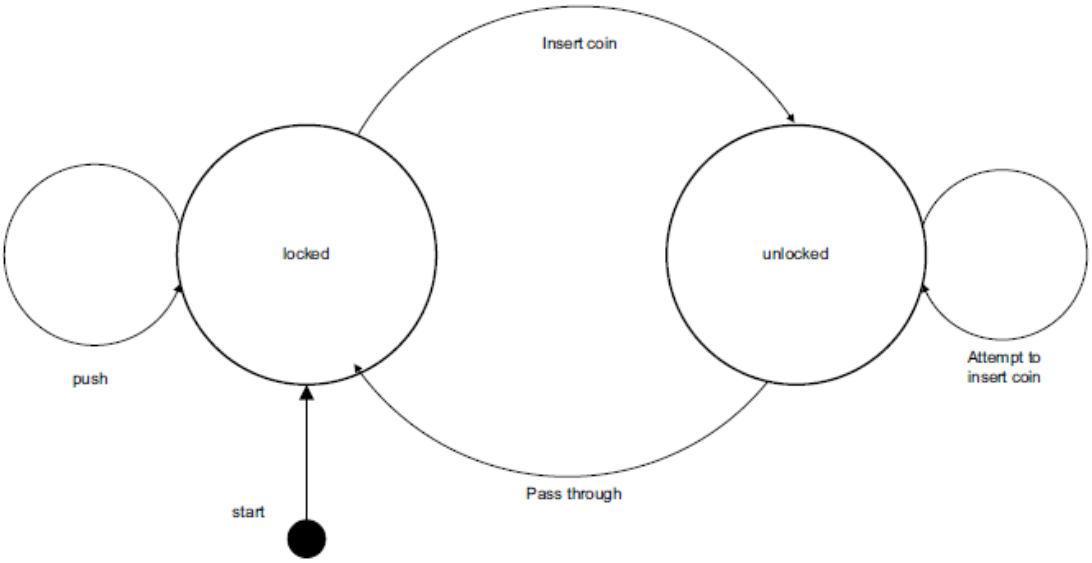
Answer

Answer 1

1



Mark as follows:
1 mark for both states correct
1 mark for each further label **[5]**

Answer 2

2 (a)
```
capital_city(santiago).
city_in_country(santiago, chile).
country_in_continent(chile,south_america).
city_visited(santiago).
```

accept in any order **[4]**

(b)
```
ThisCity =
manchester
london
```
[2]

(c)
```
countries_visited(ThisCountry)
IF
city_visited(ThisCity)                    1
AND                                       1
city_in_country(ThisCity, ThisCountry)    2
```
[4]

Answer 3

3 (a)

Conditions	goods totalling more than $20	Y	Y	Y	Y	N	N	N	N
	goods totalling more than $100	Y	Y	N	N	Y	Y	N	N
	have discount card	Y	N	Y	N	Y	N	Y	N
Actions	No discount				X	X	X	X	X
	5% discount		X	X					
	10% discount	X							
		1 mark	1 mark	1 mark	1 mark				

[4]

(b)

Conditions	goods totalling more than $20	Y	Y	Y	Y	N			
	goods totalling more than $100	Y	Y	N	N	-			
	have discount card	Y	N	Y	N	-			
Actions	No discount				X	X			
	5% discount		X	X					
	10% discount	X							

1 mark per column

[5]

Answer 4

5 (a) (i)

```
FOR ThisPointer ← 2 TO 10
    // use a temporary variable to store item which is to
    // be inserted into its correct location
    Temp ← NameList[ThisPointer]
    Pointer ← ThisPointer - 1

    WHILE (NameList[Pointer] > Temp) AND (Pointer > 0)
        // move list item to next location
        NameList[Pointer + 1] ← NameList[Pointer]
        Pointer ← Pointer - 1
    ENDWHILE

    // insert value of Temp in correct location
    NameList[Pointer + 1] Temp←
ENDFOR
```

1 mark for each gap filled correctly **[7]**

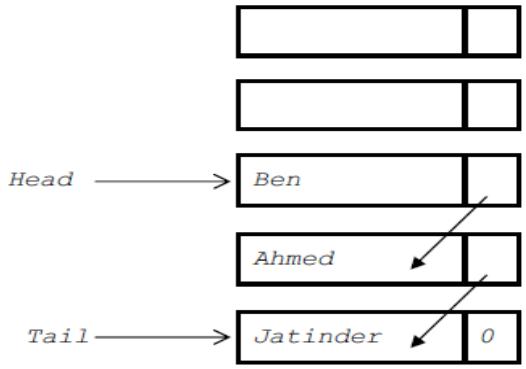
(ii) The outer loop (FOR loop) is executed 9 times (1 mark)
it is not dependant on the dataset (1 mark)

The Inner loop (WHILE loop) is not entered (1 mark)
as the condition is already false at the first encounter (1 mark) **[max 3]**

(b) (i) outer loop is executed 9 times (1 mark)
inner loop is executed 9 times (for each iteration of the outer loop) (1 mark)
not dependant on the dataset (1 mark) **[max 2]**

Answer 5

6 (a)



1 mark for Head and Tail pointers
1 mark for 3 correct items – linked as shown
1 mark for correct order with null pointer in last nod **[3]**

(b) (i)

Queue

HeadPointer		Name	Pointer
0	[1]		2
	[2]		3
TailPointer	[3]		4
0	[4]		5
	[5]		6
FreePointer	[6]		7
1	[7]		8
	[8]		9
	[9]		10
	[10]		0

