



1.2 Programming paradigms

Name: _____

Class: _____

Date: _____

Time: **499 minutes**

Marks: **318 marks**

Comments:

Q1.

- (a) This question refers to the subroutine `CreateTileDictionary`.

The points values for the letters J and X are to be changed so that they are not worth as many points as the letters Z and Q.

Adapt the subroutine `CreateTileDictionary` so that the letters J and X are worth 4 points instead of 5 points.

Test that the changes you have made work:

- run the **Skeleton Program**
- enter 2 at the main menu
- enter 1 to view the letter values.

Evidence that you need to provide

- (i) Your PROGRAM SOURCE CODE for the amended subroutine `CreateTileDictionary`.

(1)

- (ii) SCREEN CAPTURE(S) showing the results of the requested test.

(1)

- (b) This question refers to the subroutine `Main`.

Currently each player starts with 15 letter tiles. In the `Main` subroutine `StartHandSize` is set to a value of 15.

Change the subroutine `Main` so that the user can choose what the value for `StartHandSize` will be.

Before the main menu is displayed and before the first iteration structure in `Main` the message "Enter start hand size:" should be displayed and the user's input should be stored in `StartHandSize`. This should happen repeatedly until the user enters a value for the start hand size that is between 1 and 20 inclusive.

Test that the changes you have made work:

- run the **Skeleton Program**
- enter 0 when asked to enter the start hand size
- enter 21 when asked to enter the start hand size
- enter 5 when asked to enter the start hand size
- enter 1 at the main menu.

Evidence that you need to provide

- (i) Your PROGRAM SOURCE CODE for the amended subroutine `Main`.

(4)

- (ii) SCREEN CAPTURE(S) showing the requested test. You must make sure that evidence for all parts of the requested test is provided in the SCREEN CAPTURE(S).

(1)

- (c) This question refers to the subroutine `CheckWordIsValid`.

When a player enters a word, a linear search algorithm is used to check to see if the word entered is in the list of `AllowedWords`. The subroutine `CheckWordIsValid` is to be changed so that it uses a more time-efficient search algorithm.

Change the `CheckWordIsValid` subroutine so that it uses a binary search algorithm instead of a linear search algorithm.

You **must write your own search routine** and not use any built-in search function that might be available in the programming language you are using.

Each item in `AllowedWords` that is compared to the word that the user entered should be displayed on the screen.

Figure 2 shows examples of how the new version of `CheckWordIsValid` should work if `AllowedWords` contained the items shown in **Figure 1**.

Figure 1

BIG
BUG
FED
GET
JET
NOT
SIP
WON

Figure 2

- 1) If the user enters the word BIG then a value of `True` should be returned and the words GET, BUG, BIG should be displayed, in that order.
- 2) If the user enters the word JET then a value of `True` should be returned and the words GET, NOT, JET should be displayed, in that order.
- 3) If the user enters the word ZOO then a value of `False` should be returned and the words GET, NOT, SIP, WON should be displayed, in that order.

Test that the changes you have made work:

- run the **Skeleton Program**
- if you have answered part (b), enter 15 when asked to enter the start hand size; if you have not answered part (b) yet then skip this step
- enter 2 at the main menu
- enter the word `jars`.

Evidence that you need to provide

- (i) Your PROGRAM SOURCE CODE for the amended subroutine `CheckWordIsValid`.

(8)

- (ii) SCREEN CAPTURE(S) showing the requested test.

- (d) This question extends the functionality of the game.

After spelling a valid word the player decides which one of four options to select to determine how many tiles will be added to their hand. Before choosing they can look at the values of the tiles.

It would help the player make their decision if they were aware of how useful each letter was by knowing the frequency with which each letter appears in the list of allowed words.

The program is to be extended so that when the player chooses to view the tile values they are also shown the number of times that each letter appears in the list of allowed words.

What you need to do

Task 1

Create a new subroutine called `CalculateFrequencies` that looks through the list of allowed words and displays each of the 26 letters in the alphabet along with the number of times that the letter appears in the list of allowed words, which the subroutine has calculated.

Task 2

Modify the `DisplayTileValues` subroutine so that after displaying the tile values it also calls the `CalculateFrequencies` subroutine.

Task 3

Test that the changes you have made work:

- run the **Skeleton Program**
- if you have answered part (b), enter 15 when asked to enter the start hand size; if you have not answered part (b) yet then skip this step
- enter 2 at the main menu
- enter 1 to display the letter values.

Evidence that you need to provide

- (i) Your PROGRAM SOURCE CODE for the new subroutine `CalculateFrequencies`, the amended subroutine `DisplayTileValues` and any other subroutines you have modified when answering this question.

(8)

- (ii) SCREEN CAPTURE(S) showing the requested text.

(1)

- (e) The scoring system for the game is to be changed so that if a player spells a valid word they score points for all valid words that are a prefix of the word entered. A prefix is the first x characters of a word, where x is a whole number between one and the number of characters in the word.

In the **Skeleton Program**, `AllowedWords` contains the list of valid words that have been read in from the **Data File** `aqawords.txt`.

Example

If the user enters the word TOO they will be awarded points for the valid words TOO and TO as TO is a prefix of the word TOO. They will not be awarded points for the word OO even though it is a valid word and is a substring of TOO because it is not a prefix of TOO.

Example

If the user enters the word BETTER they will be awarded points for the words BETTER, BET and BE as these are all valid prefixes of the word entered by the user. They would not be awarded points for BETT or BETTE as these are not valid English words. They would not be awarded points for BEER as even though it is contained in the word BETTER it is not a prefix.

Example

If the user enters the word BIOGASSES they will be awarded points for the words BIOGASSES, BIOGAS, BIOG, BIO and BI as these are all valid prefixes of the word entered by the user. They would not be awarded points for BIOGA, BIOGASS or BIOGASSE as these are not valid English words. They would not be awarded points for GAS as even though it is contained in the word BIOGASSES it is not a prefix.

Example

If the user enters the word CALMIEST they will not be awarded any points as even though CALM at the start is a valid word the original word entered by the user, CALMIEST, is not.

Example

If the user enters the word AN they will be awarded points for the word AN. They would not be awarded points for A even though A at the start is a valid word as points are only awarded for words that are at least two letters long.

What you need to do

Task 1

Write a **recursive** subroutine called `GetScoreForWordAndPrefix` that, if given a valid word, returns the score of the word added to the score for any valid words that are prefixes of the word.

To get full marks for this task the `GetScoreForWordAndPrefix` subroutine must make use of recursion in an appropriate way to calculate the score for any prefixes that are also valid words.

If your solution uses an alternative method to recursion you will be able to get most but not all of the available marks for this question.

Task 2

Modify the `UpdateAfterAllowedWord` subroutine so that it calls the new `GetScoreForWordAndPrefix` subroutine instead of the `GetScoreForWord` subroutine.

Task 3

Test that the changes you have made to the program work:

- run the **Skeleton Program**
- if you have answered part (b), enter 15 when asked to enter the start hand size; if you have not answered part (b) yet then skip this step
- enter 2 at the main menu
- enter the word `abandon`
- enter 4 so that no tiles are replaced.

Evidence that you need to provide

- (i) Your PROGRAM SOURCE CODE for the new subroutine `GetScoreForWordAndPrefix`, the amended subroutine `UpdateAfterAllowedWord` and any other subroutines you have modified when answering this question.

(11)

- (ii) SCREEN CAPTURE(S) showing the results of the requested test.

