Paper 1 - Y12 Easter Break

Name: _____

Class: _____

Date: _____

Time: **90 minutes**

Marks: **74 marks**

Comments:

1. **Figure 1** contains the pseudo-code for a program to output a sequence according to the 'Fizz Buzz' counting game.

Figure 1

```
OUTPUT "How far to count?"
INPUT HowFar
WHILE HowFar < 1
  OUTPUT "Not a valid number, please try again."
  INPUT HowFar
ENDWHILE
FOR MyLoop ← 1 TO HowFar
  IF MyLoop MOD 3 = 0 AND MyLoop MOD 5 = 0
  THEN
      OUTPUT "FizzBuzz"
    ELSE
      IF MyLoop MOD 3 = 0
        THEN
          OUTPUT "Fizz"
        ELSE
          IF MyLoop MOD 5 = 0
            THEN
              OUTPUT "Buzz"
            ELSE
              OUTPUT MyLoop
          ENDIF
      ENDIF
  ENDIF
ENDFOR
```

What you need to do:

Write a program that implements the pseudo-code as shown in **Figure 1**.

Test the program by showing the result of entering a value of `18` when prompted by the program.

Test the program by showing the result of entering a value of `-1` when prompted by the program.

Evidence that you need to provide

(a) Your PROGRAM SOURCE CODE for the pseudo-code in **Figure 1**.

(8)

(b) SCREEN CAPTURE(S) for the tests conducted when a value of `18` is entered by the user and when a value of `-1` is entered by the user.

(1)

The main part of the program uses a `FOR` repetition structure.

(c) Explain why a `FOR` repetition structure was chosen instead of a `WHILE` repetition structure.

_____

_____

(1)

(d) Even though a check has been performed to make sure that the variable `HowFar` is greater than `1` there could be inputs that might cause the program to terminate unexpectedly (crash).

Provide an example of an input that might cause the program to terminate and describe a method that could be used to prevent this.

(3)

(e) Programs written in a high level language are easier to understand and maintain than programs written in a low level language.

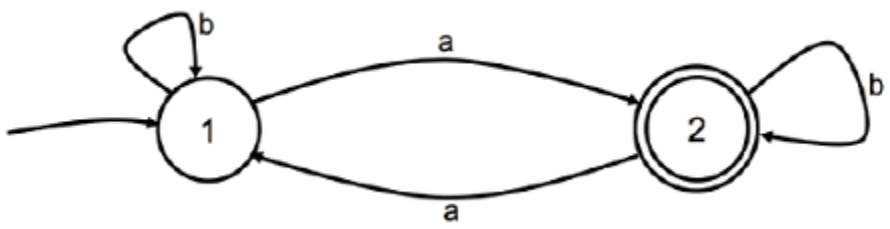
The use of meaningful identifier names is one way in which high level languages can be made easier to understand.

State **three** other features of high level languages that can make high level language programs easier to understand.

(3)

(f) The finite state machine (FSM) shown in **Figure 2** recognises a language with an alphabet of `a` and `b`.

Figure 2



Input strings of `a` and `aabba` would be accepted by this FSM.

In the table below indicate whether each input string would be accepted or not accepted by the FSM in **Figure 2**.

If an input string would be accepted write YES.
If an input string would **not** be accepted write NO.

Input string	Accepted by FSM?
`aaab`	
`abbab`	
`bbbbba`	

(2)

(g) In words, describe the language (set of strings) that would be accepted by this FSM shown in **Figure 2**.

_____

_____

_____

_____

(2)
(Total 20 marks)

2.

Based on the description in Statius' journal you are sure that this must be the right place. The blue-green moss covering the rocks and the dense tree foliage combine to conceal the cave entrance; you almost walked straight past it and it was only through luck that you saw it. There isn't any time to waste – ever since the journal was discovered, everyone has been looking for this place. The thick cobwebs across the entrance prove that you must be the first one here. You know that, if you are right, you could be the one that finds, in the cavern below the mountain, the single draft of Styxian potion contained in Statius' flask. The journal says that there is a fearsome beast lying in wait, but the risk is worth it. Statius wrote that consuming the potion would grant the drinker invulnerability. Nothing could hurt you, cut you, graze, scratch or bruise you. Your thoughts start to drift and you imagine what you could do with such power.