Mark as follows:

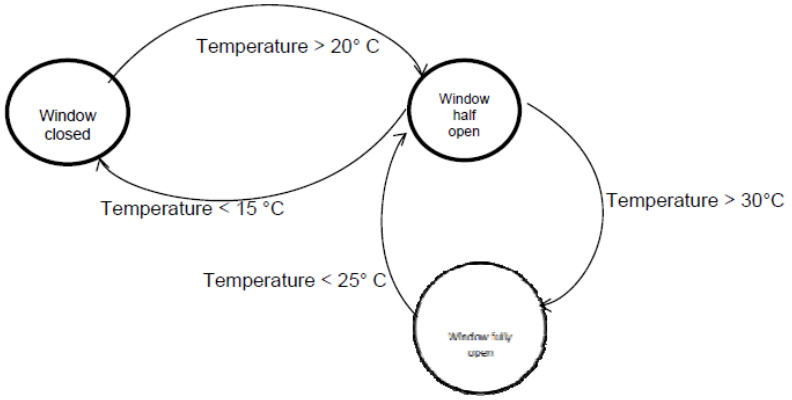
HeadPointer =0 & TailPointer = 0
FreePointer assigned a value
Pointers[1] to [9] links the nodes together
Pointer[10] = 'Null' **[4]**

(ii)

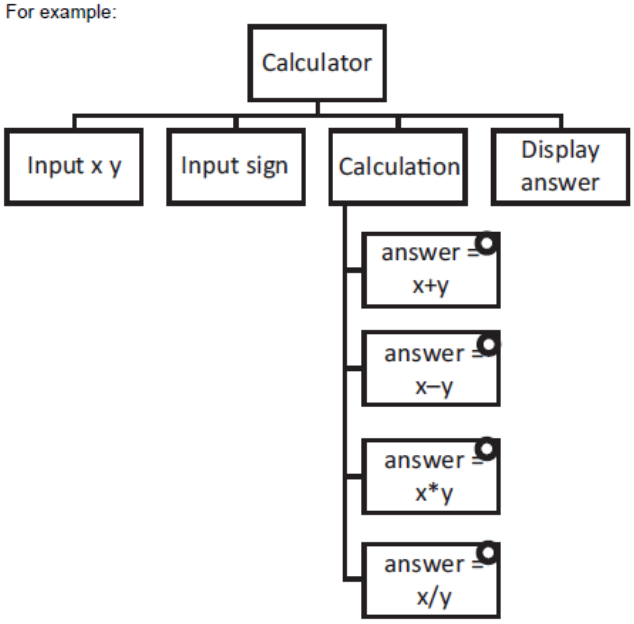
```
PROCEDURE RemoveName()
   // Report error if Queue is empty
   IF HeadPointer = 0
      THEN
         Error
      ELSE
         OUTPUT Queue[HeadPointer].Name
         // current node is head of queue
         CurrentPointer ← HeadPointer
         // update head pointer
         HeadPointer ← Queue[CurrentPointer].Pointer
         //if only one element in queue,then update tail pointer
         IF HeadPointer = 0
            THEN
               TailPointer ← 0
         ENDIF
            // link released node to free list
            Queue[CurrentPointer].Pointer ← FreePointer
         FreePointer ← CurrentPointer
   ENDIF
ENDPROCEDURE
```

[max 6]

Answer 6

1	1 mark for each completed statement  	7

Answer 7

2(a)(i)	• Asterisk (*) in the corner/top of the box(es)	1
2(a)(ii)	• Circle (○) in the corner/top of box(es)	1
2(b)	1 mark per bullet • Inputting 2 numbers, stored in x and y • Inputting sign Selection used for all four calculations • .. underneath an appropriate box at level 1 • Displaying the answer For example:  	5

Answer 8

3(a)	1 mark per clause • `person(mimi).` • `food(lettuce).` • `likes(mimi, chocolate).` • `dislikes(mimi, sushi).` • `dislikes(mimi, lettuce).`	5
3(b)	1 mark per answer `chocolate, pizza`	2

3(c)	1 mark per bullet
	• `might_like(B,A)`
	• `Person(B)`
	• `Food(A)`
	• `AND`
	• `AND NOT`
	• `Dislikes` predicate
	
	For example:
	
	`might_like(B, A).`
	
	`IF person(B) AND food(A)`
	
	`AND NOT(dislikes(B, A)).`
		6

Answer 9

4(a) — 11

Label	Op code	Operand	Comment	Marks
START:	LDM	#63	// load ASCII value for '?'	
	OUT		// OUTPUT '?'	1
	IN		// input GUESS	1
	CMP	LETTERTOGUESS	// compare with stored letter	1
	JPE	GUESSED	// if correct guess, go to GUESSED	1
	LDD	ATTEMPTS	// increment ATTEMPTS	1
	INC	ACC		1
	STO	ATTEMPTS		1
	CMP	#9	// is ATTEMPTS = 9 ?	1
	JPE	ENDP	// if out of guesses, go to ENDP	1
	JMP	START	// go back to beginning of loop	1
GUESSED:	LDM	#42	// load ASCII for '*'	
	OUT		// OUTPUT '*'	1
ENDP:	END		// end program	
ATTEMPTS:	0			
LETTERTOGUESS:	'a'			

