0 1 The algorithm, represented using pseudo-code in **Figure 2**, outputs a series of numbers. The contents of the series depends on the initial value entered by the user.

Figure 2

```
Number ← 0
WHILE (Number < 1) OR (Number > 10)
  OUTPUT "Enter a positive whole number: "
  INPUT Number
  IF Number > 10 THEN
    OUTPUT "Number too large."
  ELSE
    IF Number < 1 THEN
      OUTPUT "Not a positive number."
    ENDIF
  ENDIF
ENDWHILE
c ← 1
FOR k ← 0 TO Number - 1
  OUTPUT c
  c ← (c * (Number - 1 - k)) DIV (k + 1)
ENDFOR
```

The `DIV` operator calculates the result of an integer division, for example, `10 DIV 3 = 3`.

What you need to do:

Task 1
Write a program to implement the algorithm in **Figure 2**.

Task 2
Test the program by showing the result of entering:
`-3`
then `11`
then `10`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

0 1 . 1 Your PROGRAM SOURCE CODE for **Task 1**.

[9 marks]

0 1 . 2 SCREEN CAPTURE(S) showing the test described in **Task 2**.

[1 mark]

0 2 This question refers to the subroutine `SendMorseCode`.

What you need to do:

Task 1
Modify the subroutine `SendMorseCode` so that it checks each character of the input string entered by the user.

If a character is not an uppercase letter or a space then the program should use the subroutine `ReportError` to output a suitable error message.

The subroutine should then set `MorseCodeString` to the empty string and exit.

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

- run the program
- choose option `S`
- enter the string `Help`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

0 2 . 1 Your amended PROGRAM SOURCE CODE for the subroutine `SendMorseCode`.

[4 marks]

0 2 . 2 SCREEN CAPTURE(S) showing the requested test.

[1 mark]

0 3 This question will extend the functionality of the program. To send a Morse code message, the dots and dashes need to be converted into signals.

What you need to do:

Task 1
Create a new subroutine `SendSignals` that converts a Morse code message such as

```
-⌂.⌂.-⌂⌂⌂-..-
```

(`TEA X`) to its equivalent signal

```
===⌂⌂⌂=⌂⌂⌂=⌂===⌂⌂⌂⌂⌂⌂⌂===⌂=⌂=⌂===⌂
```

by using the rules

- a dot is converted to an equals sign (=) followed by a space
- a dash is converted to three equals signs (===) followed by a space
- a space is converted to two spaces.
  (This results in three spaces after the signals for a letter.)

Note that ⌂ represents a space and should be displayed as a space on screen.

The subroutine is to output the complete transmission string to the screen.

Add the call `SendSignals(MorseCodeString)` as the last statement to be executed in the subroutine `SendMorseCode`.

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

- run the program
- choose option `S`
- enter the string `MORSE X`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

0 3 . 1 Your PROGRAM SOURCE CODE for the subroutine `SendSignals`.
[7 marks]

0 3 . 2 SCREEN CAPTURE(S) showing the Morse code message and the signals.
[1 mark]

0 4 This question will extend the program to output the letters of the alphabet and their corresponding Morse codes.

What you need to do:

Task 1
Create a new subroutine `OutputAlphabetWithCode`. This subroutine is to display each uppercase letter of the alphabet and its corresponding Morse code. The output is to be created using the data in `Letter` and `MorseCode`.

The format of the output should be as shown in **Figure 4**.

- Each uppercase letter is followed by a space and then its Morse code, grouped four to a line.
- Each Morse code should be padded with trailing spaces to a width of 6 characters.

Figure 4

```
A .-    B -...  C -.-.  D -..
E .     F ..-.  G --.   H ....
I ..    J .---  K -.-   L .-..
M --    N -.    O ---   P .--.
Q --.-  R .-.   S ...   T -
U ..-   V ...-  W .--   X -..-
Y -.--  Z --..
```

Task 2
Amend the subroutines `DisplayMenu` and `SendReceiveMessages` so that the user can choose a new option `A` that will call `OutputAlphabetWithCode`.

Task 3
- run the program
- choose new option `A`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

0 4 . 1 Your PROGRAM SOURCE CODE for the subroutine `OutputAlphabetWithCode`. Your amended PROGRAM SOURCE CODE for the subroutines `DisplayMenu` and `SendReceiveMessages`.

[6 marks]

0 4 . 2 SCREEN CAPTURE(S) showing the main menu and the output from the test run.

[1 mark]

0 5 This question will further extend the functionality of the program.

Before sending a message in Morse code the message is to be encrypted using a Caesar cipher. To encrypt a character (letter or space) it is shifted a given number of places (the key) in the alphabet (space is taken as the first character, then A, B, C and so on to Z). If a character is required before space or after Z then wrap-round is to be implemented.

For example:

- if the key is 2, then SPACE is encrypted as B, A is encrypted as C, B is encrypted as D, …, X is encrypted as Z, Y is encrypted as SPACE, and Z is encrypted as A.
- if the key is -1, then SPACE is encrypted as Z, A is encrypted as SPACE, B is encrypted as A, …, and Z is encrypted as Y.

Three different keys are to be used depending on the position of the character in the message.

The first character in the message is to be encrypted using the first key, the second character is encrypted with the second key and the third character with the third key.

The fourth character is then encrypted using the first key again, the fifth character is encrypted with the second key and the sixth character with the third key and so on.

The three keys are repeatedly used like this until the end of the message.

What you need to do:

Task 1
At the start of the subroutine `SendReceiveMessages` add code to enter three values that **must** be integers (the keys).

Task 2
Amend the subroutine `SendMorseCode` to encrypt the message using the substitution method described above and the keys entered by the user.