"Snap out of it," you tell yourself. Someone else could find this place and you can't take that risk; the flask contains only enough potion for one. Quickly you shoulder your pack, then you light your torch, brush aside some cobwebs and step into the cave.

Inside, the ground is rough and you stumble several times. As you go deeper into the mountain, the cave darkens and soon the only light is coming from your torch. After you have been walking for a few minutes the cave widens and then ends abruptly. You take out your copy of the journal. You read the description of the cave again. It says that, at the end of the cave, there is a large fissure near the western part of the wall and that the cavern, where the flask is hidden, is at the bottom of the fissure. You move over to the west side of the cave. Sure enough, the fissure is there. You take off your pack and then move carefully nearer the edge. The light from the torch does not reach far enough down to reveal the bottom, but you can see that the fissure walls are too steep to get down unaided so you will need your climbing equipment. You place your torch carefully on the floor nearby and take your rope, pitons and carabiners out of your pack.

Your foot slips on some loose stones and you fall backwards into the fissure. You tumble down the hole in a shower of dust and pebbles, falling into the cavern below. You land painfully on the rocky floor.

It is dark; your torch is back in the cave and it weakly illuminates your immediate surroundings but you cannot see any farther into the cavern. You become aware of a sonorous noise around you and it takes you a few minutes to work out what it is. The monster is asleep somewhere in the cavern and is snoring loudly. The sound reverberates around you so you can't work out in which direction the monster lies.

You can see the bottom aperture of the fissure several metres above your head and try to scramble up the wall to reach it, but the rock face is too sheer and you can't get sufficient purchase. From what you can remember of the journal, you must be at the north-west corner of the cavern.

You make your mind up. You can't get out of the cavern the way that you came in and the monster could wake soon. You decide to explore the cavern. Maybe you will find another way out. Maybe you will find the Styxian potion. You have no choice but to play the game of...

MONSTER!

MONSTER!

The **Skeleton Program** in this Preliminary Material is a program for the one-player game MONSTER!.

When playing MONSTER! the player starts in the north-west corner of a dark cavern. The cavern is represented by a 7 × 5 rectangular grid of cells. The player's current position is indicated by an *. The player is presented with a list of five options: they can either return to the main menu (where they can save the current game if they want to) or they can move one cell in one of four possible directions. If they enter ‘N’ they will move one cell to the north, ‘S’ will move them one cell to the south, ‘E’ will move them one cell to the east and ‘W’ will move them one cell to the west. The initial position of a new game is shown in **Figure 1**.

Figure 1

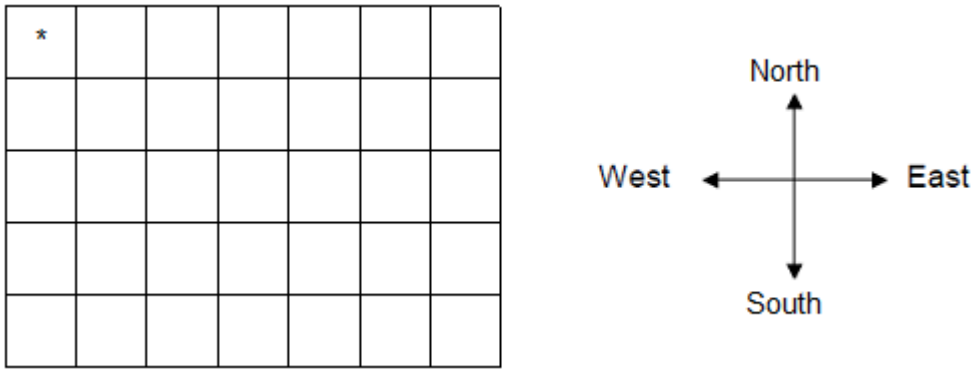
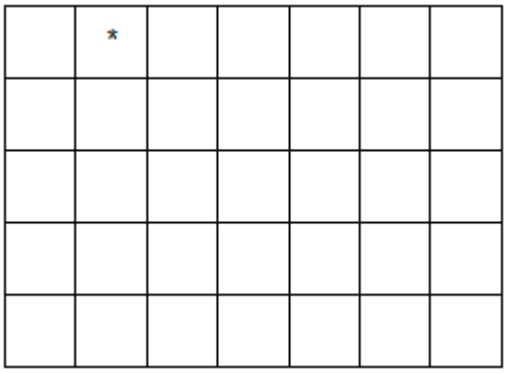


Figure 2 shows a new state, resulting from the user selecting ‘E’ in the starting position.

