13 The table shows assembly language instructions for a processor that has one register, the Accumulator (ACC).

Label	Instruction		Explanation
	Opcode	**Operand**	
	LDM	#n	Load the number n to the ACC
	LDD	<address>	Load the contents of the location at the given address to the ACC
	LDI	<address>	The address to be used is at the given address. Load the contents of this second address to the ACC
	ADD	<address>	Add the contents of the given address to the ACC
	ADD	#n	Add the number n to the ACC
	SUB	<address>	Subtract the contents of the given address from the ACC
	SUB	#n	Subtract the number n from the ACC
	STO	<address>	Store the contents of the ACC at the given address
<label>:		<data>	Gives a symbolic address <label> to the memory location with the contents <data>
# denotes a denary number, e.g. #123 <label> can be used in place of <address>			

The current contents of memory are:

Address	Contents
563	125
899	63

Write **assembly language** code, using **only** the given instruction set to:

- store the denary value `250` as labelled variable `X`
- store the value stored in location 563 as labelled variable `Y`
- add the value stored in variable `X` to the value stored in variable `Y`
- subtract the value stored in location 899 from the current value in the Accumulator
- store the result in variable `Total`.

Show the initialisation and values of the variables `X`, `Y` and `Total` in the table provided.

.....

.....

.....

.....

.....

.....

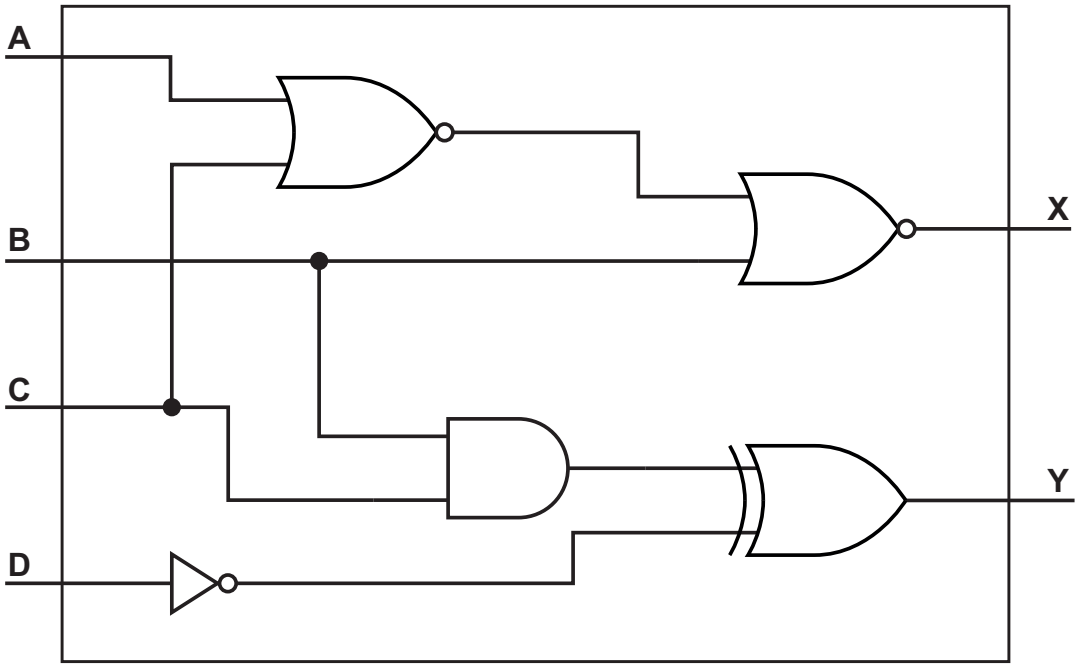
.....

.....

Label	Content

[7]

4 **(a)** Write the logic expressions for the following logic circuit.



X = ...

Y = ...

[2]

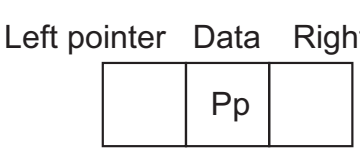
(b) Complete the truth table for the logic expression:

X = (**A** OR **B**) XOR (**B** OR **C**) XOR (NOT **A** NAND **C**)

A	B	C	Working space	X
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

[2]

11 **(a)** A linked list of nodes is used to store an ordered list of strings. Each node consists of the data, a left pointer and a right pointer.