Task 3
Test that your code works by conducting the following test:

- run the program
- for the first key enter `17`
- for the second key enter `5`
- for the third key enter `-3`
- enter `S`
- enter the string `TEA X`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

0 5 . 1 Your amended PROGRAM SOURCE CODE for the subroutines `SendMorseCode` and `SendReceiveMessages`.

[9 marks]

0 5 . 2 SCREEN CAPTURE(S) showing the keys entered, the plain text and the encrypted Morse code message.

[1 mark]

0 6 The algorithm, represented using pseudo-code, in **Figure 2** outputs a numeric result. The numeric result depends upon the value entered by the user.

Figure 2

```
OUTPUT "Enter a positive whole number: "
INPUT NumberIn
NumberOut ← 0
Count ← 0
WHILE NumberIn > 0
  Count ← Count + 1
  PartValue ← NumberIn MOD 2
  NumberIn ← NumberIn DIV 2
  FOR i ← 1 TO Count - 1
    PartValue ← PartValue * 10
  ENDFOR
  NumberOut ← NumberOut + PartValue
ENDWHILE
OUTPUT "The result is: " NumberOut
```

Table 2 lists the `MOD` and `DIV` operators for each of the available programming languages. You should refer to the row for your programming language.

Table 2

Programming language	MOD	DIV
C#	%	/
Java	%	/
Pascal	mod	div
Python	%	//
VB.Net	Mod	\

What you need to do:

Task 1
Write a program to implement the algorithm in **Figure 2**.

Task 2
Test that your program works:

- run your program, then enter the number 22
- run your program, then enter the number 29
- run your program, then enter the number -1

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

0 6 . 1 Your PROGRAM SOURCE CODE for **Task 1**.

[11 marks]

0 6 . 2 SCREEN CAPTURE(S) showing the test described in **Task 2**.

[1 mark]

0 6 . 3 What is the purpose of this algorithm?

[1 mark]

0 7 This question refers to the subroutine `SelectMove`. This subroutine makes three calls to the subroutine `DisplayErrorCode` to notify the user of errors. The error codes passed as parameters are to be made more informative for the user.

What you need to do:

Task 1
Modify the subroutine `DisplayErrorCode` so that meaningful error messages are displayed for each type of error explaining the circumstances which caused each error. The error code should also be displayed.

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

- run the program
- enter `Y`
- load `game1.txt`
- enter the string `a4`
- enter the string `a9`
- enter `3`
- enter `4`
- enter `a`
- enter `9`
- enter `3`
- enter `0`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

0 7 . 1 Your PROGRAM SOURCE CODE for the entire subroutine `DisplayErrorCode`.
[3 marks]

0 7 . 2 SCREEN CAPTURE(S) showing the requested test including the list of possible moves for Player A.
[1 mark]

0 8 This question refers to the subroutine `ValidJump`. The rules of the game are to be amended. Instead of jumping over their own piece, a player can only jump over an opponent's piece.

What you need to do:

Task 1
Amend the subroutine `ValidJump` so that a jump is only possible if the middle piece belongs to the opponent.

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

- run the program
- enter `Y`
- load `game3.txt`
- enter the string `a5`
- enter `5`
- enter `0`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

0 8 . 1 Your PROGRAM SOURCE CODE for the entire subroutine `ValidJump`.
[2 marks]

0 8 . 2 SCREEN CAPTURE(S) showing the requested test including the list of possible moves **before** the jump and your test input, and then the board display **after** the jump.
[1 mark]

0 9 This question refers to the subroutine `PrintResult`. The rules of the game are to be amended. The game still ends when a player cannot make a move, but the winner is to be determined using a formula that calculates a score for each player. The winner is the player with the **lowest** score. The formula to calculate the score is shown in **Figure 5**.

Figure 5

Player's score = Number of that player's moves **minus** total number of player's pieces on the board **minus** number of that player's dames **multiplied by** 10

For example, if Player A made 40 moves and has 12 pieces on the board, and 3 of them are dames, then Player A's score is 40 – 12 – (3 × 10) = -2

Note that a dame is also considered to be a piece.

What you need to do:

Task 1
Create a new subroutine `CountNumberOfPieces`. This subroutine is to count the number of pieces a player has on the board.

Task 2
Amend the subroutine `PrintResult` to implement the formula given in **Figure 5** and output the score for each player and display the winner.

You must consider the possibility of a draw.

Task 3
- run the program
- enter `Y`
- load `game4.txt`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

0 9 . 1 Your PROGRAM SOURCE CODE for the entire subroutine `CountNumberOfPieces` and the entire subroutine `PrintResult`.

[9 marks]

0 9 . 2 SCREEN CAPTURE(S) showing all the output from the requested test including the board display.

[1 mark]

1 0 This question will further change the rules of the game.

When a piece is promoted to a dame, the player who the new dame belongs to now chooses **one** of the opponent's pieces. This piece is removed from the board and the dame is placed in the square the removed piece was in.

1 0 . 1 State the identifier of the data structure that now needs to be passed as a parameter into the subroutine `MoveDame`.

[1 mark]

What you need to do:

Task 1
Amend the subroutine `MoveDame`.

This subroutine is to:

- ask the player the piece ID of the opponent's piece they want to remove
- check that the piece is an opponent's piece and is on the board
- remove the opponent's piece
- return the coordinates for the new dame.

Task 2
Amend the calls to `MoveDame` from within the subroutine `MovePiece`.

You will need to amend the parameter list of the subroutine heading of `MovePiece` and the call to `MovePiece` from within the subroutine `MakeMove`.