(1)

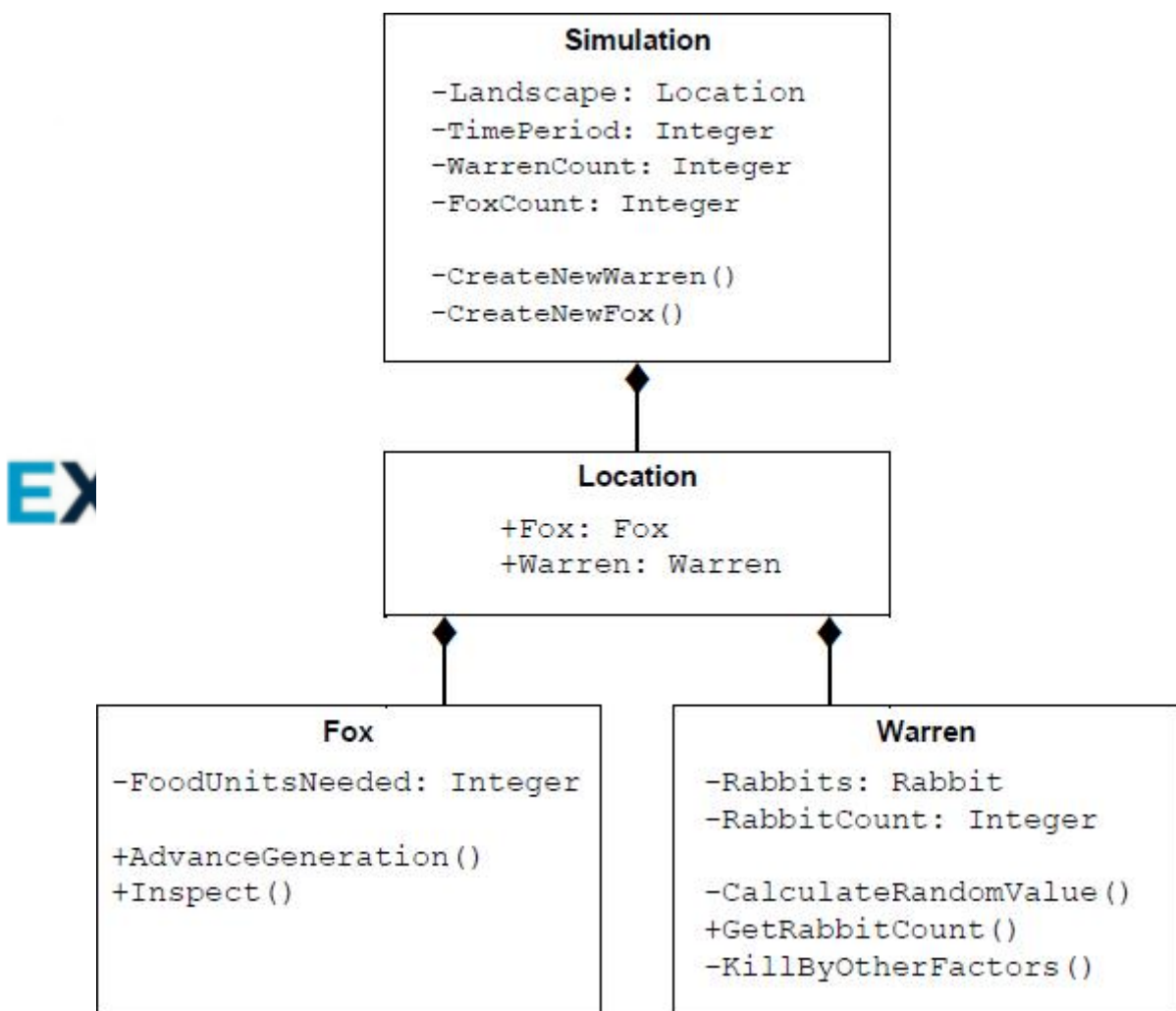
(Total 37 marks)

Q2.

The class diagram in **Figure 1** is a partial representation of the relationships between some of the classes in the **Skeleton Program**.

Note: In **Figure 1** a + sign denotes a public attribute/method.

Figure 1



A class diagram can show a variety of features used in object-oriented programming.

- (a) Write **Yes** or **No** in the unshaded cells in the table to identify if the given feature is present in the class diagram shown in **Figure 1**.

Feature	Is present in Figure 1? (Yes/No)
Inheritance	
Protected method	
Private attribute	

(3)

- (b) State the name of an identifier for a subclass in the **Skeleton Program**.

(1)

- (c) Explain the difference between a protected attribute and a private attribute.

(2)

- (d) In the `Warren` class there is an attribute `RabbitCount` and a method `GetRabbitCount`.

Explain the need for the `GetRabbitCount` method **and** explain why this approach is favoured in object-oriented programming.

EXAM PAPERS PRACTICE

(2)

- (e) During the simulation rabbits will die for a variety of reasons. One of these reasons is old age and at the end of the `KillByOtherFactors` method there is a call to `CompressRabbitList`.

Explain the need for the `CompressRabbitList` method.

Part of the class definition for `Rabbit` has been represented in **Figure 2**.

Figure 2

```

Rabbit = Class (Animal)
  Private:
    ReproductionRate: Real
    Gender: Genders
  Public:
    Procedure Inspect()
    Function IsFemale()
    Function GetReproductionRate()
End Class
  
```

A new animal is to be introduced into the simulation. This animal is the `HDRabbit`, which represents a rabbit with haemorrhagic disease. The class `HDRabbit` is to be a subclass of the `Rabbit` class. When an `HDRabbit` is inspected it should display all the information shown for a normal rabbit plus the additional information stored about an `HDRabbit`.

An `HDRabbit` has the following additional attributes:

- `InfectionRate`: stores a value that represents the probability of a rabbit that is bred from an `HDRabbit` being infected
- `Generation`: stores a value that represents how many generations have had this disease in this rabbit's family.

An `HDRabbit` has additional methods including:

- `IsInfertile()`: returns `True` if the haemorrhagic disease has been in this rabbit's family for three generations.
- (f) Write the class definition for `HDRabbit`, using similar notation to that used in **Figure 2**. You are **not** expected to make any changes to the Skeleton Program.

EXAM PAPERS PRACTICE

(4)

(Total 14 marks)

Q3.

- (a) This question refers to the subroutine `InputCoordinate` in the `Simulation` class.

The warren and fox inspection options in the **Skeleton Program** do not currently check if the coordinates entered by the user are on the landscape. This behaviour needs to be improved so that an error message is displayed if the user inputs coordinates for a location that is not on the landscape.

If the user runs a simulation with default settings then the landscape size is 15, so valid locations have an x coordinate between 0 and 14, inclusive.

What you need to do

Modify the `InputCoordinate` subroutine in the `Simulation` class so that, if a coordinate outside the range defined by the landscape size is input, the error message "Coordinate is outside of landscape, please try again." is displayed and the user is forced to re-input the coordinate.

To achieve full marks for this question, the `InputCoordinate` subroutine should work correctly for any landscape size, not just the default size of 15.

Test

Test your changes work by running the **Skeleton Program** and selecting the following options:

- "1. Run simulation with default settings"
- "3. Inspect fox"

Then input these three x coordinates for the location of the fox to inspect:

- -1
- 15
- 0

Evidence that you need to provide

- (i) Your PROGRAM SOURCE CODE for the amended subroutine `InputCoordinate`.

EXAM PAPERS PRACTICE

(4)

- (ii) SCREEN CAPTURE(S) for the described test.

Ensure that in your SCREEN CAPTURE(S) it can be seen that the x coordinates -1 and 15 are rejected **and** that the x coordinate 0 is accepted, **and** that after the 0 is input the **Skeleton Program** advances to ask the user to input the y coordinate.

(1)

- (b) The simulation is to be made more realistic by increasing the probability that a rabbit will die as a result of other causes, such as disease or injury, as the rabbit ages.

The default probability of death by another cause for a rabbit is 0.05.

- The probability of a male rabbit dying by another cause should increase by a factor of 50% after every time period.
- The probability of a female rabbit dying by another cause should remain constant until the rabbit reaches the age of 2. At the age of 2, and after every time period beyond this, the probability of a female rabbit dying by another cause should increase by 0.05.

Table 1 below summarises the probability of death by other causes for a rabbit of each gender, up to the age of 5. The probabilities will continue to increase beyond this age.

Table 1

Age	Probability of death by other causes	
	Male (2dp)	Female
0	0.05	0.05
1	0.08	0.05
2	0.11	0.1
3	0.17	0.15
4	0.25	0.2
5	0.38	0.25

What you need to do

Create a new subroutine, `CalculateNewAge`, in the `Rabbit` class, that overrides the `CalculateNewAge` subroutine in the `Animal` class.

The new `CalculateNewAge` subroutine in the `Rabbit` class should recalculate the probability of death for a rabbit as the rabbit ages. The subroutine should also call the subroutine that it has overridden in the `Animal` class to ensure that the standard ageing process for a rabbit continues to be carried out as well.

Test

Check that the changes you have made work by conducting the following test:

- Select option "1. Run simulation with default settings" from the main menu.
- Then select option "2. Advance to next time period hiding detail" **twice**, to advance the simulation to time period 2.
- Then select option "4. Inspect warren" and enter the x coordinate 1 and the y coordinate 1.
- When asked "View individual rabbits (y/n)?" enter `y`.