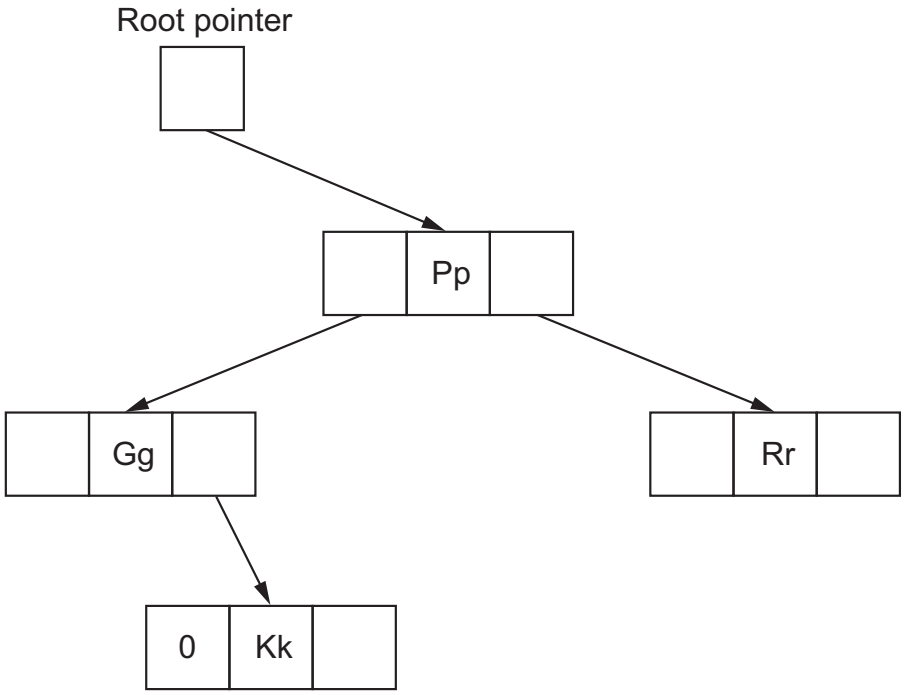


The linked list will be organised as a binary tree.

0 is used to represent a null pointer.

Complete the binary tree, including null pointers, to show how the data will be organised after the following strings have been added:

Aa, Mm, Ss, Xx



[4]

(b) A binary tree can be used to implement recursion.

Identify **one** feature of an algorithm that makes it beneficial to use recursion.
Give **one** example of an application that could use a recursive algorithm.

Feature .....

.....

Example .....

[2]

8 A program is being developed to implement a game for up to six players.

During the game, each player assembles a team of characters. At the start of the game there are 45 characters available.

Each character has four attributes, as follows:

Attribute	Examples	Comment
Player	0, 1, 3	The player the character is assigned to.
Role	Builder, Teacher, Doctor	The job that the character will perform in the game.
Name	Bill, Lee, Farah, Mo	The name of the character. Several characters may perform the same role, but they will each have a unique name.
Level	14, 23, 76	The skill level of the character. An integer in the range 0 to 100 inclusive.

The programmer has defined a record type to define each character.

The record type definition is shown in pseudocode as follows:

```
TYPE CharacterType
   DECLARE Player : INTEGER
   DECLARE Role : STRING
   DECLARE Name : STRING
   DECLARE Level : INTEGER
ENDTYPE
```

The `Player` field indicates the player to which the character is assigned (1 to 6). The field value is 0 if the character is **not** assigned to any player.

The programmer has defined a global array to store the character data as follows:

```
DECLARE Character : ARRAY[1:45] OF CharacterType
```

At the start of the game all record fields are initialised, and all `Player` field values are set to 0

The programmer has defined a program module as follows:

Module	Description

`Assign()`	• called with two parameters:
	  ○ an integer representing a player
	  ○ a string representing a character role
	• search the `Character` array for an unassigned character with the required role
	• If found, assign the character to the given player and output a confirmation message, for example:
	  `"Bill the Builder has been assigned to player 3"`
	• If no unassigned character with the required role is found, output a suitable message.