Task 3
Test that the changes you have made work by conducting the following test:

- run the program
- enter `Y`
- load `game3.txt`
- move `a2` to row 7, column 0
- take piece `b1`

Task 4

- move `b5` to row 0, column 3
- take piece `a6`

Turn over ►

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

1 0 . 2 Your PROGRAM SOURCE CODE for the entire subroutine `MoveDame` and the entire subroutine `MakeMove`.

[9 marks]

1 0 . 3 SCREEN CAPTURE(S) showing the requested test including the board display after piece `b1` has been taken.

[1 mark]

1 0 . 4 SCREEN CAPTURE(S) showing the requested test including the board display after piece `a6` has been taken.

[1 mark]

1 1 State **two** examples of work that you would expect to undertake during the analysis stage of software development.

[2 marks]

1 2 The algorithm, represented using pseudo-code in **Figure 2**, outputs a series of integers or the message `No result`. The output depends upon the value entered by the user.

Figure 2

```
OUTPUT "Enter an integer greater than 1: "
INPUT X
Product ← 1
Factor ← 0
WHILE Product < X
  Factor ← Factor + 1
  Product ← Product * Factor
ENDWHILE
IF X = Product THEN
  Product ← 1
  FOR N ← 1 TO Factor
    Product ← Product * N
    OUTPUT N
  ENDFOR
ELSE
  OUTPUT "No result"
ENDIF
```

What you need to do:

Task 1
Write a program to implement the algorithm in **Figure 2**.

Task 2
Test that your program works:
- run your program, then enter the number `720`
- run your program, then enter the number `600`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

1 2 . 1 Your PROGRAM SOURCE CODE for **Task 1**.

[9 marks]

1 2 . 2 SCREEN CAPTURE(S) showing the tests described in **Task 2**.

[1 mark]

1 2 . 3 What is true for all valid inputs for `X` that output a number of numbers which is not true for all other valid inputs that output `No result`?

[1 mark]

1 3 This question extends the functionality of the **Skeleton Program**. A mirror image is to be produced from the loaded image. For example, **Figure 4** shows the image loaded from the `ascii.txt` data file. **Figure 5** shows the mirror image.

Figure 4

	0	1	2	3	4
0	#	#	#	@	@
1	@	@	@	A	A
2	B	B	!	!	!

Figure 5

	0	1	2	3	4
0	@	@	#	#	#
1	A	A	@	@	@
2	!	!	!	B	B

What you need to do:

Task 1
Write a new subroutine `MirrorImage` that produces a mirror image of the image loaded into `Grid` and stores it in a new data structure. The subroutine then displays the mirror image.

Task 2
Amend subroutine `DisplayMenu` to include the new option:

```
M – Mirror image
```

Task 3
Amend subroutine `Graphics` to call `MirrorImage` when the user chooses option `M`

Task 4
Test that the changes you have made work by conducting the following test:
- run the program
- enter `L`
- load **`image1`**
- enter `M`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

13.1 Your PROGRAM SOURCE CODE for the entire subroutines `MirrorImage`, `DisplayMenu` and `Graphics`

[6 marks]

13.2 SCREEN CAPTURE(S) showing the requested test, including the menu and the display of the mirror image.

[1 mark]

1 4 This question refers to the subroutine `LoadGreyScaleImage`

A greyscale image can contain a hidden message, which has been encrypted using a key. The message can be revealed by decrypting the image using the method described below.

If an image contains a hidden message, the key to use is the integer value of the digit at the end of the image title in the first line of the file. For example, if the image title is `image4` then the key is 4

When decrypting a message from an image file, each greyscale value is used in turn with the key to produce a result. The key is subtracted from the greyscale value and the result is interpreted according to **Table 2**.

Table 2 describes the rules that should be followed to determine what secret character, if any, is hidden in the greyscale image.

Table 2

Value of result	Interpretation
0	The decrypted character is the space character.
> 0 AND ≤ 26	The result is the position in the alphabet of the decrypted character.
> 26	The result means that the character was not part of the message that was encrypted, and should be represented by an underscore character, ie the _ character.

Example 1
Greyscale value is 1 and the key is 0
1 – 0 = 1
The decrypted character is A as this is the first letter in the alphabet. This should be appended to the hidden message.

Example 2
Greyscale value is 5 and the key is 3
5 – 3 = 2
The decrypted character is B as this is the second letter in the alphabet. This should be appended to the hidden message.

Example 3
Greyscale value is 7 and the key is 7
7 – 7 = 0
The decrypted character is a space as the result was 0. This should be appended to the hidden message.

Example 4
Greyscale value is 52 and the key is 1
52 – 1 = 51
The result is greater than 26 so the greyscale value was not an encrypted character and an underscore character should be added to the hidden message.

The hidden message is built by concatenating each of the decrypted characters.

What you need to do:

Task 1
Write a new subroutine, `FindSecretChar`, which takes as parameters the `PixelValue` and `Key`, where `Key` is the key used to decrypt the hidden character. The subroutine should return the decrypted character if the pixel value was a character from the message, otherwise it should return the underscore character.

Task 2
Amend the subroutine `LoadGreyScaleImage` to:

- get the key from the image title
- call the subroutine `FindSecretChar` in the appropriate place and build up the hidden message from the returned characters
- output the complete hidden message.

Task 3
Test that the changes you have made work by conducting the following test:

- run the program
- enter `L`
- load `greyscale`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

14.1 Your PROGRAM SOURCE CODE for the entire subroutines `FindSecretChar` and `LoadGreyScaleImage`

[9 marks]

14.2 SCREEN CAPTURE(S) showing the requested test, including the display of the message and the image.

[1 mark]

1 5 This question extends the functionality of the **Skeleton Program**. A new option, `C`, is to be added to compress an existing ASCII art image file using run-length encoding and store it in a new file.