Evidence that you need to provide

- (i) Your PROGRAM SOURCE CODE for the new subroutine `CalculateNewAge` from the `Rabbit` class.

(5)

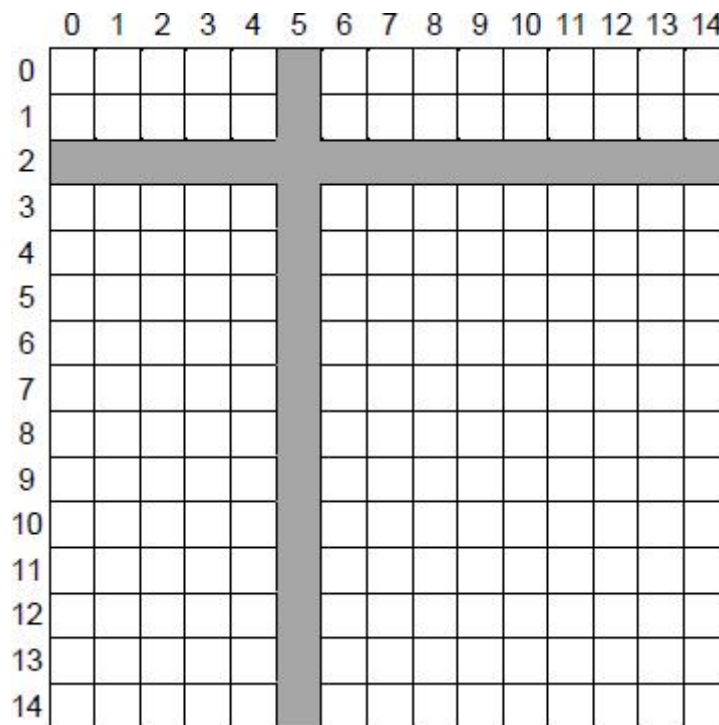
- (ii) SCREEN CAPTURE(S) for the described test.

Your SCREEN CAPTURE(S) must clearly show the probability of death by other causes of both a male and a female rabbit of age 2. SCREEN CAPTURE(S) do **not** need to show the options that you have selected or the probability of death by other causes for rabbits of other ages.

(1)

- (c) The simulation is to be extended to represent the landscape that the animals live in. Most of the landscape will be land, but two rivers will run through it. The locations of the rivers are shaded in **Figure 3**.

Figure 3



Each of the individual locations, eg (12, 7), within the landscape will be assigned to be an area of either land or river.

What you need to do

Task 1

Modify the `Location` class so that it can store a representation of the type of terrain at the location. This representation should be as a character, with "L" representing land and "R" representing river.

Task 2

Modify the constructor subroutine of the `Location` class so that when a location is created, the constructor is passed the type of terrain that the location will be and this is stored appropriately.

Task 3

Modify the `CreateLandscapeAndAnimals` subroutine in the `Simulation` class so that when the landscape is created the appropriate type of terrain, as shown in **Figure 3**, is stored in each location. The terrain should be represented as a character, with "L" representing land and "R" representing river.

Task 4

Modify the `DrawLandscape` subroutine in the `Simulation` class so that the correct type of terrain at each location is displayed when the landscape is drawn.

Figure 4 shows one example of how the landscape could be drawn, with a letter "L" indicating that a location contains land, and a letter "R" indicating that a location contains part of a river. However, you are free to indicate the type of terrain at a location in any way that you choose, so long as this is clear to the user.

Figure 4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	L	L	L	L	L	R	L	L	L	L	L	L	L	L	L
1	L	38	L	L	L	R	FL	L	L	L	L	L	L	L	L
2	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
3	L	L	L	L	L	R	L	L	L	L	52	L	L	L	L
4	L	L	L	L	L	R	L	L	L	L	L	L	FL	67	L
5	L	L	L	L	L	R	L	L	L	L	L	L	L	L	L
6	L	L	L	L	L	R	L	L	L	FL	L	L	L	L	L
7	L	L	L	L	L	R	L	L	L	L	20	L	L	L	L
8	L	L	80	L	L	R	L	L	L	L	L	L	L	L	L
9	L	L	L	L	L	R	L	L	L	L	L	L	L	L	L
10	L	L	FL	L	L	R	L	L	L	L	L	L	L	L	L
11	L	L	L	L	L	R	L	L	L	L	L	L	L	L	L
12	L	L	L	L	L	R	L	L	L	L	L	L	L	L	L
13	L	L	L	L	L	R	L	L	L	L	L	FL	L	L	L
14	L	L	L	L	L	R	L	L	L	L	L	L	L	L	L

Task 5

Modify the `CreateNewWarren` and `CreateNewFox` subroutines in the `Simulation` class so that warrens and foxes cannot be created in locations that are part of a river.

Test

Check that the changes you have made in Tasks 1 to 4 (**not** Task 5) work by conducting the following test:

- Select option “1. Run simulation with default settings” from the main menu.

Evidence that you need to provide

- Your PROGRAM SOURCE CODE for the whole of the `Location` class, including the constructor subroutine. (3)
 - Your PROGRAM SOURCE CODE for the amended `CreateLandscapeAndAnimals` subroutine from the `Simulation` class. (3)
 - Your PROGRAM SOURCE CODE for the amended `DrawLandscape` subroutine from the `Simulation` class. (2)
 - Your PROGRAM SOURCE CODE for the amended `CreateNewWarren` and `CreateNewFox` subroutines from the `Simulation` class. (3)
 - SCREEN CAPTURE(S) for the described test, showing the correct type of territory in each location on the landscape. (1)
- (d) The landscape affects the foxes’ ability to eat the rabbits. Foxes do not like to swim, so will not cross the rivers on the landscape to eat. If a river lies between a fox and a warren, the fox will not eat any rabbits in the warren, even if it is near enough for it to do so.

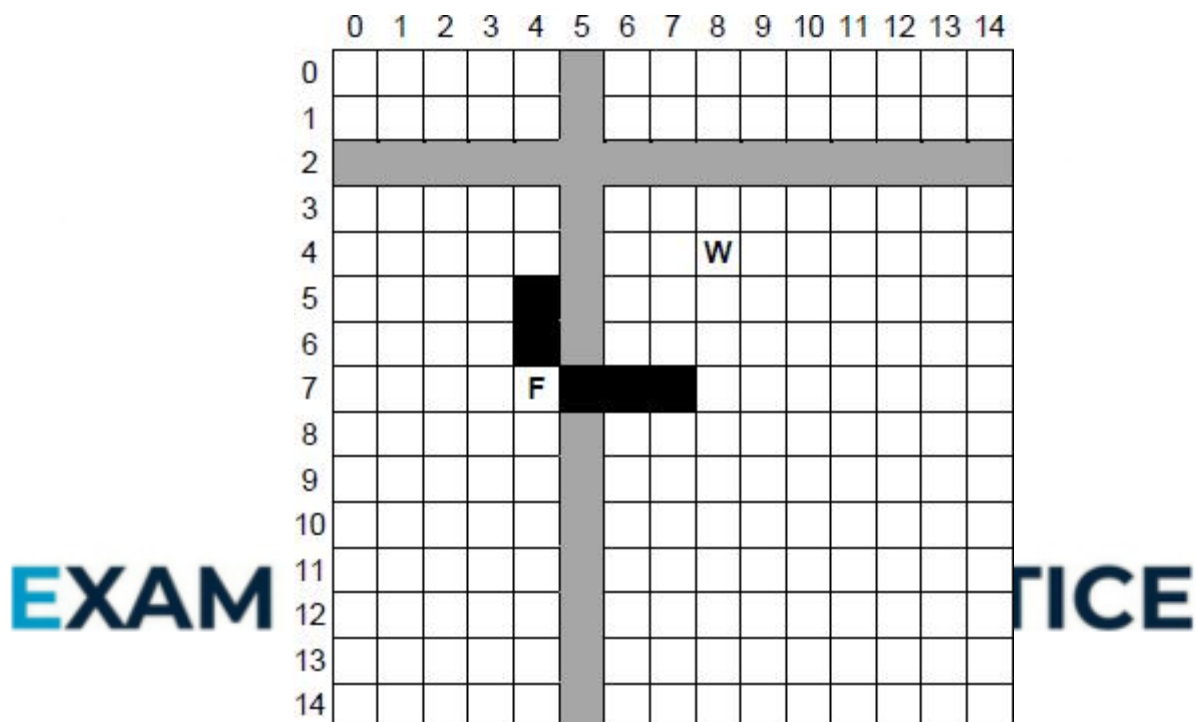
As the rivers only run horizontally and vertically, and extend from one side of the landscape to the other, a simple way to check if reaching a warren would require a

fox to cross a river is to:

- Calculate the coordinates of all of the locations between the fox and the warren in a horizontal line, level with the fox.
- Calculate the coordinates of all of the locations between the fox and the warren in a vertical line, level with the fox.
- If any of the locations horizontally or vertically between the fox and the warren contain a river, then the fox will not eat any of the rabbits in the warren as the fox's path to the warren crosses a river.

