 and 8
```

```
7.                      if successfulChoices [random column] = false then
8.                          set successfulChoices [random column] to true
9.                          set tickets[row][random column] to 0
10.                         set success to true
11.                     end if
12.                 end repeat
13.             end choice loop
14.             start loop for each column from 0 to 8
15.                 if tickets[row][column] = 0 then
16.                     display space
17.                 else
18.                     display ticket[row][column]
19.                 end if
20.         end row loop
```

Award 1 mark for random selection of 4 cells in each row of the 2D array
Award 1 mark for ensuring that 4 different cells are selected in each row
Award 1 mark for allocation of zero to 4 cells in each row of the array
Award 1 mark for displaying contents of 2D array