4(b)					10

Label	Opcode	Operand	Comment	Mark
START:	LDR	#0	// initialise the Index Register	1
LOOP:	LDX	NUMBERS	// load the value from NUMBERS	1 (LOOP) + 1(LDX NUMBERS)
	LSL	#2	// multiply by 4	1 (LSL) + 1 (#2)
	STX	NUMBERS	// store the new value in NUMBERS	1
	INC	IX	// increment the Index Register	1
	LDD	COUNT		
	INC	ACC	// increment COUNT	1
	STO	COUNT		
	CMP	#5	// is COUNT = 5 ?	1
	JPN	LOOP	// repeat for next number	1
ENDP:	END			
COUNT:	0			
NUMBERS:	22			
	13			
	5			
	46			
	12			

Answer 10

5(a)(i)	PERT / GANTT	1
5(a)(ii)	1 mark per bullet to max 3 For example: • Calculate total minimum time required for project • Identify milestones • Task dependencies • Provides the critical path analysis • Identify which tasks need to be prioritised • Determine when to begin specific tasks/stages • Identify slack time • Identify when resources need allocating • Identify tasks that can be completed in parallel	3
5(b)(i)	Integration	1
5(b)(ii)	Beta / acceptance	1

Answer 11

1(a) — 1 mark per shaded group — **4**

		Column							
		1	2	3	4	5	6	7	8
Conditions	Grade C in Computer Science	Y	Y	Y	Y	N	N	N	N
	Grade C in Maths	Y	Y	N	N	Y	Y	N	N
	Grade C in Science	Y	N	Y	N	Y	N	Y	N
Actions	Take Computer Science	Y	Y	Y	Y	Y	Y		
	Take Maths	Y	Y			Y	Y		
	Take Physics	Y				Y			

1(b) — 1 mark per column — **3**

		Column							
		S	T	U	V	W	X	Y	Z
Conditions	Grade C in Computer Science	Y	–	–					
	Grade C in Maths	–	Y	Y					
	Grade C in Science	–	–	Y					
Actions	Take Computer Science	Y	Y						
	Take Maths		Y						
	Take Physics			Y					

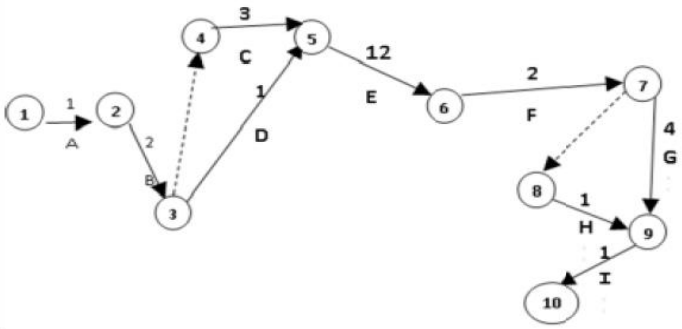
1(c) — **3**

For example:

- (Column S) combining 1,2,3,4…
- … because they only need CS to take CS // Maths and Science do not matter
- (Column T) combining 1,2,5,6…
- … because CS does not matter if it is Y/N
- (Column U) combining 1,5…
- …because CS does not matter if it is Y/N

Answer 12

2(a) — 1 mark for each correct line, duration and activity. — **7**



2(b) — **1**

- Dummy activity

Answer 13

3(a)	1 mark per clause • `room(corridor).` • `furniture(table).` • `furniture(lamp).` • `located(table, corridor).` • `located(lamp, corridor).`	5
3(b)	• `master_bedroom` • `spare_bedroom`	2
3(c)(i)	1 mark per bullet to max 2 • The first clause <u>only</u> says the nursery is next to the master bedroom • … but not that the master bedroom is next to the nursery • The second clause <u>only</u> says the master bedroom is next to the nursery • … but not that the nursery is next to the master bedroom • Goal to find rooms adjacent to master bedroom would not return nursery • … Example. `FindNextTo(X, master_bedroom)` • It is a two-way relationship	2
3(c)(ii)	1 mark per bullet • `room(main_bathroom).` • `nextTo(corridor, main_bathroom).` • `nextTo(main_bathroom, corridor).`	3

3(d)	1 mark per bullet
	• `canBeMovedTo(B,A)`
	• `Furniture(B)`
	• `Room(A)`
	• `AND / ,`
	• `AND NOT / , NOT`
	• `Located(B,A)`
	
	Example:
	
	`canBeMovedTo(B,A)`
	
	`IF furniture(B) AND room(A)`
	
	`AND NOT(located(B,A)).`
		6

Answer 14

4(a)	1 mark per item in bold
	
	`FOR Pointer ← 1 TO (Max - 1)`
	
	`  ItemToInsert ← Numbers[Pointer]`
	
	`  CurrentItem ← Pointer`
	
	`  WHILE (CurrentItem > 0) AND (Numbers[CurrentItem - 1] > ItemToInsert)`
	
	`  Numbers[CurrentItem] ← Numbers[CurrentItem - 1]`
	
	`  CurrentItem ← CurrentItem - 1`
	
	`  ENDWHILE`
	
	`  Numbers[CurrentItem] ← ItemToInsert`
	
	`ENDFOR`
		4

4(b)	• The size of the array // value of Max • How ordered the items already are	2