(a) Write pseudocode for module `Assign()`.

[7]

(b) A new module will store the contents of the `Character` array in a text file.

The module is defined as follows:

Module	Description
`Save()`	• form a string from each record with fields separated by the character `'^'` • write each string to a separate line of the new file named `SaveFile.txt`

Complete the pseudocode for module `Save()`.

```
PROCEDURE Save()
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

```
ENDPROCEDURE
```

(c) The program is changed and the record type definition is modified as follows:

```
TYPE CharacterType
   DECLARE Player : INTEGER
   DECLARE Role : STRING
   DECLARE Name : STRING
   DECLARE Level : INTEGER
   DECLARE Status : BOOLEAN
ENDTYPE
```

Describe how the additional Boolean field may be stored with the rest of the fields on one line of a text file.

.....

.....

.....

..... [2]

(d) The save operation is to be extended so that multiple files may be saved as the game progresses. This will allow the user to restore the game from any saved position. The filename must reflect the sequence in which the files are saved.

Describe a method that would allow multiple files to be saved **and** give an example of **two** consecutive filenames.

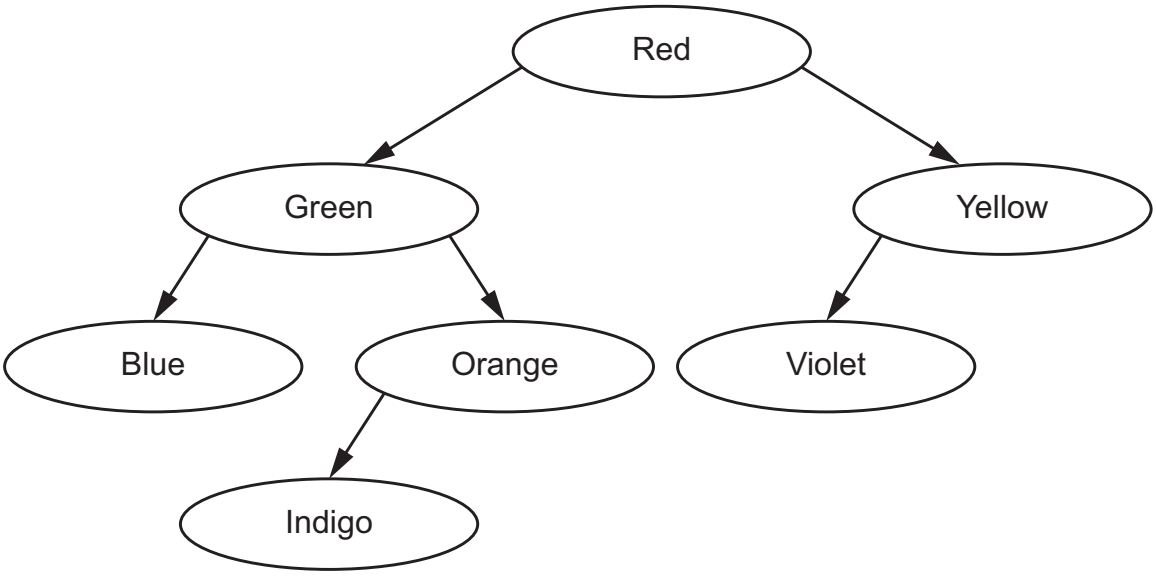
Method .....

.....

Example .....

.....
[2]

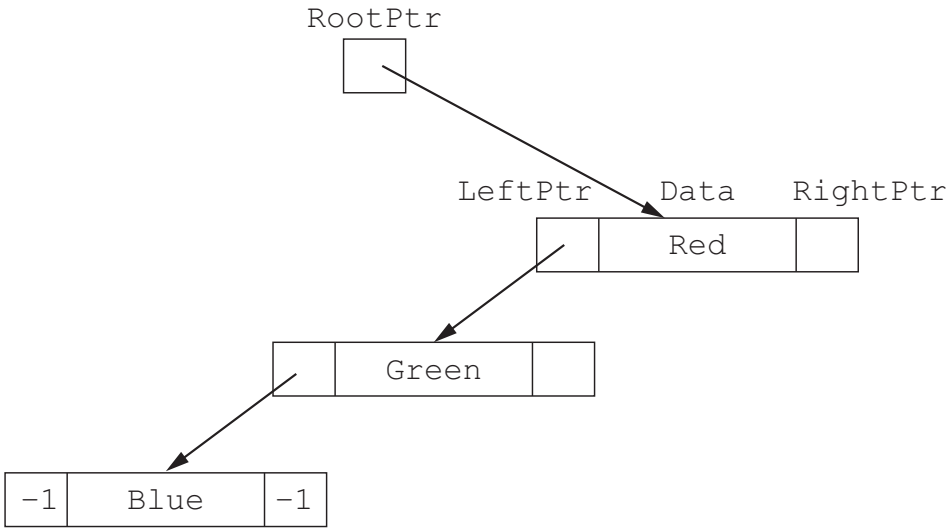
11 The following diagram shows an ordered binary tree.



(a) A linked list of nodes is used to store the data. Each node consists of a left pointer, the data and a right pointer.

–1 is used to represent a null pointer.

Complete this linked list to represent the given binary tree.



[4]

(b) A user-defined record structure is used to store the nodes of the linked list in part **(a)**.

Complete the diagram, using your answer for part **(a)**.

RootPtr

0

Index	LeftPtr	Data	RightPtr
0		Red	
1		Green	
2		Yellow	
3		Blue	
4		Orange	
5		Indigo	
6		Violet	
7			

FreePtr

[4]

(c) The linked list in part **(a)** is implemented using a 1D array of records. Each record contains a left pointer, data and a right pointer.

The following pseudocode represents a function that searches for an element in the array of records `BinTree`. It returns the index of the record if the element is found, or it returns a null pointer if the element is **not** found.

Complete the pseudocode for the function.