Figure 5 shows the locations that would need to be checked to see if fox **F** could eat any rabbits in warren **W**. The locations that need to be checked are shown in black and the rivers are shown in grey. As location (5, 7) contains part of a river, the fox would not eat any rabbits in this warren.

Figure 5



If you have not been able to fully complete part (c), you will still be able to get most of the marks for this question if you can correctly compute the coordinates of the locations that would need to be checked to see if a river was present.

To get full marks for this question, your solution must work regardless of whether a warren is above, below, to the left or to the right of a fox.

What you need to do

Task 1

Create a new subroutine `CheckIfPathCrossesRiver`, in the `Simulation` class, that takes the coordinates of two locations in the landscape **and** checks if there is a river between them.

Task 2

Modify the `FoxesEatRabbitsInWarren` subroutine in the `Simulation` class so that

it calls the `CheckIfPathCrossesRiver` subroutine, **and** ensures that if there is a river between a fox and a warren then the fox will not eat any rabbits from the warren.

Test

Check that the changes you have made work by conducting the following test:

- Select option “1. Run simulation with default settings” from the main menu.
- Then select option “1. Advance to next time period showing detail”.

When the test is conducted, no rabbits in the warren at (1, 1) should be eaten as it is bounded by rivers on all sides.

Evidence that you need to provide

- (i) Your PROGRAM SOURCE CODE for the new subroutine `CheckIfPathCrossesRiver` from the `Simulation` class.

(9)

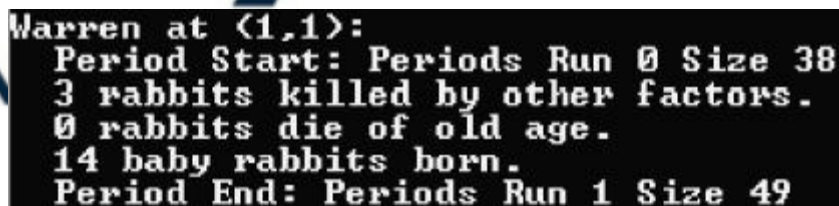
- (ii) Your PROGRAM SOURCE CODE for the amended subroutine `FoxesEatRabbitsInWarren` from the `Simulation` class.

(2)

- (ii) SCREEN CAPTURE(S) for the described test.

Your SCREEN CAPTURE(S) only needs to show what happens in the warren at location (1, 1) when the simulation advances to the next time period. It should contain similar information to Figure 6 below, but the exact number of rabbits killed, dying of old age and other details may differ owing to the random nature of parts of the simulation.

Figure 6



```
Warren at (1,1):
Period Start: Periods Run 0 Size 38
3 rabbits killed by other factors.
0 rabbits die of old age.
14 baby rabbits born.
Period End: Periods Run 1 Size 49
```

(1)

(Total 35 marks)

Q4.

An object-oriented program is being written to store details of the hardware devices that are connected to a computer network in a college. This will be used by the network manager to perform an audit of the equipment that they manage.

Two different types of devices are connected to the network. They are printers and computers. The computers are categorised as being laptops, desktops or servers.

A class **Device** has been created and two subclasses, **Printer** and **Computer** are to be developed. The **Computer** class will have three subclasses: **Laptop**, **Desktop** and **Server**.

(a) Draw an inheritance diagram for the six classes.

(3)

The **Device** class has data fields **MACAddress**, **DeviceName** and **Location**.

The class definition for **Device** is:

```
Device = Class
    Public
        Procedure AddDevice
        Function GetMACAddress
        Function GetDeviceName
        Function GetLocation
    Private
        MACAddress: String
        DeviceName: String
        Location: String
    End
```

AddDevice is the constructor function for the **Device** class and is called whenever a new **Device** object is created.

The **Computer** class has the following additional data fields:

- **ProcessorName**: Stores the name of the company that manufactured the processor installed in the computer.
- **RAMCapacity**: Stores the capacity of the RAM installed in the computer, in gigabytes, eg 16.
- **HDDCapacity**: Stores the capacity of the Hard Disk Drive installed in the computer, in gigabytes, eg 512.

(b) Write the class definition for **Computer**.

(4)

- (c) The **Laptop** class has the additional data field **BluetoothInstalled**. This field will indicate whether or not the laptop is fitted with a Bluetooth module.

Write the class definition for **Laptop**.

(2)

(Total 9 marks)

Q5.

A computer games programmer is writing a game. One aspect of the game involves a character who can carry various items, such as a bag of seeds, and an axe, around with her. The list of items that the character is currently carrying will be stored as a linked list of items of the `String` data type. The list is stored in no particular order.

The game is being developed using object-oriented programming. The **LinkedList** class will be used to store that list of items.

The class definition for the **LinkedList** class is:

```
LinkedList
= Class
  Public
    Procedure CreateList
    Procedure DestroyList
    Procedure AddItem(NewItem: String)
    Procedure DeleteItem(DelItem: String)
    Function ContainsItem(SearchItem: String): Boolean
    Function IsEmpty: Boolean
  Private
    Start: Pointer
    Current: Pointer
    Previous: Pointer
End
```


- (a) Creating a class such as the **LinkedList** class, that can be used by other parts of a much bigger program, is a form of abstraction.

Explain why the **LinkedList** class is a form of abstraction.

(1)

- (b) Explain why the functions and procedures, such as `AddItem` have been declared to be `Public` whilst the data items such as `Start` have been declared as `Private`.

(2)

- (c) Write a pseudo-code algorithm for the `DeleteItem` operation.

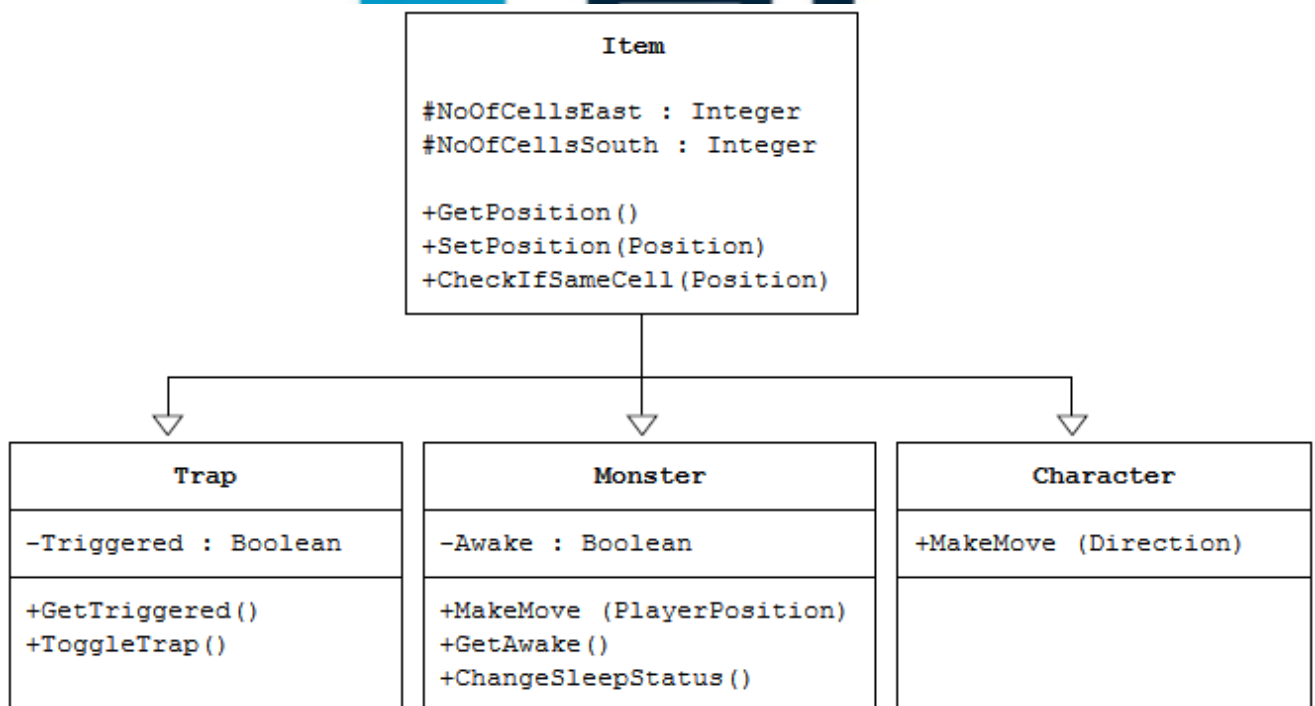
You may assume that:

- `Start` is a pointer to the memory location of the first item in the linked list
- The variable `DelItem`, which will be passed to the `DeleteItem` operation as a parameter, is a `String` that contains the name of the item to delete, exactly as the name appears in the linked list
- The linked list is not empty, and does contain the item to be deleted
- For each item stored in the list, two fields are stored, which are called `DataValue` and `Next`. The `DataValue` is the name of the item that is stored and `Next` is a pointer to the memory location of the next item in the list. To access the values stored in these fields at a particular memory location, such as `Current`, the instructions `Current.DataValue` and `Current.Next` would be used
- An operation called `Release` is provided by the operating system that will make a specified memory location that is no longer required available for re-use
- You should make use of the data items `Current` and `Previous`, both of which are pointers, when searching the list to locate the item that is to be deleted.

(8)
(Total 11 marks)

Q6.

The class diagram below is an attempt to represent the relationships between some of the classes in the MONSTER! Game.



(a) Explain what errors have been made in the class diagram.

_____ (2)

(b) Give an example of instantiation from the **Skeleton Program**.

_____ (1)

(c) State the name of an identifier for an array variable.

_____ (1)

(d) State the name of an identifier for a subclass.

_____ (1)

(e) State the name of an identifier for a variable that is used to store a whole number.

_____ (1)

(f) State the name of an identifier for a class that uses composition.

EXAM PAPERS PRACTICE (1)

(g) Look at the `GetNewRandomPosition` subroutine in the `Game` class in the **Skeleton Program**.

Explain why the generation of a random position needs to be inside a repetition structure.

_____ (1)

(h) Look at the `Game` class in the **Skeleton Program**.

Why has a named constant been used instead of the numeric value 5?

(2)

- (i) Describe the changes that would need to be made to the `Game` class to add a third trap to the cavern. The third trap should have exactly the same functionality as the other two traps. You do **not** need to describe the changes that would need to be made to the `SetUpGame` subroutine.

(2)

(Total 12 marks)

Q7.

- (a) This question refers to the subroutines `CheckValidMove` and `Play` in the `Game` class.

The **Skeleton Program** currently does not make all the checks needed to ensure that the move entered by a player is an allowed move. It should not be possible to make a move that takes a player outside the 7 × 5 cavern grid.

The **Skeleton Program** needs to be adapted so that it prevents a player from moving west if they are at the western end of the cavern.

The subroutine `CheckValidMove` needs to be adapted so that it returns a value of `FALSE` if a player attempts to move west when they are at the western end of the cavern.

The subroutine `Play` needs to be adapted so that it displays an error message to the user if an illegal move is entered. The message should state "That is not a valid move, please try again".

Evidence that you need to provide

- (i) Your amended PROGRAM SOURCE CODE for the subroutine `CheckValidMove`. (3)
- (ii) Your amended PROGRAM SOURCE CODE for the subroutine `Play`. (2)
- (iii) SCREEN CAPTURE(S) for a test run showing a player trying to move west when they are at the western end of the cave. (1)
- (b) This question will extend the functionality of the game.

The game is to be altered so that there is a new type of enemy: a sleepy enemy. A

sleepy enemy is exactly the same as a normal enemy, except that after making four moves it falls asleep again.

Task 1

Create a new class called `SleepyEnemy` that inherits from the `Enemy` class.

Task 2

Create a new integer attribute in the `SleepyEnemy` class called `MovesTillSleep`.

Task 3

Create a new public subroutine in the `SleepyEnemy` class called `ChangeSleepStatus`. This subroutine should override the `ChangeSleepStatus` subroutine from the `Enemy` class. The value of `MovesTillSleep` should be set to 4 in this subroutine.

Task 4

Create a new public subroutine in the `SleepyEnemy` class called `MakeMove`. This subroutine should override the `MakeMove` subroutine from the `Enemy` class. When called this subroutine should reduce the value of `MovesTillSleep` by 1 and then send the monster to sleep if `MovesTillSleep` has become equal to 0.

Task 5

Modify the `Game` class so that the `Monster` object is of type `SleepyEnemy` (instead of `Enemy`).

Task 6

Check that the changes you have made work by conducting the following test:

- play the training game
- move east
- move east
- move south.

Evidence that you need to provide

(i) Your PROGRAM SOURCE CODE for the new `SleepyEnemy` class.

(8)

(ii) SCREEN CAPTURE(S) showing the requested test.

(2)

- (c) This question refers to the `Game` and `Character` classes and will extend the functionality of the game.

The game should be altered so that once per game the player can shoot an arrow instead of making a move in the cavern. The arrow travels in a straight line, in a direction of the player's choice, from the cell the player is in to the edge of the cavern. If the arrow hits the monster then the player wins the game and a message saying that they have shot the monster should be displayed.

For this question you are **only** required to extend the program so that it checks if the monster is hit by the arrow when the user chooses to shoot an arrow northwards. However, the user should be able to select any of the four possible directions.

In the diagram below, the two shaded cells show the cells which, if the monster is in one of them, would result in the player winning the game, as long as the player is in the cell five to the east and three to the south and chooses to shoot an arrow

northwards.

				*		

Task 1

Modify the `DisplayMoveOptions` subroutine in the `Game` class so that the option to enter A to shoot an arrow is added to the menu.

Task 2

Create a new Boolean attribute called `HasArrow` in the `Character` class.

The value of `HasArrow` should be set to `True` when a new object of class `Character` is instantiated.

Task 3

Create a new public subroutine called `GetHasArrow` in the `Character` class that returns the value of the `HasArrow` attribute to the calling routine.

Task 4

Modify the `CheckValidMove` subroutine in the `Game` class so that:

- it is a valid move if A is selected and the player does have an arrow
- it is not a valid move if A is selected and the player does not have an arrow.

Task 5

Create a new public subroutine called `GetArrowDirection` in the `Character` class.

This subroutine should return a character to the calling routine.

The user should be asked in which direction they would like to shoot an arrow (N, S, E or W) and the value entered by the user should be returned to the calling routine.

If an invalid direction is entered then the user should be repeatedly asked to enter a new direction, until a valid direction is entered.

The value of `HasArrow` should then be changed to `FALSE`.

Task 6

Modify the `Play` subroutine in the `Game` class so that if the move chosen by the user is not M it then checks if the move chosen is A.

If the move chosen was A, then there should be a call to the player's `GetArrowDirection` subroutine. If the user chooses a direction of N then the program should check to see if the monster is in one of the squares directly north of the player's current position. If it is then a message saying "You have shot the monster and it cannot stop you finding the flask" should be displayed. The value of `FlaskFound` should then be set to `TRUE`.

After the arrow has been shot, if the monster is still alive and awake, it is now the monster's turn to move, the player should remain in the same cell as they were in before the arrow was shot.

There is **no** need to write any code that checks if the monster has been shot when the player chooses to shoot either to the east, to the west or to the south.

Task 7: test 1

Test that the changes you have made work by conducting the following test:

- play the training game
- shoot an arrow
- choose a direction of N for the arrow.

Task 8: test 2

Test that the changes you have made work by conducting the following test:

- play the training game
- move east
- shoot an arrow
- choose a direction of N for the arrow
- shoot an arrow.

Evidence that you need to provide

- (i) Your amended PROGRAM SOURCE CODE for the subroutine `DisplayMoveOptions`.

(1)

- (ii) Your amended PROGRAM SOURCE CODE for the subroutine `CheckValidMove`.

(2)

- (iii) Your amended PROGRAM SOURCE CODE for the class `Character`.