Figure 2



The aim of MONSTER! is to find the hidden treasure (a flask containing a Styxian potion) that is in one of the cells of the cavern. Unfortunately, the cavern is dark and the only way that a player can find the flask is to move around until they are in the same cell as the flask. When a new game is started, the flask will be in a random position in the cavern (though it won't be in the same cell as the player starts in). If the player moves into the cell that contains the flask, then they win the game.

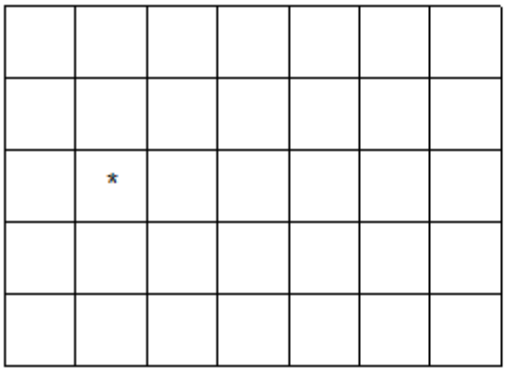
The cavern is also the lair of a fearsome monster that guards the flask. At the start of a new game the monster is asleep. As the cavern is dark the player cannot see where the monster is. If the player moves into the cell that contains the monster then the monster will wake up and eat the player and the player will have lost the game.

The monster has set two traps in its cavern. The player cannot see where these traps are. If the player moves into a cell that contains a trap then the monster will wake up. When the monster is awake, it will move around the cavern until it either eats the player or the player finds the flask. The monster is twice as fast as the player and makes two moves for each player move. Each move is one cell in one of the four possible directions. When it is awake the monster's skin glows so the player can see the position of the monster. The monster can see in the dark and so knows where the player is. The current position of the moving monster is indicated by an ‘M’ in the cavern displayed to the player.

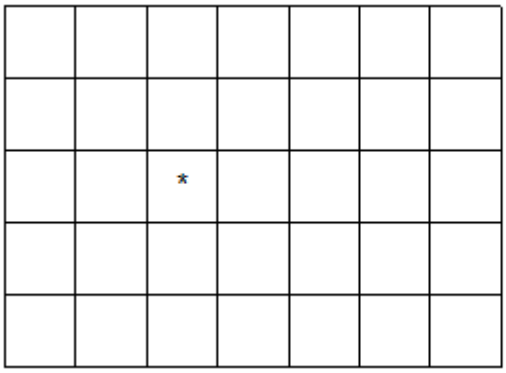
If the monster moves into the same cell as the flask, it kicks the flask out of the way and the flask will be moved to the cell the monster came from. When the monster is awake, it does not matter if the player (or the monster) moves into a cell containing one of the traps.

Figure 3 shows part of a possible game as displayed to the player. The player moves one cell to the east, which triggers a trap that wakes the monster. The player is then shown the position of the monster in the cavern. The monster then makes its first move and the new state of the cavern is shown. It then makes its second move and the updated state of the cavern is shown. It is then the player's turn to move.

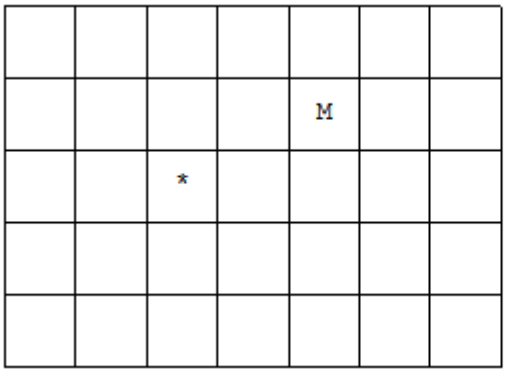
Figure 3



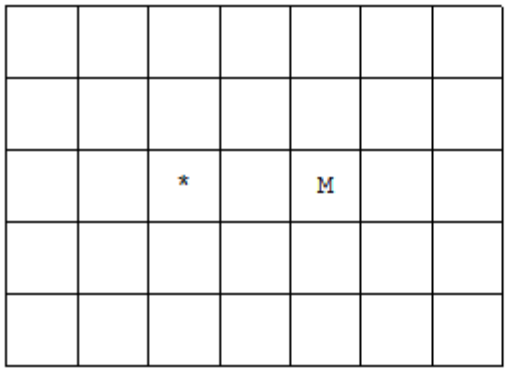
The position of the player is shown.



