|  |  |
| --- | --- |
|  | |
| **A-level Computer Science (7516/7517)**  Spring Programming Benchmark  Preditor Prey | Name:  Class: |

|  |  |
| --- | --- |
|  | |
| Time: | **1hr15mins** |
| Marks: | **75** |
| Comments: | **There is NO EAD!!! Most of the Paper is fill in the blanks.. For question 5 please paste into a word document with your name. Print and Staple Q5 to the back of this paper.** |
|  | |

|  |  |
| --- | --- |
| Q1 |  |
| Q2 |  |
| Q3 |  |
| Q4 |  |
| Q5 |  |
| Total |  |

# Section A

**Q1.**  Create a folder / directory for your new program.

One method for converting a decimal number into binary is to repeatedly divide by 2 using integer division. After each division is completed, the remainder is output and the integer result of the division is used as the input to the next iteration of the division process. The process repeats until the result of the division is 0.

Outputting the remainders in the sequence that they are calculated produces the binary digits of the equivalent binary number, but in reverse order.

For example, the decimal number 210 could be converted into binary as shown below.

|  |  |  |
| --- | --- | --- |
|  | 210 ÷ 2 =105 105 ÷ 2 =  52  52 ÷ 2 =  26 26 ÷ 2 = 13 13 ÷ 2 =  6 6 ÷ 2 =  3 3 ÷ 2 =  1 1 ÷ 2 =  0 | remainder 0 remainder 1 remainder 0 remainder 0 remainder 1 remainder 0 remainder 1 remainder 1 |

The sequence 0, 1, 0, 0, 1, 0, 1, 1 which would be output by this process is the reverse of the binary equivalent of 210 which is 11010010.

**What you need to do**

**Task 1**Write a program that will perform the conversion process described above. The program should display a suitable prompt asking the user to input a decimal number to convert and then output the bits of the binary equivalent of the decimal number in reverse order.

**Task 2**Improve the program so that the bits are output in the correct order, e.g. for 210 the output would be 11010010.

**Task 3**Test the program works by entering the value 210.

Save the program in your new folder / directory.

**Evidence that you need to provide**

(a)     Your PROGRAM SOURCE CODE after you have completed both **Task 1** and **Task 2**.

If you complete **Task 1** but do not attempt **Task 2** then a maximum of 9 marks will be awarded.

**(12)**

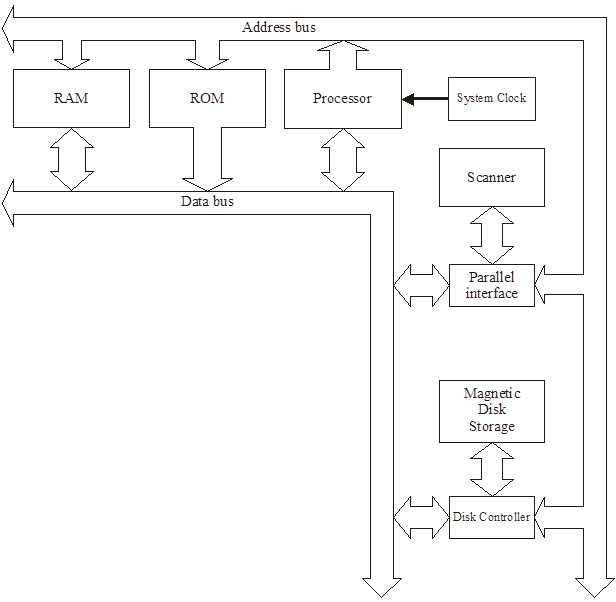
(b)     SCREEN CAPTURE(S) for the test showing the output of the program when 210 is entered.

The marks for this test will be awarded whether the binary digits are output in reverse order or in the correct order.

**(2)**

**(Total 14 marks)**

**Q2.**The figure below shows an incomplete diagram of a typical computer system architecture.



*(a)     {question removed}*

(b)     A third bus has been omitted from the diagram in the figure above.

Name this bus.

...........................................................................**(1)**

(c)     Explain why the data bus is bi-directional, but the address bus is one-way only.

……......................................................................

…………………………………………………………

………………………………………………………..

………………………………………………………...**(2)**

(d)     The processor performs different types of operations; for example, arithmetic operations.

Name **one** other type of operation. ................................................................

........................................................................................................................

**(1)**

(e)     Explain the **stored program concept**. ..........................................................