The number of symbols in a run of symbols is to be counted and this count, separated by a comma from the symbol, is stored in the new file. Each count and symbol pair should be put on a new line.

For example, **Figure 6** shows the contents of the `ascii.txt` data file.

Figure 6

```
TestImage1,5,3,A
###@@@@@AABB!!!
```

The compressed file will contain the file header followed by several value pairs and the file type in the header should be changed from `A` to `C`, as shown in **Figure 7**.

Figure 7

```
TestImage1,5,3,C
3,#
5,@
2,A
2,B
3,!
```

You can assume that there are no newline characters in the ASCII art image file.

What you need to do:

Task 1
Write a new subroutine, `CompressFile`

This subroutine is to:
- ask the user for the name of the file to be compressed
- read the existing file header and change the file type to `C` to indicate that this is a compressed file
- create a new file with the existing filename preceded by `CMP`, for example, if the file read in was `image2` the new filename should be `CMPimage2`
- save the edited file header to the new file
- save each pair of symbol count and symbol separated by a comma on a new line to the new file.

Task 2
Amend subroutine `DisplayMenu` to include the new option:

```
C – Compress file
```

Task 3
Amend subroutine `Graphics` to call `CompressFile` when the user chooses option `C`

Task 4
Test that the changes you have made work by conducting the following test:

- run the program
- enter `C`
- enter the file name `image2`
- load the file `CMPimage2` into a text editor.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

15.1 Your PROGRAM SOURCE CODE for the entire subroutine `CompressFile`

[12 marks]

15.2 SCREEN CAPTURE(S) showing the requested test, including the menu and all of the contents of the file `CMPimage2` in the text editor.

[1 mark]

1 6 **Figure 3** shows an algorithm represented using pseudo-code.

Figure 3

```
C ← 0
D ← 0
S ← 0
T ← 0
WHILE C < 3 AND D < 3
  T ← T + 1
  N1 ← generate random integer between 1 and 6 inclusive
  N2 ← generate random integer between 1 and 6 inclusive
  OUTPUT N1, N2
  S ← S + N1 + N2
  IF N1 = 6 OR N2 = 6 THEN
    C ← C + 1
  ENDIF
  IF N1 = N2 THEN
    D ← D + 1
  ENDIF
ENDWHILE
A ← S DIV (T * 2)
OUTPUT C, D, A
```

The `DIV` operator calculates the whole number part resulting from an integer division, for example, `10 DIV 3 = 3`

Table 4 lists the `DIV` operators for each of the available programming languages. You should refer to the row for your programming language.

Table 4

Programming language	**DIV**
C#	/
Java	/
Pascal	div
Python	//
VB.NET	\

What you need to do:

Task 1
Write a program to implement the algorithm in **Figure 3**.

Task 2
Test that your program works:
- run your program.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

16.1 Your PROGRAM SOURCE CODE for **Task 1.**
[8 marks]

16.2 SCREEN CAPTURE(S) showing the test described in **Task 2**.
[1 mark]

1 7 This question adds further validation to the **Skeleton Program**.

When a puzzle is loaded, some cells already contain numbers. These cells are referred to as protected cells.

Figure 5 shows the numbers in the puzzle grid cells when **`puzzle1`** is first loaded. These are the protected cells for this puzzle.

Figure 5

```
   1   2   3   4   5   6   7   8   9
 |===.===.===|===.===.===|===.===.===|
1| 8 .   . 5 |   .   .   |   .   . 7 |
 |...........|...........|...........|
2| 9 .   .   | 5 .   . 4 |   .   .   |
 |...........|...........|...........|
3| 4 . 1 .   |   . 6 .   |   .   .   |
 |===.===.===|===.===.===|===.===.===|
4|   .   .   | 7 .   .   | 1 . 6 .   |
 |...........|...........|...........|
5| 1 .   .   | 4 .   . 6 |   .   . 3 |
 |...........|...........|...........|
6|   . 5 . 8 |   .   . 1 |   .   .   |
 |===.===.===|===.===.===|===.===.===|
7|   .   .   |   . 1 .   |   . 4 . 9 |
 |...........|...........|...........|
8|   .   .   | 2 .   . 7 |   .   . 1 |
 |...........|...........|...........|
9| 2 .   .   |   .   .   | 5 .   . 6 |
 |===.===.===|===.===.===|===.===.===|
```

The **Skeleton Program** is to be changed so that the user cannot change the contents of any protected cells.

The subroutine `SolvePuzzle` needs to be modified so that the user cannot change a digit in a protected cell but can still enter a digit into an empty cell or change a digit that they have previously entered.

What you need to do:

Task 1
Amend the subroutine `SolvePuzzle` so that it checks every cell reference entered by the user against the protected cell references in `Puzzle` and only allows the contents of the cell to be changed if the cell referenced by `CellInfo` is not a protected cell.

If a protected cell is referenced, an appropriate error message should be displayed.

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

- run your amended **Skeleton Program**
- enter `P`
- load **`puzzle1`**
- enter `S`
- enter `117`
- enter `323`
- enter `993`
- enter `853`
- enter `854`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

17.1 Your PROGRAM SOURCE CODE for the entire subroutine `SolvePuzzle`.

[7 marks]

17.2 SCREEN CAPTURE(S) showing the requested test described in **Task 2**.

The SCREEN CAPTURE(S) need(s) to show the puzzle grid before and after the described test.

[1 mark]

1 8 This question adds further validation to the **Skeleton Program**. The subroutine `SolvePuzzle` asks the user to enter coordinates and a digit. The **Skeleton Program** is to be changed so that if the digit entered by the user already exists in the referenced row, column or sub-grid, the digit cannot be used.