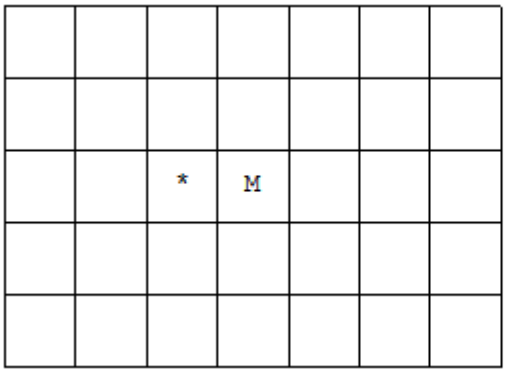
The player moves to the east.



This sets off a trap and wakes up the monster.



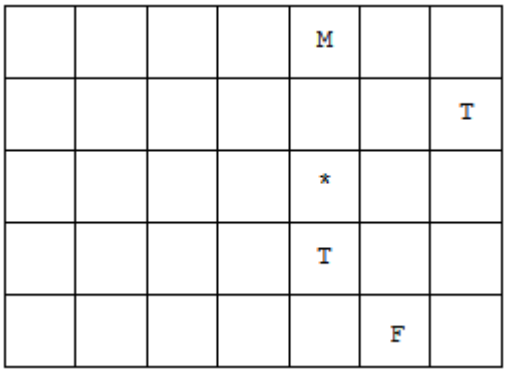
The monster makes a move.



The monster makes another move. Now it is the player's turn again.

In the Skeleton Program there is a menu containing **five** options: 'Start new game', 'Load game', 'Save game', 'Play training game' and 'Quit'. If the user chooses 'Load game' then the contents of a user-specified file are loaded and the user will start playing MONSTER! from the game state saved in the file. If the user chooses 'Save game' then they will be asked to enter a name for the file and then the current state of the game will be stored in a file with the name supplied by the user. If the user chooses 'Play training game' then the user will start playing MONSTER! from the game state shown in **Figure 4**.

Figure 4



'F' denotes the position of the flask; 'T' denotes the position of a trap. The flask, traps and monster are not displayed to the player when the training game starts.

3. State the name of an identifier for:

(a) an array or list variable

_____

_____

(1)

(b) a user-defined subroutine that has four parameters

_____

_____

(1)

(c) a variable that is used to store a whole number.

_____

_____

(1)

(d) a user-defined subroutine that returns one or more values.

_____

_____

(1)

(e) Look at the repetition structures in the `DisplayCavern` subroutine.

Explain the need for a nested `FOR` loop and the role of the `Count1` and `Count2` variables.

_____

_____

_____

_____

_____

_____

(3)

(f) Look at the `ResetCavern` subroutine.

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

(2)

(g) Look at the `SetPositionOfItem` subroutine.

Describe the purpose of the `WHILE` loop and the command within it in this subroutine.

(3)

(h) Look at the `MakeMonsterMove` subroutine.

Describe why it is necessary to check if the monster moves into the same cell as the flask and how any problem caused by this is solved by the **Skeleton Program**.

(3)

(i) Look at the `PlayGame` subroutine.

Explain why a `WHILE` loop has been made to complete the two moves for the monster rather than a `FOR` loop.

_____

_____

_____

_____

(2)

(j) The subroutines in the **Skeleton Program** avoid the use of global variables: they use local variables and parameter passing instead.

State **two** reasons why subroutines should, ideally, **not** use global variables.

_____

_____

_____

_____

(2)

(Total 19 marks)

4. (a) This question refers to the subroutines `CheckValidMove` and `PlayGame`.

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** is to be adapted so that it prevents a player from moving north if they are at the northern 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 north when they are at the northern end of the cavern.

The subroutine `PlayGame` is 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`.

(4)

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

(1)

(iii) SCREEN CAPTURE(S) for a test run showing a player trying to move north when they are at the northern end of the cavern.

(1)

(b) This question refers to the `PlayGame` subroutine and will extend the functionality of the game.

A scoring system is to be implemented as a game of MONSTER! is played. A variable called `Score` will be used to store the current score of each player.

The final score will be displayed to the user at the end of the game. At the end of the game, either the player will have found the flask or the player will have been eaten by the monster.

The final score should be displayed with the message "`Your score was: Y`" where `Y` is the value of `Score`.