........................................................................................................................

........................................................................................................................

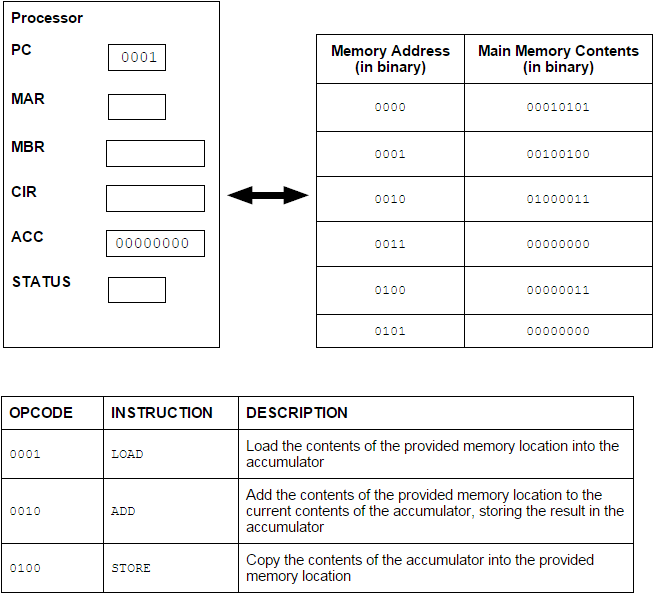
........................................................................................................................

**(3)**

**(Total 9 marks)**

**Q3.**

The diagram below shows some of the registers used in the fetch-execute cycle of a simple processor and the contents of a small section of main memory that it is connected to by the system bus ().



(a)     In the diagram above the first 4 bits of an instruction represent the opcode and give the type of instruction to be executed.

What name is given to the second 4 bits of an instruction?

........................................................................................................................

**(1)**

(b)     (i)      Currently the value in the Program Counter (PC) is example 0001.

Complete the table below by writing the values, expressed in binary, in the following registers after completing the fetch part of the fetch-execute cycle.

|  |  |  |
| --- | --- | --- |
|  | **Register** | **Value** |
|  | PC |  |
|  | MAR |  |
|  | MBR |  |

**(3)**

(ii)     Describe what will happen during the decode and execute part of the cycle.

...............................................................................................................

...............................................................................................................

...............................................................................................................

...............................................................................................................

...............................................................................................................

...............................................................................................................

...............................................................................................................

**(3)**

(c)     What would be the outcome of executing the instruction 01000011?

........................................................................................................................