Figure 6 shows the numbers in the puzzle grid cells of **`puzzle1`** when first loaded.

Figure 6

```
   1   2   3   4   5   6   7   8   9
  |===.===.===|===.===.===|===.===.===|
 1| 8 .   . 5 |   .   .   |   .   . 7 |
  |...........|...........|...........|
 2| 9 .   .   | 5 .   . 4 |   .   .   |
  |...........|...........|...........|
 3| 4 . 1 .   |   . 6 .   |   .   .   |
  |===.===.===|===.===.===|===.===.===|
 4|   .   .   | 7 .   .   | 1 . 6 .   |
  |...........|...........|...........|
 5| 1 .   .   | 4 .   . 6 |   .   . 3 |
  |...........|...........|...........|
 6|   . 5 . 8 |   .   . 1 |   .   .   |
  |===.===.===|===.===.===|===.===.===|
 7|   .   .   |   . 1 .   |   . 4 . 9 |
  |...........|...........|...........|
 8|   .   .   | 2 .   . 7 |   .   . 1 |
  |...........|...........|...........|
 9| 2 .   .   |   .   .   | 5 .   . 6 |
  |===.===.===|===.===.===|===.===.===|
```

What you need to do:

Task 1
Write a new subroutine, `DuplicateDigit`, which takes `PuzzleGrid`, `Row`, `Column` and `Digit` as parameters. The subroutine should return `True` if the digit entered by the user is already present in the row, column or sub-grid of the cell the user has entered. Otherwise, the subroutine should return `False`.

Task 2
Amend the subroutine `SolvePuzzle` so that it uses `DuplicateDigit` to check the user input and only allows the digit to be placed into the puzzle grid if the digit is not already present in the row, column or sub-grid of the cell the user has entered. If the digit is already present, an appropriate message should be given to the user.

Task 3
Test that the changes you have made work by conducting the following test:
- run your amended **Skeleton Program**
- enter `L`
- load **`puzzle1`**
- enter `S`
- enter `178`
- enter `819`
- enter `124`
- enter `989`
- enter `555`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

1 8 . 1 Your PROGRAM SOURCE CODE for the entire subroutine `DuplicateDigit` and the entire subroutine `SolvePuzzle`.

[8 marks]

1 8 . 2 SCREEN CAPTURE(S) showing the requested test described in **Task 3**.

The SCREEN CAPTURE(S) need(s) to show the puzzle grid before and after the described test.

[1 mark]

1 9 This question extends the functionality of the **Skeleton Program**.

The user is to be allowed to clear the contents of the most recently changed cells. To do this, they will enter a negative integer when prompted to enter a row, column and digit. The integer will represent the number of cells to be cleared.

For example, if the user enters `-3`, the three most recently entered digits should each be replaced by a space in the puzzle grid and the `Answer` data structure should be updated.

If the number of cells to be cleared is greater than the number of entries in the `Answer` data structure, all digits entered by the user should be cleared.

What you need to do:

Task 1
Write a new subroutine, `ClearEntries`, that clears the required number of previously entered digit(s) from the puzzle grid and updates the `Answer` data structure as described above.

Task 2
Amend the subroutine `SolvePuzzle` to test user input for a negative value. If the user input is a negative integer, the subroutine should call `ClearEntries` with the necessary parameters and then re-display the puzzle grid.

Task 3
Test that the changes you have made work by conducting the following test:
- run your amended **Skeleton Program**
- enter `P`
- load **`puzzle1`**
- enter `S`
- enter `-x`
- enter `-1`
- enter `-5`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

1 9 . 1 Your PROGRAM SOURCE CODE for the entire subroutine `ClearEntries` and the entire subroutine `SolvePuzzle`.

[12 marks]

1 9 . 2 SCREEN CAPTURE(S) showing the requested test described in **Task 3**.

The SCREEN CAPTURE(S) need(s) to show the puzzle grid before and after the described test.

[1 mark]

2 0 The algorithm, represented using pseudo-code, in **Figure 3** outputs a series of integers. The output depends upon the value entered by the user.

Figure 3

```
OUTPUT "Enter an integer greater than 1: "
INPUT Number
X ← 2
Count ← 0
WHILE Number > 1
  Multi ← FALSE
  WHILE (Number MOD X) = 0
    IF NOT Multi THEN
      OUTPUT X
    ENDIF
    Count ← Count + 1
    Multi ← TRUE
    Number ← Number DIV X
  ENDWHILE
  X ← X + 1
ENDWHILE
OUTPUT Count
```

Table 3 lists the `MOD` and `DIV` operators for each of the available programming languages. You should refer to the row for your programming language.

Table 3

Programming language	**MOD**	**DIV**
C#	%	/
Java	%	/
Pascal	mod	div
Python	%	//
VB.NET	Mod	\

What you need to do:

Task 1
Write a program to implement the algorithm in **Figure 3**.

Task 2
Test that your program works:

- run your program, then enter the number 23
- run your program, then enter the number 25
- run your program, then enter the number 1260

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

20.1 Your PROGRAM SOURCE CODE for **Task 1**.

[9 marks]

20.2 SCREEN CAPTURE(S) showing the tests described in **Task 2**.

[1 mark]

2 1 This question extends the functionality of the **Skeleton Program**.

The functionality of the SKP opcode is to be changed so that it increments the value stored in the accumulator by 1

For example, if the accumulator contained 4, then executing the instruction

```
SKP
```

would change the value in the accumulator to 5

What you need to do:

Task 1
Amend the subroutine `ExecuteSKP` so that it adds 1 to the accumulator and then updates the status register if required. The `Registers` data structure should be passed as a parameter.