Answer 15

5(a) Max 10 — 10

Label	Op code	Operand	Comment	Marks
START:	LDR	#0	// initialise Index Register	
LOOP:	LDX	LETTERS	// load LETTERS	1
	CMP	LETTERTOFIND	// is LETTERS = LETTERTOFIND ?	1
	JPN	NOTFOUND	// if not, go to NOTFOUND	1
	LDD	FOUND	// increment FOUND	1
	INC	ACC		1
	STO	FOUND		1
NOTFOUND:	LDD	COUNT	//increment COUNT	1
	INC	ACC		
	STO	COUNT		
	CMP	#6	// is COUNT = 6 ?	1
	JPE	ENDP	// if yes, end	1
	INC	IX	// increment Index Register	1
	JMP	LOOP	// go back to beginning of loop	1
ENDP:	END		// end program	
LETTERTOFIND:	'x'			
LETTERS:	'd'			
	'u'			
	'p'			
	'l'			
	'e'			
	'x'			
COUNT:	0			
FOUND:	0			

5(b) — 10

Label	Op Code	Operand		Comment
START:	LDR	#0	// initialise the Index Register	1
LOOP:	LDX	VALUES	// load the value from VALUES	1(loop) + 1(LDX Values)
	LSR	#3	// divide by 8	1 (LSR) + 1 (#3)
	STX	VALUES	// store the new value in VALUES	1
	INC	IX	// increment the Index Register	1
	LDD	REPS	// increment REPS	1
	INC	ACC		
	STO	REPS		
	CMP	#6	// is REPS = 6 ?	1
	JPN	LOOP	// repeat for next value	1
	END			
REPS:	0			
VALUES:	22			
	13			
	5			
	46			
	12			
	33			

Answer 16

1(a) — **9**

Label	Op code	Operand	Comment	
START:	IN		// INPUT character	1
	STO	CHAR1	// store in CHAR1	
	IN		// INPUT character	1
	STO	CHAR2	// store in CHAR2	
	LDD	CHAR1	// initialise ACC to ASCII value of CHAR1	1
LOOP:	OUT		//output contents of ACC	1+1
	CMP	CHAR2	// compare ACC with CHAR2	1
	JPE	ENDFOR	// if equal jump to end of FOR loop	1
	INC	ACC	// increment ACC	1
	JMP	LOOP	// jump to LOOP	1
ENDFOR:	END			
CHAR1:				
CHAR2:				

1(b) — **6**

Label	Op code	Operand	Comment	
START:	LDD	NUMBER1		1
	XOR	MASK	// convert to one's complement	1
	INC	ACC	// convert to two's complement	1
	STO	NUMBER2		1
	END			
MASK:	B11111111		// show value of mask in binary here	1
NUMBER1:	B00000101		// positive integer	
NUMBER2:	B11111011		// show value of negative equivalent	1

Answer 17

2(a) — **1**

- A pointer that doesn't point to another node/other data/address // indicates the end of the branch

2(b) — **2**

one mark per bullet
- node with 'Athens' linked to left pointer of Berlin (ignore null pointer)
- null pointers in left and right pointers of Athens

2(c)(i) — **5**

RootPointer: 0

FreePointer: 7

1 mark

	LeftPointer	Tree Data	RightPointer
[0]	2	Dublin	1
[1]	-1/∅	London	3
[2]	6	Berlin	5
[3]	4	Paris	-1/∅
[4]	-1/∅	Madrid	-1/∅
[5]	-1/∅	Copenhagen	-1/∅
[6]	-1/∅	Athens	-1/∅
[7]	8		-1/∅
[8]	9		-1/∅
[9]	-1/∅		-1/∅

2(c)(ii) — **2**

- –1
- It is not the number for any node.

2(d)(i)	TYPE Node
	  LeftPointer : INTEGER
	  RightPointer : INTEGER 1
	  Data : STRING
	ENDTYPE
	DECLARE Tree : ARRAY[0 : 9] OF Node 1
	DECLARE FreePointer : INTEGER
	DECLARE RootPointer : INTEGER
	
	PROCEDURE CreateTree()
	  DECLARE Index : INTEGER
	  RootPointer ← -1 1
	  FreePointer ← 0 1
	  FOR Index ← 0 TO 9 // link nodes
	  Tree[Index].LeftPointer ← Index + 1 1
	  Tree[Index].RightPointer ← -1 1
	  ENDFOR
	  Tree[9].LeftPointer ← -1 1
	ENDPROCEDURE
		7

2(d)(ii)	PROCEDURE AddToTree(ByVal NewDataItem : STRING)
	// if no free node report an error
	  IF FreePointer = -1 1
	  THEN
	  ERROR("No free space left")
	  ELSE // add new data item to first node in the free list
	  NewNodePointer ← FreePointer
	  Tree[NewNodePointer].Data ← NewDataItem 1
	  // adjust free pointer
	  FreePointer ← Tree[FreePointer].LeftPointer 1
	  // clear left pointer
	  Tree[NewNodePointer].LeftPointer ← -1 1
	  // is tree currently empty ?
	  IF RootPointer = -1 1
	  THEN // make new node the root node
	  RootPointer ← NewNodePointer 1
	  ELSE // find position where new node is to be added
	  Index ← RootPointer
	  CALL FindInsertionPoint(NewDataItem, Index, Direction)
		8

```
                IF Direction = "Left"
                   THEN   // add new node on left
          Tree[Index].LeftPointer ← NewNodePointer                    1
                   ELSE   // add new node on right
          Tree[Index].RightPointer ← NewNodePointer                   1
                ENDIF
            ENDIF
      ENDIF
ENDPROCEDURE
```