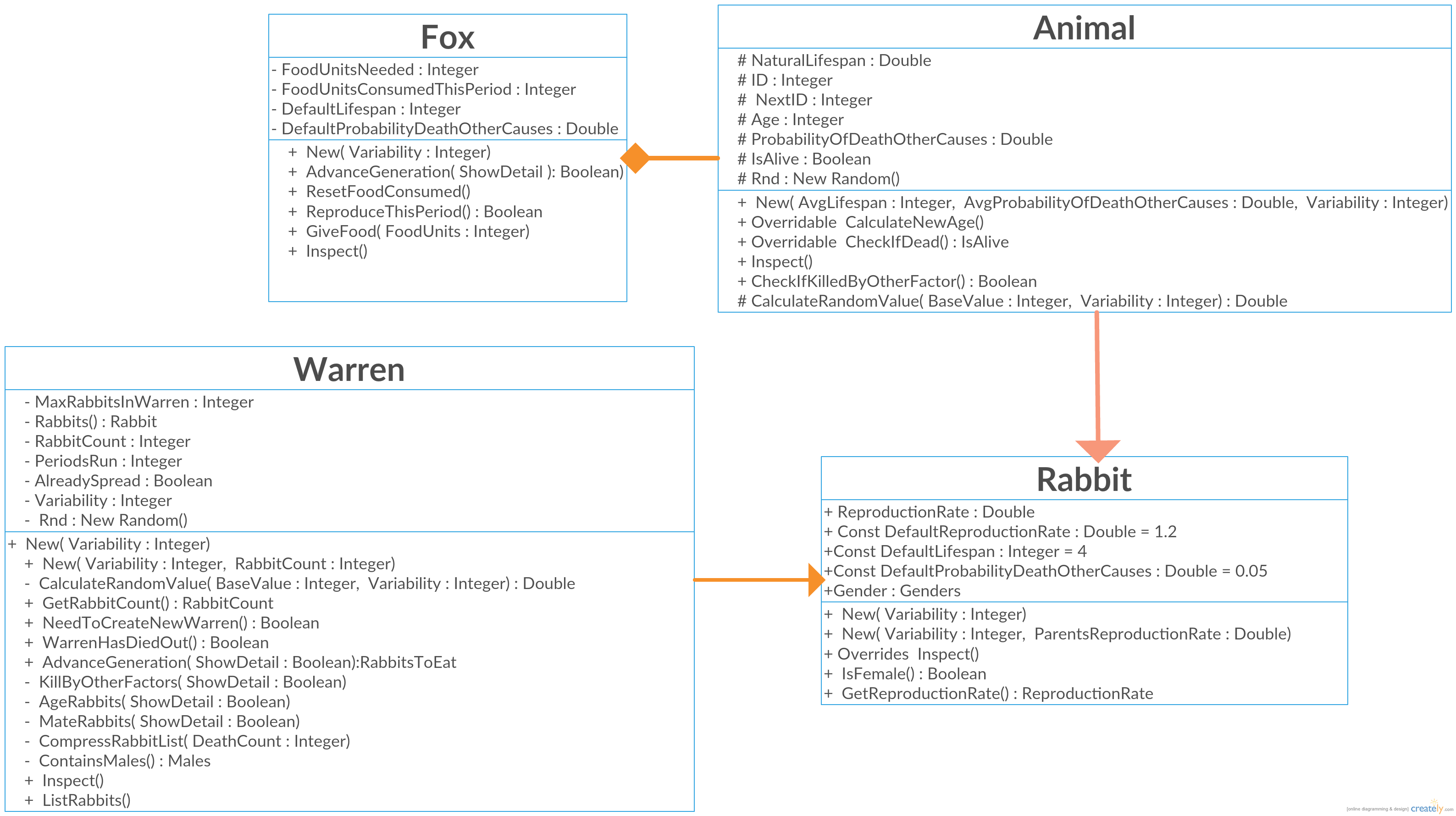
........................................................................................................................

........................................................................................................................

........................................................................................................................

**(1)**

**(Total 8 marks)**

**Q4.**  The class diagram below is an attempt to represent the relationships between some of the classes in the Predator Prey Simulation. ALL of question 4 refers only to the correct classes from the **Skeleton Program** shown in the diagram.

(a)     Describe three errors that have been made in the class diagram.

**(3)**

**Arrows should go FROM Rabbit/Fox to Animal;**

**FOX arrow should not be a diamond (as inherited not composed)**

**Rabbit to Warren line should be a diamond showing composition**

(b)     Give an example of instantiation

Rabbits() as Rabbit ; A another example correct in the code but not shown on diagram **(1)**

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

**(1)**

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

**(1)**

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

**Any Integer variable (1)**

(f)     State the name of an identifier for a class that uses composition. Warren **(1)**

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

Why has a named constant DefaultLifeSpan been used instead of the numeric value 4? **(2)**

**(Total 10 marks)**

**Q5.**

(a)     This question refers to the Main Program

The **Skeleton Program** currently does not validate the user input for the number of foxes.

The subroutine Main needs to be adapted so that it displays an error message to the user if an illegal value is entered. A message should state "That is not a valid value".

**Evidence that you need to provide**

(i)      Your amended PROGRAM SOURCE CODE for the MAIN subroutine.

**(3)**

(ii)    SCREEN CAPTURE(S) for a test run showing a player trying to enter the string “wibble” in the number of foxes

**(1)**

(iii)    SCREEN CAPTURE(S) for a test run showing a player trying to enter more foxes than there are locations

**(1)**

(b) This question will require you to create a new class

You are required to create a new class called **Owl** that inherits **Animal**, The DefaultLifeSpan should be a constant and set to 5. No other changes need to be made

You should instantiate an object named “Barney” of type OWL in the Location Class   
**NB this is not a complete solution and will not run! You will need to comment out the instantiation if you need to go back to a previous question**

(i)      Your PROGRAM SOURCE CODE for the **OWL** Class.