(8)

- (iv) Your amended PROGRAM SOURCE CODE for the subroutine `Play`.

(6)

- (v) SCREEN CAPTURE(S) showing the results of **Test 1**.

(1)

- (vi) SCREEN CAPTURE(S) showing the results of **Test 2**.

(1)

(Total 35 marks)

Q8.

Figure 1 shows the structure of an example machine code instruction, taken from the instruction set of a particular processor.

Figure 1

Opcode							Operand(s)							
Basic Machine Operation					Addressing Mode									
0	1	1	0	1	0	1	0	0	1	0	1	0	1	1

- (a) How many different basic machine operations could be supported by the instruction set of the processor used in the example in **Figure 1**?

(1)

Figure 2 shows an assembly language program together with the contents of a section of the main memory of the computer that the program will be executed on.

The assembly language instruction set that has been used to write the program is listed in **Table 1**. The lines of the assembly language program have been numbered to help you answer question parts (b) to (d)

Figure 2

Line	Command	Memory Address (in decimal)	Main Memory Contents (in decimal)
1	LDR R2, #100		
2	LDR R3, 101		
3	ADD R2, R2, R3	100	23
4	LSL R3, R2, #1	101	10
5	HALT	102	62
		103	18

- (b) What value will be stored in register R2 immediately after the command in line 1 has been executed?

(1)

- (c) What value will be stored in register R2 immediately after the program has executed the commands from line 1 through to line 3?

(1)

- (d) What value will be stored in register R3 after the complete program has finished executing?

Table 1

LDR Rd, <memory ref>	Load the value stored in the memory location specified by <memory ref> into register d.
STR Rd, <memory ref>	Store the value that is in register d into the memory location specified by <memory ref>.
ADD Rd, Rn, <operand2>	Add the value specified in <operand2> to the value in register n and store the result in register d.
SUB Rd, Rn, <operand2>	Subtract the value specified by <operand2> from the value in register n and store the result in register d.
MOV Rd, <operand2>	Copy the value specified by <operand2> into register d.
CMP Rn, <operand2>	Compare the value stored in register n with the value specified by <operand2>.
B <label>	Always branch to the instruction at position <label> in the program.
B<condition> <label>	<p>Conditionally branch to the instruction at position <label> in the program if the last comparison met the criteria specified by the <condition>. Possible values for <condition> and their meaning are:</p> <ul style="list-style-type: none"> • EQ: Equal to. • NE: Not equal to. • GT: Greater than. • LT: Less than.
AND Rd, Rn, <operand2>	Perform a bitwise logical AND operation between the value in register n and the value specified by <operand2> and store the result in register d.
ORR Rd, Rn, <operand2>	Perform a bitwise logical OR operation between the value in register n and the value specified by <operand2> and store the result in register d.
EOR Rd, Rn, <operand2>	Perform a bitwise logical exclusive or (XOR) operation between the value in register n and the value specified by <operand2> and store the result in register d.
MVN Rd, <operand2>	Perform a bitwise logical NOT operation on the value specified by <operand2> and store the result in register d.
LSL Rd, Rn, <operand2>	Logically shift left the value stored in register n by the number of bits specified by <operand2> and store the result in register d.
LSR Rd, Rn, <operand2>	Logically shift right the value stored in register n by the number of bits specified by <operand2> and store the result in register d.
HALT	Stops the execution of the program.

Interpretation of <operand2>

<operand2> can be interpreted in two different ways, depending upon whether the first symbol is a # or an R:

- # - use the decimal value specified after the #, eg #25 means use the decimal value 25.
- R_m - use the value stored in register _m, eg R6 means use the value stored in register 6.

The available general purpose registers that the programmer can use are numbered 0 to 12.

Programs written in a high-level language can be compiled or interpreted.

Companies that develop computer programs to sell usually compile the final version of a program before distributing it to customers.

- (e) Explain why the final version of a computer program is usually translated using a compiler.

(2)

- (f) The JavaScript programming language can be used to write programs that are executed in a web browser on any Internet user's computer.

Explain why programs written in the JavaScript language, to be executed in a web browser, are interpreted rather than compiled.

EXAM PAPERS PRACTICE

(2)

(Total 8 marks)

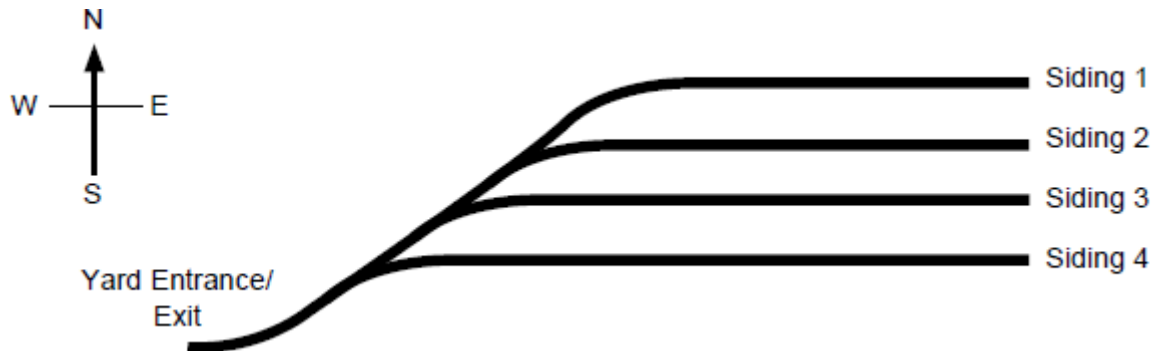
Q9.

A computer program is being developed that will simulate the organisation of wagons (trucks) in a railway shunting yard. The simulation will be based on a model developed by the shunting yard manager and a systems analyst.

- (a) In the context of simulation, explain what a model is.

(1)

The diagram below shows the layout of the railway yard. The wagons enter the yard and are pushed into an appropriate siding, depending upon their final destination. Each siding can hold many wagons. Wagons can **only** enter and leave a siding using the Yard Entrance / Exit at the west.



Wagons will be represented as objects in an object-oriented programming language.

Each of the sidings will be represented as a stack data structure.

(b) Explain why a stack data structure is appropriate for representing a siding.

(2)

(c) The computer program developer intends to implement a stack by using a fixed length array of 30 wagon objects, named `StackArray`, with indices running from 1 to 30. An integer variable `TopOfStackPointer`, that will store the array index of the item at the top of the stack, will also be used. The first object stored in the array will be stored at index 1, the second at index 2 and so on. `TopOfStackPointer` will be initialised to 0.

Write a pseudo-code algorithm for the `Pop` operation to remove a value from the stack and store it in a wagon object variable named `CurrentWagon`.

Your algorithm should cope appropriately with any potential errors that might occur.

(4)

- (d) Wagons come in two different categories: open wagons (without a roof) and closed wagons (with a roof). Closed wagons can be either refrigerated or non-refrigerated.

In an object-oriented programming language, five classes are to be created, named **Wagon**, **OpenWagon**, **ClosedWagon**, **RefrigeratedWagon** and **NonRefrigeratedWagon**.

Draw an inheritance diagram for the five classes.

(3)

- (e) The **Wagon** class has data fields **OwnerName**, **Weight** and **NumberOfWheels**.

The class definition for **Wagon** is:

```
Wagon = Class
    Public
        Procedure CreateWagon
        Function GetOwnerName
        Function GetWeight
        Function GetNumberOfWheels
    Private
        OwnerName: String
        Weight: Real
        NumberOfWheels: Integer
End
```

The **ClosedWagon** class has the following additional data fields:

- **Height:** The height of the wagon in metres, which could be a non-integer number
- **NumberOfDoors:** The number of doors that can be used to access the wagon
- **SuitableForFoodstuffs:** A true or false value that indicates if it is safe to carry food in the wagon or not.

Write the class definition for **ClosedWagon**.

You should include the necessary data fields and any additional procedures or functions that the class would require in your definition.

(4)
(Total 14 marks)

An event-driven, object-oriented programming language lets the programmer create a Graphical User Interface (GUI) from components such as forms and buttons. The components of the GUI are implemented using a class hierarchy and inheritance.

(1)

(b) One GUI component is a **Selector**. Selectors come in two different types: **ComboBox** and **ListBox**.

Page 29 of 50

Draw an inheritance diagram for the classes: Selector, ComboBox, ListBox, SingleSelectionListBox and MultipleSelectionListBox.