2(e) **1 mark per bullet** — **5**

- test for base case (null/-1)
- recursive call for left pointer
- output data
- recursive call for right pointer
- order, visit left, output, visit right

```
   IF Pointer <> NULL                                                 1
      THEN
         TraverseTree(Tree[Pointer].LeftPointer)                      1
         OUTPUT Tree[Pointer].Data                                    1 + 1
         TraverseTree(Tree[Pointer].RightPointer)                     1
   ENDIF
ENDPROCEDURE
```

Answer 18

1(a) — **8**

Label	Op code	Operand	Comment	
START:	IN		// INPUT character	1
	STO	CHAR	// store in CHAR	
	LDM	#65	// Initialise ACC (ASCII value for 'A' is 65)	1
LOOP:	OUT		// OUTPUT ACC	1 + 1
	CMP	CHAR	// compare ACC with CHAR	1
	JPE	ENDFOR	// if equal jump to end of FOR loop	1
	INC	ACC	// increment ACC	1
	JMP	LOOP	// jump to LOOP	1
ENDFOR:	END			
CHAR:				

1(b)	START:	LDD	NUMBER		1	7
		AND	MASK	// set to zero all bits except sign bit	1	
		CMP	#0	// compare with 0	1	
		JPN	ELSE	// if not equal jump to ELSE	1	
	THEN:	LDM	#80	// load ACC with 'P' (ASCII value 80)	1	
		JMP	ENDIF			
	ELSE:	LDM	#78	// load ACC with 'N' (ASCII value 78)		
	ENDIF:	OUT		//output character	1	
		END				
	NUMBER:	B00000101		// integer to be tested		
	MASK:	B10000000		// show value of mask in binary here	1	

Answer 19

2(a)	1 mark for the declaration of the array. 1 mark for assigning a 0 to Customer ID (CustomerID ← 0) 1 mark for getting the correct record (Customer[x].) 1 mark for setting up a loop to go <u>from 0 to 199</u> `DECLARE Customer : ARRAY[0 : 199] OF CustomerRecord` 1 `FOR x ← 0 TO 199` 1 `Customer[x].CustomerID ← 0` 1+1 `ENDFOR`	4

2(b)(i)		9

```
PROCEDURE InsertRecord(BYVAL NewCustomer : CustomerRecord)
   TableFull ← FALSE
   // generate hash value
   Index ← Hash(NewCustomer.CustomerID)                    1
   Pointer ← Index   // take a copy of index

   // find a free table element
   WHILE Customer[Pointer].CustomerID > 0                  1
      Pointer ← Pointer + 1                                1
      // wrap back to beginning of table if necessary
      IF Pointer > 199                                     1
         THEN
            Pointer ← 0                                    1
      ENDIF
      // check if back to original index
      IF Pointer = Index                                   1
         THEN
            TableFull ← TRUE
      ENDIF
   ENDWHILE
   IF NOT TableFull                                        1
      THEN
         Customer[Pointer] ← NewCustomer                   1
      ELSE
         OUTPUT "Error"                                    1
   ENDIF
ENDPROCEDURE
```

2(b)(ii)		9

```
FUNCTION SearchHashTable(BYVAL SearchID : INTEGER) RETURNS
INTEGER
  // generate hash value
  Index ← Hash(SearchID)                                   1
  // check each record from index until found or not there
  WHILE (Customer[Index].CustomerID <> SearchID)           1
     AND (Customer[Index].CustomerID > 0)                  1
     Index ← Index + 1                                     1
  // wrap if necessary
     IF Index > 199                                        1
        THEN
           Index ← 0                                       1
     ENDIF
  ENDWHILE
  // has customer ID been found?

  IF Customer[Index].CustomerID = SearchID                 1
     THEN
        RETURN Index                                       1
     ELSE
        RETURN -1                                          1
  ENDIF
ENDFUNCTION
```

2(b)(iii)	A record out of place may not be found	1

Answer 20

3		7

```
FUNCTION Find(BYVAL Name : STRING,
              BYVAL Start : INTEGER,
              BYVAL Finish : INTEGER) RETURNS INTEGER
   // base case
   IF Finish < Start                                  1
      THEN
         RETURN -1
      ELSE
         Middle ← (Start + Finish) DIV 2                1
         IF NameList[Middle] = Name                     1
            THEN
               RETURN Middle                            1
            ELSE    // general case
               IF SearchItem > NameList[Middle]         1
                  THEN
                     Find(Name, Middle + 1, Finish)     1
                  ELSE
                     Find(Name, Start, Middle - 1)      1
               ENDIF
         ENDIF
   ENDIF
ENDFUNCTION
```

Answer 21

2 [6]

	(i) Alpha testing	(ii) Beta testing
Who	In house testers / developers / programmers	(potential) (end) user(s)/client(s)
When	Near the end of development // program is nearly fully-usable // after integration and before beta	Before general release of software // passed Alpha testing
Purpose	To find errors not found in earlier testing // ensure ready for beta testing	For constructive comments/ feedback // to test in real-life scenarios/situations/ environments // ensure it is ready for release // ensure it meets users' needs

Answer 22

3 (a) (i) 1 mark per bullet to max 2: [2]

- 11011111
- AND

(ii) 1 mark per bullet to max 2: [2]