**(3)**

(ii) Your PROGRAM SOURCE CODE for the **Location** Class. (3)

**M1.**

(a)     **4 marks for AO3 (design) and 8 marks for AO3 (programming)**

**Mark Scheme**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Level** | **Description** | **Mark Range** |
|  | 4 | A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution that meets all of the requirements of **Task 1** and some of the requirements of **Task 2**. All of the appropriate design decisions have been taken. To award 12 marks, all of the requirements of both tasks must be met. | 10-12 |
|  | 3 | There is evidence that a line of reasoning has been followed to produce a logically structured program. The program displays a prompt, inputs the decimal value and includes a loop, which might be a definite or indefinite loop. An attempt has been made to do the integer division, output the remainder within the loop and use the result of the division for the next iteration, although some of this may not work. The solution demonstrates good design work as most of the correct design decisions have been taken. To award 9 marks, all of the requirements of **Task 1** must have been met. | 7-9 |
|  | 2 | A program has been written and some appropriate, syntactically correct programming language statements have been written. There is evidence that a line of reasoning has been partially followed as although the program may not have the required functionality for either task, it can be seen that the response contains some of the statements that would be needed in a working solution to **Task 1**. There is evidence of some appropriate design work as the response recognises at least one appropriate technique that could be used by a working solution, regardless of whether this has been implemented correctly. | 4-6 |
|  | 1 | A program has been written and a few appropriate programming language statements have been written but there is no evidence that a line of reasoning has been followed to arrive at a working solution. The statements written may or may not be syntactically correct. It is unlikely that any of the key design elements of the task have been recognised. | 1-3 |

**Guidance**

**Task 1:**

**Evidence of AO3 (design) – 3 points:**

Evidence of design to look for in responses:

•        Identifying that an indefinite loop must be used (as the length of the input is variable)

•        Identifying the correct Boolean condition to terminate the loop

•        Correct identification of which commands belong inside and outside the loop

*Note that AO3 (design) points are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works.*

**Evidence of AO3 (programming) – 6 points:**

Evidence of programming to look for in responses:

•        Prompt displayed

•        Value input by user and stored into a variable with a suitable name

•        Loop structure coded

•        Remainder of integer division calculated

•        Remainder of integer division output to screen

•        Result of integer division calculated and assigned to variable so that it will be used in the division operation for the next iteration

*Note that AO3 (programming) points are for programming and so should only be awarded for syntactically correct code.*

**Task 2:**

**Evidence of AO3 (design) – 1 point:**

Evidence of design to look for in responses:

•        A sensible method adopted for reversing the output eg appending to a string or storing into an array

*Note that AO3 (design) points are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works.*

**Evidence of AO3 (programming) – 2 points:**

Evidence of programming to look for in responses:

•        After each iteration remainder digit is stored into array / string or similar

•        At end of program bits output in correct order

*Note that AO3 (programming) points are for programming and so should only be awarded for syntactically correct code.*

**Example Solution VB.Net**

**Task 1:**

Dim DecimalNumber As Integer   
Dim ResultOfDivision As Integer   
Dim BinaryDigit As Integer  
  
Console.WriteLine("Please enter decimal number to convert")   
DecimalNumber = Console.ReadLine  
  
Do   
     ResultOfDivision = DecimalNumber \ 2   
     BinaryDigit = DecimalNumber Mod 2   
     Console.Write(BinaryDigit)   
     DecimalNumber = ResultOfDivision   
Loop Until ResultOfDivision = 0

**Task 2:**

Dim DecimalNumber As Integer   
Dim ResultOfDivision As Integer   
Dim BinaryDigit As Integer   
**Dim BinaryString As String**Console.WriteLine("Please enter decimal number to convert")   
DecimalNumber = Console.ReadLine   
**BinaryString = ""**Do   
     ResultOfDivision = DecimalNumber \ 2   
     BinaryDigit = DecimalNumber Mod 2   
     **BinaryString = BinaryDigit.ToString() + BinaryString**   
     DecimalNumber = ResultOfDivision   
Loop Until ResultOfDivision = 0  
  
**Console.WriteLine(BinaryString)**

**12**

(b)     **All marks AO3 (evaluate)**

\*\*\*\*SCREEN CAPTURE(S)\*\*\*\*

**Info for examiner:** Must match code from (a), including prompts on screen capture matching those in code. Code for (a) must be sensible.