(3)

(c) The **Selector** class has data fields **Items** and **NumberOfItemsInList**:

- **Items**: an array that stores the list of strings that will appear in the selector.
- **NumberOfItemsInList**: a number that indicates how many items there are in the selector.

It also has a procedure that the programmer can call to add an item to the list of strings (**AddItemToList**) and a procedure that is called by the operating system whenever the user selects an item from the list (**SelectItemFromList**).

The **Selector** class does not include a procedure to display the items in the list as the way items are displayed is different for each type of selector.

The class definition for **Selector** is:

```
Selector = Class
    Public
        Procedure AddItemToList
        Procedure SelectItemFromList
    Private
        Items: Array of String
        NumberOfItemsInList: Integer
End
```

EXAM PAPERS PRACTICE

A class is to be created for the **ComboBox** type of selector.

The **ComboBox** class needs the following additional data fields:

- **TextTyped**: Stores the characters that have been typed by the user if they have made their input by typing rather than picking an option from the list.
- **SelectedItemNumber**: Stores the position in the list of the item that has been selected by the user, if one has been selected.
- **AllowNonListInputs**: A True or False value that indicates whether the user should be allowed to type in text that is not one of the items in the list.

The class will need to implement the operation of selecting an item from the list differently from the way the **Selector** class implements this operation, but the operation of adding an item to the list will be implemented in the same way by both of these classes.

The class must provide subroutines to:

- display the combo box
- respond to the operating system's notification of a key press
- return the text that has been typed in

- Write the class definition for the **ComboBox** class.



(Total 9 marks)

An object-oriented program is being written to store details of the hardware devices that are connected to a computer network in a college. This will be used by the network manager to perform an audit of the equipment that the college owns.

A class **Device** has been created and two subclasses, **Printer** and **Computer** are to be developed. The **Computer** class will have three subclasses: **Laptop**, **Desktop** and **Server**.

- Page 31 of 50

- (b) The **Device** class has data fields **MACAddress**, **DeviceName** and **Location**.

The class definition for **Device** is:

```
Device = Class
    Public
        Procedure AddDevice
        Function GetMACAddress
        Function GetDeviceName
        Function GetLocation
    Private
        MACAddress: String
        DeviceName: String
        Location: String
    End
```

The **Computer** class has the following additional data fields:

- **ProcessorName**: Stores the name of the company that manufactured the processor.
- **RAMCapacity**: Stores the capacity of the RAM installed in the computer, in gigabytes.
- **HDDCapacity**: Stores the capacity of the Hard Disk Drive installed in the computer, in gigabytes.

EXAM PAPERS PRACTICE

Write the class definition for **Computer**.

(4)

- (c) The **Laptop** class has the additional data field **BluetoothInstalled**. This field will indicate whether or not the laptop is fitted with a Bluetooth module.

Write the class definition for **Laptop**.

(2)

- (d) Explain what Bluetooth is and give an example of a task for which a laptop user might use Bluetooth.

What Bluetooth is:

EXAM PAPERS PRACTICE

(2)

Example use:

(1)

(Total 12 marks)

Q12.

State **three** features of well-written program code that help to make it understandable without the need to include lots of comments.

(Total 3 marks)

Q13.

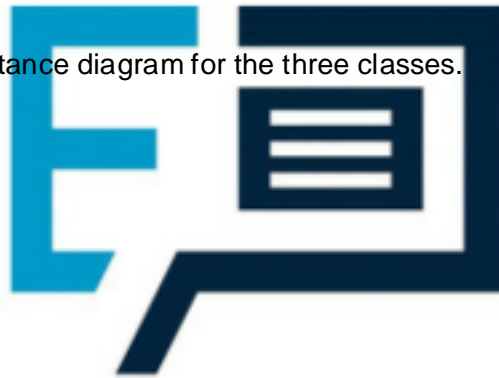
An object-oriented program is being written to store details of and play digital media files that are stored on a computer. A class **MediaFile** has been created and two subclasses, **VideoFile** and **MusicFile** are to be developed.

The classes **VideoFile** and **MusicFile** are related to **MediaFile** by single inheritance.

- (a) Explain what is meant by *inheritance*.

(1)

- (b) Draw an inheritance diagram for the three classes.



EXAM PAPERS PRACTICE

(2)

- (c) One important feature of an object-oriented programming language is the facility to override methods (functions and procedures).

Explain what is meant by *overriding* when writing programs that involve inheritance.

(2)

- (d) The **MediaFile** class has data fields **Title** and **Duration**.

The class definition for **MediaFile** is:

```
MediaFile = Class
    Public
        Procedure PlayFile
        Function GetTitle
        Function GetDuration
    Private
        Title : String
        Duration : Real
    End
```

Note that the class does not have procedures to set the values of the variables as these are read automatically from data stored within the actual media file.

The **MusicFile** class has the following additional data fields:

- **Artist**: Stores the name of the band or singer that recorded the music.
- **SampleRate**: Stores the rate at which the music has been sampled.
- **BitDepth**: Stores the number of bits in which each sampled value is represented.

Write the class definition for **MusicFile**.

EXAM PAPERS PRACTICE

(4)

(Total 9 marks)

Q14.

A library system uses three classes, **BookCopy**, **Borrower** and **Loan**. A **BookCopy** object represents a book, a **Borrower** object represents someone who borrows books and a **Loan** object represents the loan of a single **BookCopy** to a **Borrower**.

- (a) Draw a class diagram to represent the relationships between these classes.

(3)

- (b) The Borrower class has data fields Name and Address. The class definition for Borrower is

```
Borrower = Class
    Public
        Procedure AddNewBorrower
        Procedure AmendBorrowerDetails
        Procedure GetBorrowerDetails
    Private
        Name : String
        Address : String
    End
```

The BookCopy class has data fields Title, Author, OnLoan and ISBN. The class definition for BookCopy is

```
BookCopy = Class
    Public
        Procedure AddNewBookCopy
        Procedure ChangeLoanStatus
        Procedure GetBookDetails
    Private
        Title : String
        Author : String
        OnLoan : Boolean
        ISBN : String
    End
```

The Loan class needs operations (methods) to create a loan, delete a loan and get loan details. The data fields are the person, the book loaned, the date of the loan and the date of return.

Write the class definition for the Loan class.

(4)

- (c) The library has decided to introduce short-loan books in addition to standard-loan books. How would you modify the BookCopy class to allow for this change?

(2)

(Total 9 marks)

Q15.

- (a) Writing program code requires the programmer to use identifiers for variables and procedures.

- (i) State **two** other uses for identifiers.

1.

2.

(2)

- (ii) Most programming languages impose restrictions or rules about what is and is not allowed for identifier names. State **one** such rule.

EXAM PAPERS PRACTICE

(1)

- (b) Program code is often written with the use of procedures. Describe **one** reason why a programmer would decide to use procedures.

(1)

- (c) A programmer-written function **SearchThisArray** is defined as follows.

```
SearchThisArray(ThisArray : Array[1..10] Of String;
                ThisString : String) : Integer ;

The function searches the array ThisArray for the value ThisString.

If an exact match is found, the function returns the index position
in ThisArray.
```

If not found, the function returns -1.
 If the function's arguments, ThisArray and ThisString are illegally formed, the function returns -2

The function is used in a program with the statements shown below and uses the data shown in the Customer array in the figure below.

Index (Subscript)	Customer
[1]	Weeks
[2]	Adamson
[3]	Patel
[4]	Berkovic
[5]	Ince
[6]	Neale
[7]	Williamson
[8]	Collins
[9]	Davis
[10]	Beckham

What is the value returned to variable Result in each case?

(i) Result := SearchThisArray(Customer, 'Beckham')

Value of Result _____

(1)

(ii) Result := SearchThisArray(Customer, 'Williams')

Value of Result _____

(1)

(Total 6 marks)

Q16.

(a) In object-oriented programming, what is meant by aggregation?

(1)

(b) An object-oriented program is required to handle details of items of furniture that are for sale. The furniture sold includes dining suites. A dining suite consists of a table and a number of chairs.

Some fields required for the suites are

TableType
ChairType
NumberOfChairs

A method required for the suites is

DisplayDetails

Some fields required for the tables are

TableType
Size
Colour

Some fields required for the chairs are

ChairType
Colour

(i) Draw a **class diagram** of these classes, Suite, Table and Chair.



(2)

EXAM PAPERS PRACTICE

(ii) Write class definitions for Chair, Table and Suite.