- 00100000
- OR

(b) 1 mark per line

START:	LDR	#0	// initialise index register to zero	1
	LDX	WORD	// get first character of WORD	1
	AND	MASK1	// ensure it is in upper case using MASK1	1
	OUT		// output character to screen	
	INC	IX	// increment index register	1
	LDM	#1	// load 1 into ACC	1
	STO	COUNT	// store in COUNT	1
LOOP:	LDX	WORD	// load next character from indexed address WORD	1
	OR	MASK2	// make lower case using MASK2	1
	OUT		// output character to screen	
	LDD	COUNT	// increment COUNT	1
	INC	ACC	//	
	STO	COUNT	//	
	CMP	LENGTH	// is COUNT = LENGTH?	1
	JPN	LOOP	// if FALSE - jump to LOOP	1
	END		// end of program	1
COUNT:	0			
MASK1:	B11011111		// bit pattern for upper case	1
MASK2:	B00100000		// bit pattern for lower case	
LENGTH:	4			
WORD:	B01100110		//ASCII code in binary for 'f'	
	B01101000		//ASCII code in binary for 'r'	
	B01000101		//ASCII code in binary for 'E'	
	B01000100		//ASCII code in binary for 'D'	

[max 12]

Answer 23

4 (a) (i) 1 mark per feature to max 3 [3]

e.g.

- auto-indent
- auto-complete / by example
- colour-coded keywords/ strings/ comments/ built-in functions/ user-defined function names pop-up help
- can set indent width
- expand/collapse subroutines/code
- block highlighting

incorrect syntax highlighting/ underlining // dynamic syntax checker

(ii) Read and mark the answer as one paragraph. Mark a how and a when anywhere in the answer.
1 mark for when, 1 mark for how.

e.g.

When:
- the error has been typed
- when the program is being run/compiled/interpreted

How:
- highlights/underlines

displays error message/pop-up

(iii) 1 mark for identifying the correct line, 1 mark for writing the corrected line

A - Line 5	B - Line 6	C - Line 5	
`for i in range(Max-1):`	`FOR i := 1 TO (Max-1) DO`	`For i = 0 To (Max - 1)`	[1]

(b) (i) Python: compiled/interpreted [1]
VB.NET: compiled
Pascal:compiled/interpreted
Delphi: compiled/interpreted

Answer 24

2 (a) (i) 1 mark per feature to max 3 **[3]**

e.g.

- auto-indent
- auto-complete / by example
- colour-coded keywords/ strings/ comments/ built-in functions/ user-defined function names
- pop-up help
- can set indent width
- expand/collapse subroutines/code
- block highlighting

incorrect syntax highlighting/underlining //dynamic syntax checker

(ii) Read and mark the answer as one paragraph. Mark a 'how' and a 'when' anywhere in the answer. **[2]**

1 mark for when, 1 mark for how.

e.g.

When:

- the error has been typed
- when the program is being run/compiled/interpreted

How:

- highlights/underlines

displays error message/pop-up

(iii)

A	B	C	
Line 3	Line 5	Line 4	**[1]**
`while (Index == -1) & (Low <= High):`	`WHILE (Index = -1) AND (Low <= High) DO`	`DO WHILE (Index = -1) AND (Low <= High)`	**[1]**

(b) (i) Python: compiled/interpreted **[1]**

VB.NET: compiled

Pascal: compiled/interpreted

Delphi: compiled/interpreted

(ii)

Logic error	Logic error	Logic error	
			[1]
`11 return(Index)`	`14 Result := Index;`	`14 BinarySearch = Index`	**[1]**

(iii) 1 mark for each name, 1 for each description **[4]**

- breakpoint
- a point where the program can be halted to see if the program works at this point

- stepping / step through
- executes one statement at a time and then pauses to see the effect of each statement

- variable watch window
- observe how variables changed during execution

Answer 25

3

START:	LDR	#0	// initialise index register to zero	[1]
	LDM	#0	// initialise COUNT to zero	[1]
	STO	COUNT		
LOOP1:	LDX	NAME	// load character from indexed address NAME	[1]
	OUT		// output character to screen	[1]
	INC	IX	// increment index register	[1]
	LDD	COUNT	// increment COUNT starts here	
	INC	ACC		[1]
	STO	COUNT		
	CMP	MAX	// is COUNT = MAX?	[1]
	JPN	LOOP1	// if FALSE, jump to LOOP1	[1]
REVERSE:	DEC	IX	// decrement index register	[1]
	LDM	#0	// set ACC to zero	[1]
	STO	COUNT	// store in COUNT	
LOOP2:	LDX	NAME	// load character from indexed address NAME	[1]
	OUT		// output character to screen	
	DEC	IX	// decrement index register	[1]
	LDD	COUNT	// increment COUNT starts here	
	INC	ACC	//	[1]
	STO	COUNT	//	
	CMP	MAX	// is COUNT = MAX?	[1]
	JPN	LOOP2	// if FALSE, jump to LOOP2	
	END		// end of program	[1]
COUNT:				
MAX:	4			
NAME:	B01000110		// ASCII code in binary for 'F'	
	B01010010		// ASCII code in binary for 'R'	
	B01000101		// ASCII code in binary for 'E'	
	B01000100		// ASCII code in binary for 'D'	

[Max 15]

Answer 26

4

	Acceptance testing	Integration testing	
Who	The end user // user of the software	The programmer / in-house testers	**[1]** + **[1]**
When	When the software is finished/ when it is installed	When the separate modules have been written and tested	**[1]** + **[1]**
Purpose	To ensure the software is what the customer ordered // to check that the software meets the user requirements	To ensure the modules work together as expected	**[1]** + **[1]**