**1 mark:** Display of suitable prompt and user input of value 210;

**1 mark:** Display of correct bits in reverse (01001011) or forward (11010010) order;

**A** Each bit value displayed on a separate line

**2**

**[14]**

**M2.**

1. removed

(b)     Control (bus) ;

**1**

(c)     Data bus has to transport data values to and from various devices /internal components ;

Only the processor assigns address values to the different devices ;

**Max 2**

(d)     Logical // read // write // jump/branch // input // output // data transfer ;

**A** Boolean

**1**

(e)     Program instructions are transferred from backing store to main memory ;

Program consists of a sequence of instructions ;

Program is stored in main memory ;

and can be replaced by another program at any time ;

Instructions are fetched (in sequence) ;

Decoded ;

and then executed ;

**Max 3**

**[9]**

**M3.**

(a)     operand;

**R** operand code

**1**

(b)     (i)      PC        0010;   
MAR     0001;  
MBR     00100100;

**3**

(ii)     The instruction is held in the CIR // instruction in CIR is decoded;  
**A** IR

The control unit / instruction decoder decodes the instruction;  
**NE** the processor decodes the instruction

Instruction will be split into opcode and operand;  
**R** if it is implied that a register will do this splitting / decoding

Relevant part of processor / CPU executes instruction // using ALU to perform calculations;  
**A** instruction executed by the control unit / ALU  
**NE** processor executes instruction

Further memory fetches / saves carried out if required;

Result of computation stored in accumulator / register / written to main memory;

Status register updated;  
If jump / branch instruction PC is updated;

*By example:*Will ADD contents memory location 0100 to accumulator;

**MAX 3**

(c)     The current value in the accumulator would be stored in (memory) address / location 0011 / 3;

Number 011 / 3 stored in (memory) address / location 0011 / 3;

**MAX 1**

**[8]**

**M4.**

(a)     **All marks AO2 (analyse)**

**1 mark:** The arrow should be pointing towards the base class;

**1 mark:** There is no class called Monster / / it should say Enemy, not Monster;

**2**

(b)     **Mark is for AO2 (apply)**

**VB.Net**

Dim MyGame As New Game(False) / /  
Dim MyGame As New Game(True) / /  
Private Player As New Character / /  
Private Cavern As New Grid(NSDistance, WEDistance) / /  
Private Monster As New Enemy / /  
Private Flask As New Item / /  
Private Trap1 As New Trap / /  
Private Trap2 As New Trap;

**R** If any additional code

**R** If spelt incorrectly

**I** Case

**1**

(c)     **Mark is for AO2 (apply)**

**VB.Net**

CavernState;

**R** If any additional code

**R** If spelt incorrectly

**I** Case

**1**

(d)     **Mark is for AO2 (apply)**

Trap / / Character / / Enemy;

**A** SleepyEnemy

**R** If any additional code

**R** If spelt incorrectly

**I** Case

**1**

(e)     **Mark is for AO2 (apply)**

Choice / / NoOfCellsEast / / NoOfCellsSouth / / Count / / NSDistance / / WEDistance / / Count1 / / Count2;

**R** If any additional code

**R** If spelt incorrectly

**I** Case

**1**

(f)     **Mark is for AO2 (apply)**

Game;

**R** If any additional code

**R** If spelt incorrectly

**I** Case

**1**

(g)     **Mark is for AO2 (analyse)**

So that a position of (0,0) is rejected / / so that the item can't be in the player's starting position;

**1**

(h)     **Marks are for AO1 (understanding)**

Makes the program code easier to understand;

Makes it easier to update the program;

Makes it easier to change the size of the cavern (in the game);

**Max 2 points from the list above**

**2**

(i)      **Marks are for AO2 (analyse)**

**1 mark:** Create a new object (Trap3) of class Trap;

**1 mark:** Change the (3rd ) If statement in the PlayGame subroutine by adding conditions to check if the player is in the same cell as Trap3 and that Trap3 has not been triggered already;

**2**

**[12]**

**M5.**

(a)     (i)      **Marks are for AO3 (programming)**

**1 mark:** Selection structure with one correct condition;

**1 mark:** Both conditions correct and correct logical operator(s);

**1 mark:** Subroutine returns the correct True / False value under all conditions;

**A** New conditions added to existing selection structure

**VB.Net**

Public Function CheckValidMove(ByVal Direction As Char) As Boolean  
  Dim ValidMove As Boolean  
  ValidMove = True  
  If Not (Direction = "N" Or Direction = "S" Or Direction = "W" Or Direction = "E" Or Direction = "M") Then  
    ValidMove = False  
  End If  
  **If Direction = "W" And  
Player.GetPosition.NoOfCellsEast = 0 Then**    **ValidMove = False**  **End If**  Return ValidMove  
End Function

**3**

(ii)     **Marks are for AO3 (programming)**

**1 mark:** Selection structure with correct condition added in correct place in the code;

**1 mark:** Correct error message displayed which will be displayed when move is invalid, and only when the move is invalid;

**I** Case of output message

**A** Minor typos in output message

**I** Spacing in output message

**VB.Net**

  ...  
  ValidMove = CheckValidMove(MoveDirection)  
  **If Not ValidMove Then**    **Console.WriteLine("That is not a valid move, please try again")**  **End If**Loop Until ValidMove  
...