Task 2
Amend the call to `ExecuteSKP` in the `Execute` subroutine as required.

Task 3
Test that the changes you have made work by conducting the following test:

- run your amended **Skeleton Program**
- enter L
- load **prog2**
- enter A
- enter R

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

2 1 . 1 Your PROGRAM SOURCE CODE for the entire subroutine `ExecuteSKP` and the entire subroutine `Execute`.

[4 marks]

2 1 . 2 SCREEN CAPTURE(S) showing the requested test described in **Task 3**.

The SCREEN CAPTURE(S) only need to show all of Frame 0 and all of the final frame.

[1 mark]

2 2 This question adds validation to the **Skeleton Program**. The subroutine `EditSourceCode` asks the user to enter a line number. The line number must be an integer and the number of an existing line in the file containing the program.

For example, the last line (line 11) of `prog2.txt` is:

```
FINAL:      0
```

Therefore, a valid line number for `prog2.txt` would be an integer between 1 and 11 inclusive.

What you need to do:

Task 1
Amend the subroutine `EditSourceCode` to check that the line number entered by the user is a valid existing line number. If an invalid value is entered the subroutine should output an appropriate error message. The program must not continue until a valid line number has been entered.

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

- run your amended **Skeleton Program**
- enter `L`
- load **prog2**
- enter `E`
- enter `Q`
- enter `22`
- enter `0`
- enter `2`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

2 2 . 1 Your PROGRAM SOURCE CODE for the entire subroutine `EditSourceCode`.
[5 marks]

2 2 . 2 SCREEN CAPTURE(S) showing the requested test described in **Task 2**.
[1 mark]

2 3 This question adds a memory address check to the **Skeleton Program**.

Each time a JSR opcode is executed the return address is stored in memory. The memory location used for this could already contain an instruction or data.

If storing a return address will overwrite an instruction or data, then an appropriate error message should be output using the `ReportRunTimeError` subroutine. The original code in the `ExecuteJSR` subroutine should only be carried out if there is no error.

What you need to do:

Task 1

Amend the `ExecuteJSR` subroutine. If storing a return address would overwrite an instruction or data, the `ReportRunTimeError` subroutine should be called with an appropriate error message.

One method of completing this task would require the addition of extra parameter(s) to the `ExecuteJSR` subroutine.

Task 2

If your solution requires additional parameter(s) for the `ExecuteJSR` subroutine amend the call to `ExecuteJSR` in the `Execute` subroutine.

Task 3

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

- run your amended **Skeleton Program**
- enter L
- load **prog3**
- enter A
- enter R

Evidence that you need to provide

Include the following evidence in your Electronic Answer Document.

2 3 . 1 Your PROGRAM SOURCE CODE for the entire subroutine `ExecuteJSR` and the entire subroutine `Execute` if you have made changes in **Task 2**.

[4 marks]

2 3 . 2 SCREEN CAPTURE(S) showing the requested test described in **Task 3**.

The SCREEN CAPTURE(S) only need to show the final frame and the contents of the stack before execution terminates.

[1 mark]

2 4 This question extends the functionality of the **Skeleton Program**. The option to edit the source code is to be extended to allow lines to be deleted or inserted within a loaded source code program.

The options within the `EditSourceCode` subroutine should now be:

`E – Edit this line`

`D – Delete the current line`

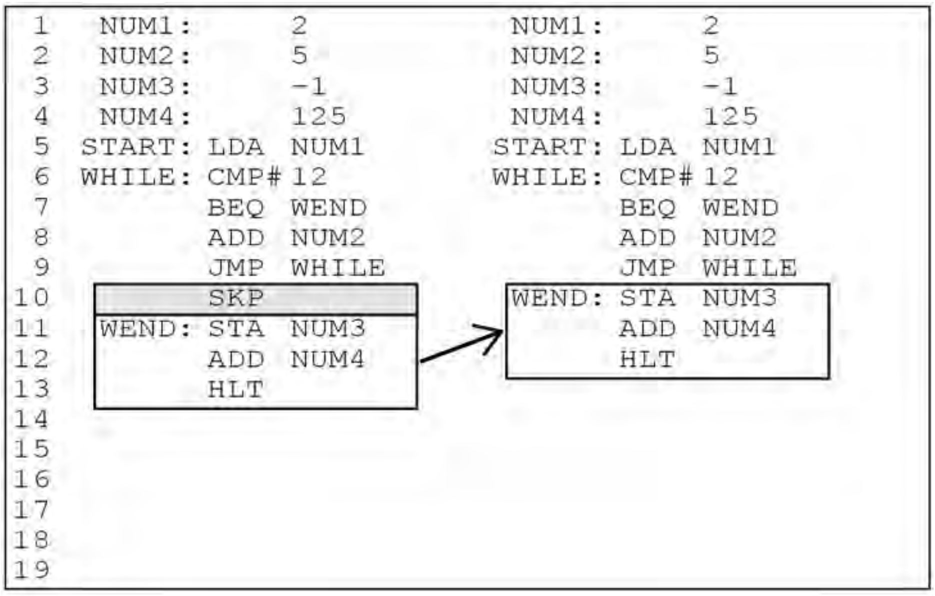
`I – Insert a new line above this line`

`C – Cancel edit`

Option D should delete the current line without changing the size of the data structure. The lines after the deleted line need to be moved within the data structure so that there is no gap in the source code.

Figure 4 shows an example where line 10 is being deleted. As a result lines 11 to 13 move. The size of the data structure does not change.