Answer 27

1 (a) (i)	`TYPE LinkedList`	1	3
	`(DECLARE) Surname : STRING` `(DECLARE) Ptr : INTEGER`	1	
	`ENDTYPE`	1	
	Accept: `LinkedList : RECORD`	1	
	`Surname : STRING` `Ptr : INTEGER`	1	
	`ENDRECORD`	1	
	Accept: `TYPE LinkedList = RECORD`	1	
	`Surname : STRING` `Ptr : INTEGER`	1	
	`ENDTYPE / ENDRECORD`	1	
	Accept: `STRUCTURE LinkedList`	1	
	`(DECLARE) Surname : STRING` `(DECLARE) Ptr : INTEGER`	1	
	`ENDSTRUCTURE`	1	
	Accept `AS` / `OF` instead of `:`		
(ii)	`(DECLARE)` **`SurnameList[1:5000]`** `:` **`IinkedList`** Accept `AS` / `OF` instead of `:` Accept `()` instead of `[]` Accept without lower bound Index separator can be `,` `:` `...`		2
(b) (i)	`Wu` Accept with quotes		1
(ii)	`6`		1
(c) (i)	`IsFound` + relevant description `BOOLEAN`	1 1	2

(ii)	Accept () instead of []
	Current ← **StartPtr**
	IF Current = 0
	THEN
	OUTPUT **"Empty List"** (or similar message)
	(accept without quotes) Reject "Error"
	ELSE
	IsFound ← **FALSE**
	INPUT ThisSurname
	REPEAT
	IF **SurnameList[Current].Surname** = ThisSurname
	THEN
	IsFound ← TRUE
	OUTPUT "Surname found at position ", Current
	ELSE
	// move to the next list item
	Current ← SurnameList[Current].Ptr
	ENDIF
	UNTIL IsFound = TRUE OR **Current = 0**
	IF IsFound = FALSE
	THEN
	OUTPUT "Not Found"
	ENDIF
	ENDIF
		6

	Accept = for assignment	

Answer 28

2 (a) (i)	A procedure which is defined in terms of itself // A procedure which makes a call to itself // A procedure that calls itself	1
(ii)	08 // 8	1
(b) (i)		4

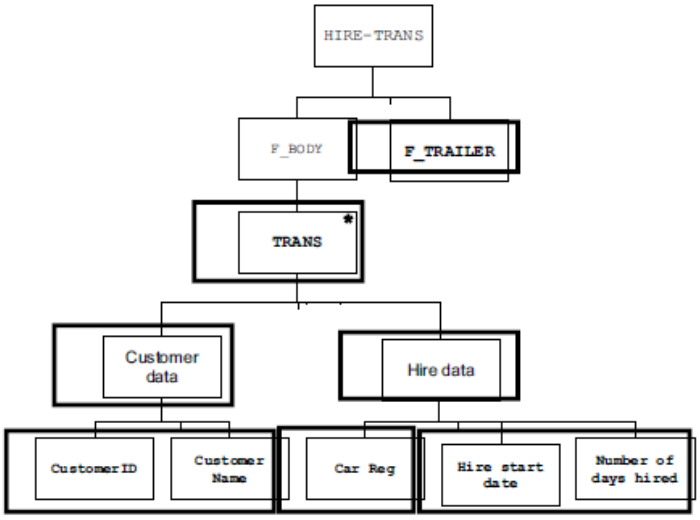
Index	Item
1	9
2	
3	
4	
5	
6	
7	
8	

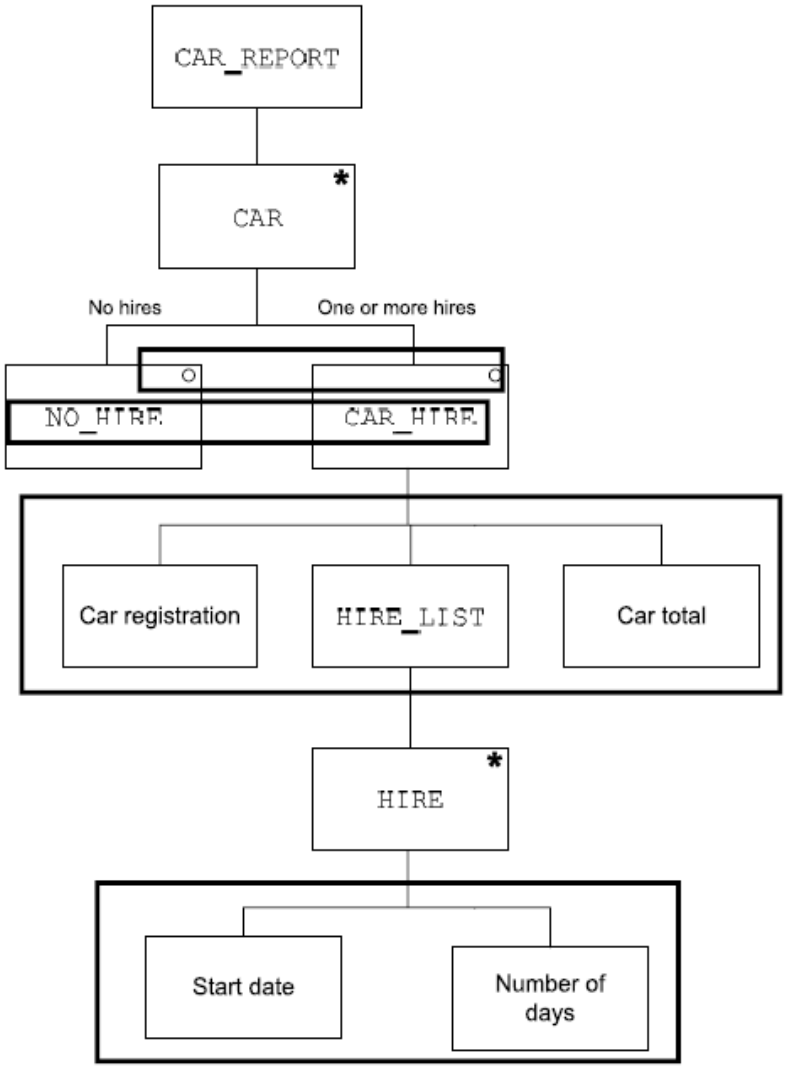
MyList									
1	2	3	4	5	6	7	8	9	10
3	5	8	9	13	16	27	0	0	0
			13						
				16					
					27				
						0			