```
FUNCTION SearchTree(Item : STRING) .....

   NowPtr ← .....
   WHILE NowPtr <> -1

      IF ..... THEN
         NowPtr ← BinTree[NowPtr].LeftPtr
      ELSE
         IF BinTree[NowPtr].Data < Item THEN

            .....
         ELSE
            RETURN NowPtr
         ENDIF
      ENDIF
   ENDWHILE
   RETURN NowPtr
ENDFUNCTION
```

[4]

3 A linked list stores positive integer data in a 2D array. The first dimension of the array stores the integer data. The second dimension of the array stores the pointer to the next node in the linked list.

A linked list node with no data is initialised with the integer `-1`. These nodes are linked together as an empty list. A pointer of `-1` identifies that node as the last node.

The linked list can store 20 nodes.

The global 2D array `LinkedList` stores the linked list.

`LinkedList` is initialised as an empty list. The data in each node is initialised to `-1`. Each node's pointer stores the index of the next node. The last node stores the pointer value `-1`, which indicates it is the last node.

The global variable `FirstEmpty` stores the index of the first element in the empty list. This is the first node in the empty linked list when it is initialised, which is index `0`.

The global variable `FirstNode` stores the index of the first element in the linked list. There is no data in the linked list when it is initialised, so `FirstNode` is initialised to `-1`.

This diagram shows the content of the initialised array.

`FirstEmpty = 0`

`FirstNode = -1`

Index	Data	Pointer
0	-1	1
1	-1	2
2	-1	3
3	-1	4
4	-1	5
19	-1	-1

(a) Write program code for the main program to declare and initialise `LinkedList`, `FirstNode` and `FirstEmpty`.

Save your program as **Question3_N24**.

Copy and paste the program code into part **3(a)** in the evidence document.

[2]

(b) The procedure `InsertData()` takes **five** positive integers as input from the user and inserts these into the linked list.

Each data item is inserted at the front of the linked list.

The table shows the steps to follow depending on the state of the linked list:

Linked list state	Steps
not full	insert the data in the index pointed to by `FirstEmpty` change the pointer to the index pointed to by `FirstNode` change the values of `FirstNode` and `FirstEmpty`
full	end the procedure

Any node that is at the end of the linked list has a pointer of `-1`.

Write program code for `InsertData()`.

Save your program.

Copy and paste the program code into part **3(b)** in the evidence document.

[6]

(c) The procedure `OutputLinkedList()` outputs the data in the linked list in order by following the pointers from `FirstNode`.

(i) Write program code for `OutputLinkedList()`.

Save your program.

Copy and paste the program code into part **3(c)(i)** in the evidence document.

[2]

(ii) Amend the main program to call `InsertData()` and then `OutputLinkedList()`.

Save your program.

Copy and paste the program code into part **3(c)(ii)** in the evidence document.

[1]

(iii) Test your program with the test data:

5 1 2 3 8

Take a screenshot of the output.

Save your program.

Copy and paste the screenshot into part **3(c)(iii)** in the evidence document.

[1]

(d) The procedure `RemoveData()` removes a node from the linked list.

The procedure takes the data item to be removed from the linked list as a parameter.