**2**

(iii)    **Mark is for AO3 (evaluate)**

\*\*\*\*SCREEN CAPTURE(S)\*\*\*\*

**Info for examiner:** Must match code from (a)(i) and (a)(ii), including prompts on screen capture matching those in code. Code for (a)(i) and (a)(ii) must be sensible.

Screen capture(s) showing the error message being displayed after the player tried to move to the west from a cell at the western end of the cavern;

**A** Alternative output messages if match code for (a)(ii)

**1**

(b)     (i)      **Marks are for AO3 (programming)**

**1 mark:** SleepyEnemy class created;

**1 mark:** Inheritance from Enemy class;

**1 mark:** MovesTillSleep property declared;

**1 mark:** Subroutine MakeMove that overrides the one in the base class;

**1 mark:** MovesTillSleep decremented in the MakeMove subroutine;

**1 mark:** Selection structure in MakeMove that calls ChangeSleepStatus if the value of MovesTillSleep is 0; **A** Changing Awake property instead of call to ChangeSleepStatus

**1 mark:** Subroutine ChangeSleepStatus that overrides the one in the base class;

**1 mark:** Value of MovesTillSleep set to 4 in the ChangeSleepStatus subroutine;

**I** Case of identifiers

**A** Minor typos in identifiers

**VB.Net**

Class SleepyEnemy  
  Inherits Enemy  
  Private MovesTillSleep As Integer  
  
  Public Overrides Sub MakeMove(ByVal PlayerPosition As CellReference)  
    MyBase.MakeMove(PlayerPosition)  
    MovesTillSleep = MovesTillSleep - 1  
    If MovesTillSleep = 0 Then  
      ChangeSleepStatus()  
    End If  
  End Sub  
  
  Public Overrides Sub ChangeSleepStatus()  
    MyBase.ChangeSleepStatus()  
    MovesTillSleep = 4  
  End Sub  
End Class

**8**

(ii)     **Marks are for AO3 (evaluate)**

\*\*\*\*SCREEN CAPTURE(S)\*\*\*\*

**Info for examiner:** Must match code from (b)(i), including prompts on screen capture matching those in code. Code for (b)(i) must be sensible.

**1 mark:** Screen capture(s) showing the player moving east and then east again at the start of the training game. The monster then wakes up and moves two cells nearer to the player. The player then moves south;

**1 mark:** The monster moves two cells nearer to the player and then disappears from the cavern display;

**2**

(c)     (i)      **Mark is for AO3 (programming)**

Appropriate option added to menu;

**VB.Net**

Public Sub DisplayMoveOptions()  
  Console.WriteLine()  
  Console.WriteLine("Enter N to move NORTH")  
  Console.WriteLine("Enter S to move SOUTH")  
  Console.WriteLine("Enter E to move EAST")  
  Console.WriteLine("Enter W to move WEST")  
  **Console.WriteLine("Enter A to shoot an arrow")**  Console.WriteLine("Enter M to return to the Main Menu")  
  Console.WriteLine()  
End Sub

**1**

(ii)     **Marks are for AO3 (programming)**

**1 mark:** Direction of A is allowed;

**1 mark:** Direction of A allowed only if player has got an arrow;

**Maximum 1 mark:** If any other invalid moves would be allowed or any valid moves not allowed

**VB.Net**

Public Function CheckValidMove(ByVal Direction As Char) As Boolean  
  Dim ValidMove As Boolean  
  ValidMove = True  
  If Not (Direction = "N" Or Direction = "S" Or Direction = "W" Or Direction = "E" Or Direction = "M" **Or Direction = "A"**) Then  
    ValidMove = False  
  End If  
  **If Direction = "A" And Not Player.GetHasArrow Then**    **ValidMove = False**  **End If**  Return ValidMove  
End Function