Figure 4



```
 1   NUM1:      2              NUM1:      2
 2   NUM2:      5              NUM2:      5
 3   NUM3:      -1             NUM3:      -1
 4   NUM4:      125            NUM4:      125
 5  START: LDA  NUM1          START: LDA  NUM1
 6  WHILE: CMP# 12            WHILE: CMP# 12
 7         BEQ  WEND                 BEQ  WEND
 8         ADD  NUM2                 ADD  NUM2
 9         JMP  WHILE                JMP  WHILE
10         SKP                WEND:  STA  NUM3
11   WEND: STA  NUM3                 ADD  NUM4
12         ADD  NUM4                 HLT
13         HLT
14
15
16
17
18
19
```

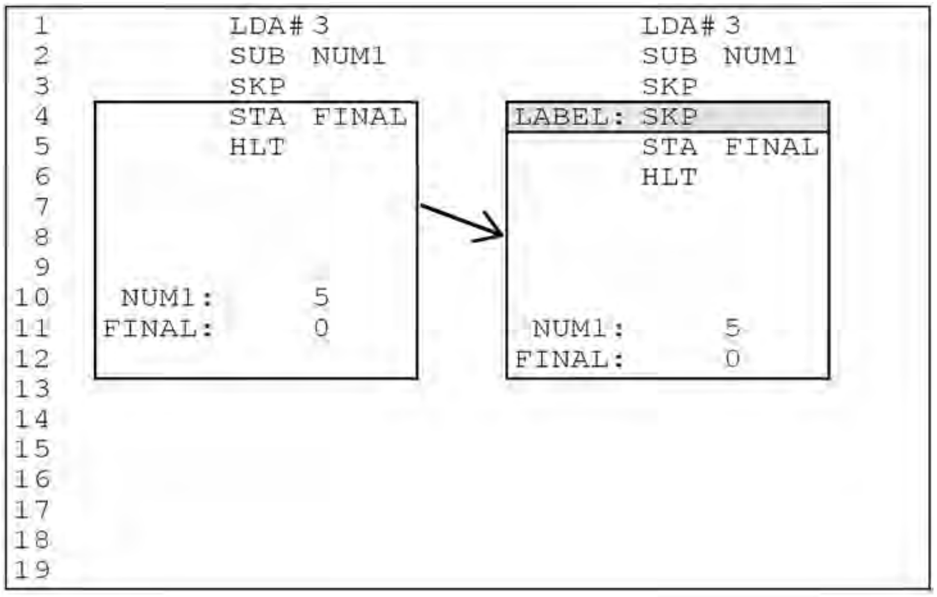
You **must write your own delete routine** and not use any built-in delete function that might be available in the programming language you are using.

Option `I` should allow the user to enter a new line to be inserted above the chosen line without changing the size of the data structure. The lines after the inserted line need to be moved within the data structure so they are not overwritten.

You should check that there is sufficient space in the data structure to accommodate a new line. If there is not sufficient space, an error message should be displayed.

Figure 5 shows an example where lines 4 to 11 move and a new line 4 is inserted. The size of the data structure does not change.

Figure 5



```
 1          LDA# 3
 2          SUB  NUM1
 3          SKP
 4          STA  FINAL
 5          HLT
 6
 7
 8
 9
10   NUM1:       5
11  FINAL:       0
12
13
14
15
16
17
18
19
```

```
 1          LDA# 3
 2          SUB  NUM1
 3          SKP
 4  LABEL:  SKP
 5          STA  FINAL
 6          HLT
 7
 8
 9
10
11   NUM1:       5
12  FINAL:       0
13
14
15
16
17
18
19
```

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

What you need to do:

Task 1
Amend the `EditSourceCode` subroutine to include the delete line option.

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

- run your amended **Skeleton Program**
- enter `L`
- load **prog1**
- enter `E`
- enter `10`
- enter `D`

Task 3
Amend the `EditSourceCode` subroutine to include the insert line option.

Task 4
Test that the changes you have made work by conducting the following test:

- run your amended **Skeleton Program**
- enter `L`
- load **prog2**
- enter `E`
- enter `4`
- enter `I`
- enter `LABEL: SKP`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

24.1 Your PROGRAM SOURCE CODE for the entire subroutine `EditSourceCode`.
[12 marks]

24.2 SCREEN CAPTURE(S) showing the requested test described in **Task 2**.

The SCREEN CAPTURE(S) only need to show the test from entering option E until after the edited source code has been displayed.
[1 mark]

24.3 SCREEN CAPTURE(S) showing the requested test described in **Task 4**.

The SCREEN CAPTURE(S) only need to show the test from entering option E until after the edited source code has been displayed.
[1 mark]

2 5 **Figure 2** shows an algorithm represented using pseudo-code.

Figure 2

```
OUT_NOLF "Enter an integer: "
INPUT Number1
OUT_NOLF "Enter another integer: "
INPUT Number2
IF Number1 > Number2 THEN
  Number ← Number1 DIV Number2
ELSE
  Number ← Number2 DIV Number1
ENDIF
Count ← 0
WHILE Count ≠ Number
  Count ← Count + 1
  IF (Count MOD 10) = 0 THEN
    OUT_NOLF "X"
  ELSE
    IF (Count MOD 5) = 0 THEN
      OUT_NOLF "V"
    ELSE
      OUT_NOLF "/"
    ENDIF
  ENDIF
ENDWHILE
```

The `OUT_NOLF` command displays the output without a line feed. The following series of `OUT_NOLF` commands will display ABC:

```
OUT_NOLF "A"
OUT_NOLF "B"
OUT_NOLF "C"
```

Table 2 lists the `MOD` and `DIV` operators for each of the available programming languages. You should refer to the row for your programming language.