The scoring system will be based upon the following:

- each valid move by the player is +10 points
- finding the flask is +50 points
- setting off a trap is −10 points
- being killed by the monster is −50 points.

Task 1
Adapt the **Skeleton Program** so that the scoring system described above is implemented, with the value of `Score` being updated as indicated and the required message being displayed at the end of a game.

Task 2
Test that the changes you have made work by conducting the following test:

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

Evidence that you need to provide

(i) Your amended PROGRAM SOURCE CODE for the subroutine `PlayGame` and (if relevant) the PROGRAM SOURCE CODE for any other subroutine(s) you have amended.

(8)

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

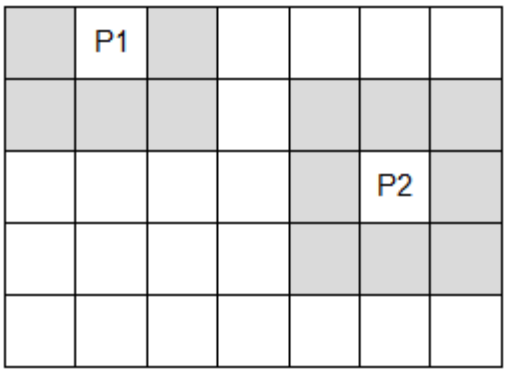
(1)

(c) This question will extend the functionality of the game.

The player will now have access to a close-range trap detector. After making a directional move in the cavern, the trap detector will perform a sweep of the neighbouring cells and report back if a trap is detected. Unfortunately, the detector can only detect the presence of a trap in a neighbouring cell, and not which individual cell the trap is in.

In **Figure 1** the shaded cells show the cells that would be scanned by the trap detector if the player were in the cell marked P1 or P2. The trap detector cannot scan outside the cavern.

Figure 1



Task 1
Create a new subroutine, `TrapDetector`, that, when given the current location of the player, returns `True` if a trap is in a neighbouring cell and `False` if there is no trap in a neighbouring cell.

When creating this subroutine you should ensure that your solution is efficiently coded.

Task 2
Modify the `PlayGame` subroutine so that after the player moves and the new state of the cavern is displayed:

- the message '`Trap detected`' is displayed if there is a trap in any neighbouring cell.
- the message '`No trap detected`' is displayed if there are no traps in any neighbouring cell.

Task 3
Test that your program works by loading the training game and showing that:

- a trap is detected after the player's first move, west
- a trap is detected after the player's second move, south
- a trap is not detected after the player's third move, west.

Evidence that you need to provide

(i) Your PROGRAM SOURCE CODE for the subroutine `TrapDetector`.

(12)

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

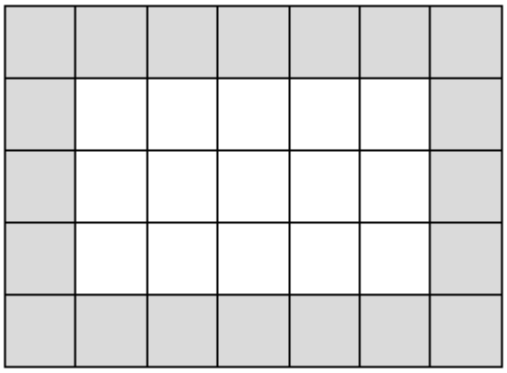
(1)

(iii) SCREEN CAPTURE(S) showing the required sequence of tests being carried out, with the trap detected message being displayed after each of the first two moves and the trap not detected message being displayed after the third move.

(1)

(iv) The game of MONSTER!, as represented by the **Skeleton Program**, is to be extended so that the cavern generated is not rectangular. The outer cells, shaded in **Figure 2**, will be randomly selected to be either rock or normal space when a new game starts. A cell that contains rock cannot be entered by the monster or player.

Figure 2



Describe changes that could be made to the **Skeleton Program** to achieve this.

In your answer you should ensure that you discuss changes to the data held in the `Cavern` variable and how the subroutines `ResetCavern` and `CheckValidMove` will need to be altered.

You are **not** expected to actually make the changes.

(5)

(v) A request has been made that the layout of the whole cavern should be more random. It has been suggested that all of the cells should be made a random choice between rock and normal space during setup.

Identify **two** problems that might occur with the MONSTER! game if this suggestion was made to the program.

_____

_____

_____

_____

(2)

(Total 35 marks)