The procedure checks each node in the linked list, starting with the node `FirstNode`, until it finds the node to be removed. This node is added to the empty list, and pointers are changed as appropriate. The procedure only removes the first occurrence of the parameter.

Assume that the data item being removed is in the linked list.

(i) Write program code for `RemoveData()`.

Save your program.

Copy and paste the program code into part **3(d)(i)** in the evidence document.

[5]

(ii) Amend the main program to:

- call `RemoveData()` with the parameter 5
- output the word `"After"`
- call `OutputLinkedList()`.

Save your program.

Copy and paste the program code into part **3(d)(ii)** in the evidence document.

[1]

(iii) Test your program with both sets of given test data:

Test data set 1: 5 6 8 9 5

Test data set 2: 10 7 8 5 6

Take a screenshot of each output.

Save your program.

Copy and paste the screenshot(s) into part **3(d)(iii)** in the evidence document.

[1]

6 A music player stores music in a digital form and has a display which shows the track being played.

(a) Up to 16 characters can be displayed. Track titles longer than 16 characters will need to be trimmed as follows:

- Words must be removed from the end of the track title until the resulting title is less than 14 characters.
- When a word is removed, the space in front of that word is also removed.
- Three dots are added to the end of the last word displayed when one or more words have been removed.

The table below shows some examples:

Original title	Display string															
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Bat out of Hull	**B**	**a**	**t**		**o**	**u**	**t**		**o**	**f**		**H**	**u**	**l**	**l**	
Bohemian Symphony	**B**	**o**	**h**	**e**	**m**	**i**	**a**	**n**	**.**	**.**	**.**					
Paperbook Writer	**P**	**a**	**p**	**e**	**r**	**b**	**o**	**o**	**k**		**W**	**r**	**i**	**t**	**e**	**r**
Chris Sings the Blues	**C**	**h**	**r**	**i**	**s**		**S**	**i**	**n**	**g**	**s**	**.**	**.**	**.**		
Green Home Alabama	**G**	**r**	**e**	**e**	**n**		**H**	**o**	**m**	**e**	**.**	**.**	**.**			

A function `Trim()` will:

- take a string representing the original title
- return the string to be displayed.

Assume:

- Words in the original title are separated by a single space character.
- There are no spaces before the first word or after the last word of the original title.
- The first word of the original title is less than 14 characters.

Write pseudocode for the function `Trim()`.

[7]

(b) Music is stored as a sequence of digital samples.

Each digital sample is a denary value in the range 0 to 99999999 (8 digits).

The samples are to be stored in a text file. Each sample is converted to a numeric string and 32 samples are concatenated (joined) to form a single line of the text file.

Each numeric string is 8 characters in length; leading '0' characters are added as required.

Example:

Sample	Denary value	String
1	456	"00000456"
2	48	"00000048"
3	37652	"00037652"
32	673	"00000673"

The example samples will be stored in the text file as a single line:

"00000456000000480003765 2...00000673"

(i) Identify one drawback of adding leading '0' characters to each numeric string.

.....

..... [1]

(ii) Suggest an alternative method of storing the samples which does **not** involve adding leading '0' characters but which would still allow each individual sample to be extracted.

.....

.....

..... [1]

(iii) State **one** drawback of the alternative method given in part **(b)(ii)**.

.....

..... [1]