Table 2

Programming language	**`MOD`**	**`DIV`**
C#	`%`	`/`
Java	`%`	`/`
Pascal	`mod`	`div`
Python	`%`	`//`
VB.NET	`Mod`	`\`

What you need to do:

Task 1
Write a program to implement the algorithm in **Figure 2**.

Task 2
Test that your program works:
- run your program
- enter 4
- enter 99

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

25.1 Your PROGRAM SOURCE CODE for **Task 1.**

[8 marks]

25.2 SCREEN CAPTURE(S) showing the test described in **Task 2**.

[1 mark]

2 6 This question extends the functionality of the **Skeleton Program**.

When a buyer enters the shop and sees a long queue, they will change their mind and leave the shop without buying anything. This is known as a **shun**.

The **Skeleton Program** is to be changed so that a buyer shunning the queue will cause the shop to open another till, subject to the maximum number of tills available not being exceeded.

The subroutine `BuyerArrives` needs to be modified so that a buyer arriving when the queue length is 5 will not join the queue. A running total of the number of buyers shunning the queue is to be kept.

After a buyer has shunned the queue, the shop will open another till, until the maximum number of tills available have opened.

At the end of the simulation the number of buyers that have shunned the queue should be output as part of the statistics.

What you need to do:

Task 1
Create a constant value to use as an index for a previously unused element of the data structure `Stats` in which to store the count of the number of shuns.

Task 2
Amend the subroutine `BuyerArrives` so that a buyer only joins the queue when fewer than 5 buyers are in the queue and shuns the queue if there are 5 buyers in the queue.

If a buyer shuns the queue, the number of shuns and the number of tills operating should be updated as appropriate and a suitable message showing the ID of the buyer that has shunned the queue should be displayed.

Task 3
Amend the subroutine `OutputStats` to output the total number of shuns with an appropriate message.

Task 4
Test that the changes you have made work by conducting the following test:

- run your amended **Skeleton Program**
- enter `Y`
- enter `50`
- enter `1`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

26.1 Your PROGRAM SOURCE CODE:

- that sets the value of the new constant
- for the entire subroutine `BuyerArrives`
- for the entire subroutine `OutputStats`.

[7 marks]

26.2 SCREEN CAPTURE(S) showing the requested test described in **Task 4**.

The SCREEN CAPTURE(S) **only** need to show the simulation statistics that are displayed when the simulation finishes.

[1 mark]

2 7 This question extends the functionality of the **Skeleton Program**, so that the length of time it takes to serve a buyer depends on the speed of the till operator. The speed of the till operator is the number of items they can handle in a time unit.

The simulation is to calculate the serving time for a buyer based upon the number of items in the buyer's basket and the speed of the till operator. The speed of each till operator needs to be stored for each till.

What you need to do:

Task 1
An extra value is required for each element of `Tills`, representing the speed of the operator at a till. Amend the declaration for `Tills`.

Task 2
Amend the subroutine `ResetDataStructures` so that the extra values are set so that till 0 has an operator speed of 7 and till 1 has an operator speed of 6. Each subsequent till should have its operator speed set to a value 1 less than the previous till.

Task 3
Amend the subroutine `ChangeSettings` so that it requires the user to set the till speed of each till in use. The user should be given information about which till they are setting the current till operator speed of and the current default value. `ChangeSettings` will require `Tills` as a parameter and, depending on your language, as a return value, so the call to this subroutine will also need amending.

Task 4
Amend the subroutine `CalculateServingTime` so that the serving time depends on the speed of the till operator at the till that is being used.

Task 5
Test that the changes you have made work by conducting the following test:
- run your amended **Skeleton Program**
- enter Y
- enter 6
- enter 3
- enter 3
- enter 2
- enter 1

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

27.1 Your PROGRAM SOURCE CODE for the entire subroutine `ResetDataStructures`, the entire subroutine `ChangeSettings` and the entire subroutine `CalculateServingTime` and any code that you have changed or added to the Skeleton Program.

[8 marks]

27.2 SCREEN CAPTURE(S) showing the requested test described in **Task 5**.

The SCREEN CAPTURE(S) **only** need to show the entire output for **time unit 5**.

[1 mark]

2 8 This question extends the functionality of the **Skeleton Program**.

Till 0 is not currently used. It is to be used as the express till that buyers with fewer than 10 items in their basket can use.

If till 0 is idle, the next buyer in the queue with fewer than 10 items in their basket comes out of the queue and gets served at till 0.

What you need to do:

Task 1
Write a new subroutine `ServeBuyerExpress` that:

- finds the first buyer in the queue who has fewer than ten items in their basket and **if there is one**:
  - o removes the buyer's details from the queue
  - o closes the gap in the queue
  - o displays the buyer's ID
  - o calls the `UpdateStats` subroutine for the express till
  - o calls the `CalculateServingTime` subroutine for the express till.

Task 2
Amend the subroutine `Serving` to check whether the express till is free and, if it is, call the subroutine `ServeBuyerExpress` before checking the availability of other tills.

Task 3
Amend the subroutine `OutputTillAndQueueStates` to output the data for till 0 as well as the data about the other tills which it already outputs.

Task 4
Test that the changes you have made work by conducting the following test:

- run your amended **Skeleton Program**
- enter `N`

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

2 8 . 1 Your PROGRAM SOURCE CODE for the entire subroutine `ServeBuyerExpress`, the entire subroutine `Serving` and the entire subroutine `OutputTillAndQueueStates`.

[12 marks]

2 8 . 2 SCREEN CAPTURE(S) showing the requested test described in **Task 4**.

The SCREEN CAPTURE(S) **only** need to show the entire output for **time unit 8**.

[1 mark]