**2**

(iii)     **Marks are for AO3 (programming)**

**1 mark:** Property HasArrow created;

**1 mark:** HasArrow set to True when an object is instantiated;

**1 mark:** Subroutine GetHasArrow created;

**1 mark:** GetHasArrow returns the value of HasArrow;

**1 mark:** Subroutine GetArrowDirection created;

**1 mark:** GetArrowDirection has an appropriate output message and then gets a value entered by the user;

**1 mark:** In GetArrowDirection, value keeps being obtained from user until it is one of N, S, W or E;

**1 mark:** HasArrow is set to False in GetArrowDirection;

**I** Additional output messages

**I** Case of identifiers

**A** Minor typos in identifiers

**VB.Net**

Class Character  
  Inherits Item  
  **Private HasArrow As Boolean**  Public Sub MakeMove(ByVal Direction As Char)  
    Select Case Direction  
    Case "N"  
      NoOfCellsSouth = NoOfCellsSouth - 1  
    Case "S"  
      NoOfCellsSouth = NoOfCellsSouth + 1  
    Case "W"  
      NoOfCellsEast = NoOfCellsEast - 1  
    Case "E"  
      NoOfCellsEast = NoOfCellsEast + 1  
    End Select  
  End Sub  
  
  **Public Sub New()**    **HasArrow = True**  **End Sub**  **Public Function GetHasArrow() As Boolean**    **Return HasArrow**  **End Function**  **Public Function GetArrowDirection() As Char**    **Dim Direction As Char**    **Do**      **Console.Write("What direction (E, W, S, N) would you like to shoot in?")**      **Direction = Console.ReadLine**    **Loop Until Direction = "E" Or Direction = "W" Or Direction = "S" Or Direction = "N"**    **HasArrow = False**    **Return Direction**  **End Function**End Class

**8**

(iv)    **Marks are for AO3 (programming)**

**1 mark:** Check for A having been entered – added in a sensible place in the code;

**1 mark:** If A was entered there is a call to GetArrowDirection;

**1 mark:** Selection structure that checks if the arrow direction is N;

**1 mark:** Detects if the monster is in any of the cells directly north of the player's current position;

**1 mark:** If the monster has been hit by an arrow then the correct output message is displayed and the value of FlaskFound is set to True;

**1 mark:** The code for moving the player and updating the cavern display is inside an *else* structure (or equivalent) so that this code is not executed if the player chooses to shoot an arrow;

**I** Case of output message

**A** Minor typos in output message

**I** Spacing in output message

**VB.Net**

If MoveDirection "M" Then  
  **If MoveDirection = "A" Then  
    MoveDirection = Player.GetArrowDirection  
    Select MoveDirection  
      Case "N"  
        If Monster.GetPosition.NoOfCellsSouth   
        Console.WriteLine("You have shot the monster and it cannot stop you finding the flask")  
        FlaskFound = True  
      End If  
    End Select  
  Else**    Cavern.PlaceItem(Player.GetPosition, " ")  
    Player.MakeMove(MoveDirection)  
    Cavern.PlaceItem(Player.GetPosition, "\*")  
    Cavern.Display(Monster.GetAwake)  
    FlaskFound = Player.CheckIfSameCell(Flask.GetPosition)  
  **End If**  If FlaskFound Then  
  ...

**6**

(v)     **Mark is for AO3 (evaluate)**

\*\*\*\*SCREEN CAPTURE(S)\*\*\*\*

**Info for examiner:** Must match code from (c)(i), (c)(ii), (c)(iii) and (c)(iv), including prompts on screen capture matching those in code. Code for (c)(i), (c)(ii), (c)(iii) and (c)(iv) must be sensible.

Screen capture(s) showing the user shooting an arrow northwards at the start of the training game and the message about the monster being shot is displayed;

**A** Alternative output messages if match code for (c)(iv)

**1**

(vi)    **Mark is for AO3 (evaluate)**

\*\*\*\*SCREEN CAPTURE(S)\*\*\*\*

**Info for examiner:** Must match code from (c)(i), (c)(ii), (c)(iii) and (c)(iv), including prompts on screen capture matching those in code. Code for (c)(i), (c)(ii), (c)(iii) and (c)(iv) must be sensible.

Screen capture(s) showing an arrow being shot, no message about the monster being hit is displayed and then the invalid move message is displayed when the player tries to shoot an arrow for a second time;

**1**

**[35]**