(8)

(Total 11 marks)

Q17.

- (a) Well constructed programs use a structured approach for the design and coding stages.

One practical way in which the programmer will use a structured approach to programming is the use of subroutines (procedures/functions). Give **three** other ways.

1. _____

2. _____
3. _____

(3)

- (b) A program is to be written which calculates the hourly pay rate for an employee. The calculation is based on the number of complete years the employee has worked for the firm (e.g. 3 years). All employees get a basic £7.88 per hour. For each year worked, up to a maximum of 5 years only, an additional £0.65 is added to the basic hourly rate.

The algorithm for this program is as follows:

1. Enter the surname
2. Enter the number of years of service
3. Calculate the employee's pay rate
4. Output the surname and pay rate

- (i) Complete the table showing **three** variable identifiers and their data types you would use for this problem.

Variable Identifier	Data Type

(3)

- (ii) The detail for step 3 in the algorithm is broken down into more detail as follows:

3.1 If the number of years of service value is over 5, then change the value stored to 5

3.2 Calculate the employee's pay rate

Write pseudo-code for these two steps using the appropriate identifiers from the table.

- 3.1 _____
- _____
- 3.2 _____
- _____

(3)

(Total 9 marks)

Q18.

Many programs executed within a Graphical User Interface (GUI) environment are *object-oriented* and *event-driven*.

- (a) Give an example of an event in this context.

_____ (1)

- (b) Describe how event-driven programs differ from non event-driven programs.

_____ (2)

- (c) List **two** features of an object.

1. _____
2. _____ (2)

- (d) Name an object that might be part of a GUI.

_____ (1)
(Total 6 marks)

Q19.

For an object-oriented program to store and retrieve details of a company's vehicles, a Vehicle class is needed. **two** subclasses have been identified: **Car** and **Van**, which have inheritance relationships with class **Vehicle**.

EXAM PAPERS PRACTICE

- (a) Draw an inheritance diagram for these classes.

(2)

- (b) The Vehicle class has data fields RegistrationNumber, Make, Colour. The class definition for Vehicle is

```

Vehicle = Class
Public
    Procedure SetVehicleDetails
    Function GetRegistrationNumber
    Function GetMake
    Function GetColour
Private
    RegistrationNumber : String
    Make : String
    Colour : String
End

```

While preserving the private status of the Colour field, what modification would you make to this class definition in order to allow the colour of the vehicle to be changed?

(2)

(c) The Van class has additional private data fields:

- Capacity that represents the weight that can be carried in kilograms;
- TailLift that represents whether the van has a tail lift or not.

Write the class definition for **Van**.

EXAM PAPERS PRACTICE

(6)

(Total 10 marks)

Q20.

(a) In object-oriented programming, what is meant by inheritance?

(1)

(b) An object-oriented program is required to handle details of a lending library's books and CDs.

Some fields required for the books are:

Title,
Author,
ISBN,
OnLoan,
DateAcquired.

Some fields required for the CDs are:

Title,
Artist,
PlayingTime,
OnLoan,
DateAcquired.

Some methods required are:

SetLoan,
DisplayDetails

This could be implemented by declaring two separate classes Book and CD. This would result in a lot of repetitive code. Making use of inheritance, **write** class definitions for one superclass **StockItem** and two subclasses **Book** and **CD**.



(7)

(Total 8 marks)

Q21.

- (a) State **two** advantages of the object-oriented approach to program design over the structured approach to program design.

1. _____

2. _____

(2)

- (b) A golf club keeps details of its members. Each member has a unique membership number, first name, surname and telephone number recorded. Three classes have been identified:

Member
MidWeekMember
FullMember

The classes *MidWeekMember* and *FullMember* are related, by single inheritance, to the class *Member*.

Draw an inheritance diagram for the given classes.



(2)

- (c) Programs that use objects of the class *Member* need to add a new member's details, amend a member's details, and show a member's details. No other form of access is to be allowed. Write a class definition for this class.

`Member = Class`

EXAM PAPERS PRACTICE

End

(4)

(Total 8 marks)

Q22.

For an object-oriented program to store and calculate payroll details for an organisation, an **Employee** class is needed. A subclass has been identified: **HourlyPaidEmployee**, which has an inheritance relationship with class **Employee**.

- (a) Draw an inheritance diagram for these classes.

(2)

- (b) The **Employee** class has data fields *Name*, *National Insurance Number*, *Annual Pay*, *Gross Pay To Date*.

The class definition for **Employee** is

```
TEmployee = Class
    Public
        Procedure AddNewEmployee
        Procedure AmendEmployeeDetails
        Procedure PrintPaySlip
        Procedure CalculatePay
    Private
        Name : String
        NationalInsuranceNumber : String
        Annual Pay : Currency
        GrossPayToDate : Currency
    End
```

Monthly pay for an employee object of **TEmployee** class definition is calculated differently from the monthly pay for an employee object of **THourlyPaidEmployee** class definition.

In the case of an employee object of class definition

- **THourlyPaidEmployee**: monthly pay is calculated by multiplying number of hours worked in month by hourly pay rate.
- **TEmployee**: monthly pay is calculated by dividing the annual pay by 12.

An hourly paid employee object needs one additional operation, which collects the number of hours worked in a month.

Write the class definition **THourlyPaidEmployee**:

(6)
(Total 8 marks)

Q23.

One of the concepts of Object Oriented Programming is *containment*.

Class TForm1 inherits from class TForm.

A form, Form1, of class TForm1, contains 2 buttons, Button1 and Button2, of class TButton.

Write the class definition for TForm1.

(Total 3 marks)

Q24.

- (a) State **two** advantages of the object-oriented approach to program design over the structured approach to program design.

1. _____

2. _____

(2)

- (b) A sailing club has both junior and senior members. Each member has a unique membership number, a name and an address recorded. Three classes have been identified:

Member
JuniorMember
SeniorMember

The classes JuniorMember and SeniorMember are related, by single inheritance, to the class Member.

Draw an inheritance diagram for the given classes.

(2)

- (c) Programs that use objects of the class Member need to add a new member's details, amend a member's details, and show a member's details. No other form of access is to be allowed. Write a class definition for this class.

Member = **Class**

End;

(4)

(Total 8 marks)

Q25.

A supermarket has a section labelled 'Bottled Water'. Bottled water comes as 'still bottled water' or 'carbonated bottled water'.

EXAM PAPERS PRACTICE

In an object-oriented program, 'bottled water', 'still bottled water' and 'carbonated bottled water' are three defined *classes*. The classes 'still bottled water' and 'carbonated bottled water' are related, by single *inheritance*, to 'bottled water'.

- (a) What is meant here by

- (i) class? _____

(ii) inheritance? _____

(2)

- (b) Draw an inheritance diagram for the given classes.

(3)

- (c) Give **three** advantages of the object-oriented approach to programming over a structured approach.

1. _____

2. _____

3. _____

(3)

(Total 8 marks)

Q26.

- (a) In an object-oriented, computerised encyclopaedia, there is a class called *Creatures*. Two sub-classes of *Creatures* are *Spiders* and *Beetles*. Draw an inheritance diagram for this.

EXAM PAPERS PRACTICE

(2)

- (b) For the sub-class *Spiders* suggest:

- (i) **one** property;

- (ii) **one** method.

(2)

(Total 4 marks)

Q27.

Data may be recorded as *analogue* or *digital* signals.

- (a) Explain or show by diagram the difference between analogue and digital signals.

(2)

A computer system programmed in an object-oriented language is capable of displaying time in both analogue and digital form.

Three classes have been identified.

Clock

Digital Clock

Analogue Clock

The classes Digital Clock and Analogue Clock are related by single inheritance to the class Clock.

(b) In object-oriented programming what is meant by:

(i) class;

EXAM PAPERS PRACTICE

(1)

(ii) inheritance?

(1)

(c) Draw an inheritance diagram for the given classes.

(3)
(Total 7 marks)

Q28.

A vehicle manufacturer of both cars and lorries has a computer system programmed in an object-oriented language. Three classes have been identified:

Vehicle
Car
Lorry

The classes Car and Lorry are related by single inheritance to the class Vehicle.

(a) In object-oriented programming what is meant by:

(i) a class;

(1)

(ii) inheritance?

(1)

(b) Draw an inheritance diagram for the given classes.

(3)
(Total 5 marks)

EXAM PAPERS PRACTICE