Note: Final mark only if no additional entries in table
Accept last row to show all final values

(ii)	Any one from: Deletes/removes parameter value/ Item (from the array `MyList`) // Deletes the first entry (in `MyList`) that equals or is bigger than `Item` Overwrites `Item` by moving subsequent items up/down/across/left R right	1

Answer 29

3 (a)		7
	Mark as follows: Label F_TRAILER 1 Label TRANS 1 Customer box (Accept label Customer) 1 Hire box (Accept label Hire) 1 Customer fields : Customer Name, CustomerID/IDnumber 1 Hire fields: Car Reg 1 Hire fields: Hire start date, Number of days hired 1 accept level 5 fields in any order Ignore parent	

(b)	 *Mark as follows:* Selection symbol x 2 (Car-hire / No car-hire) 1 Labelling for `CAR_HIRE` / `NO_HIRE` (accept similar labels*) 1 Labelling for Car registration and Car total / Total hires 1 Iteration symbol for `HIRE` (accept in `HIRE_LIST` as a BOD) 1 Labelling for start date and number of days (as per diagram) 1 * For `CAR_HIRE` label: Accept: Hires / hired / Car data / hire data / hire record / one or more hires	**5**

Answer 30

4 (a) (i)	a03, h07, a23 accept in any order, must be lower case	1
(ii)	The car must pass (both) brake test **and** tyres test	1
(b)	retestAllowed(**ThisCar**) 1 If **(testBrakes(ThisCar, pass) and testTyres(ThisCar, fail))** 1 **or (testBrakes(ThisCar, fail) and testTyres(ThisCar, pass))** 1 (one mark per bold underlined all correct) accept another variable instead of **ThisCar**, but must be same throughout.	3
(c) (i)	a07 [p03] must be [] must be lower case, but don't penalise twice, so follow through from part(b)	2
(ii)	[p05,m04]	1
(iii)	[ ]	1
(d)	[ ]	1

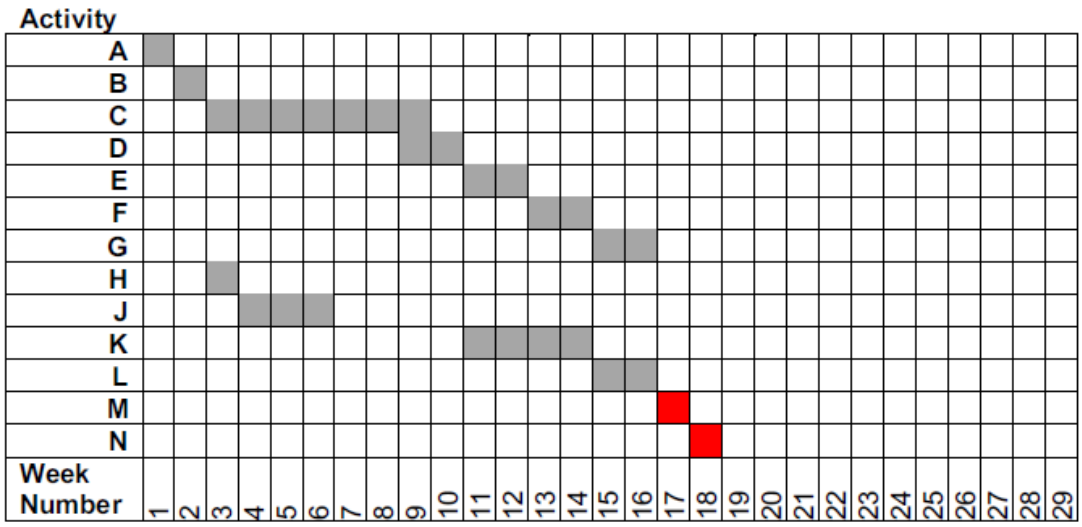
Answer 31

5 (a) (i)	<table><tr><th>Mark</th><th>Description</th><th>Expected result (Grade)</th></tr><tr><td></td><td>Normal</td><td>FAIL/PASS/MERIT/DISTINCTION</td></tr><tr><td></td><td>Abnormal</td><td>Error</td></tr><tr><td></td><td>Extreme/Boundary</td><td>FAIL/PASS/MERIT/DISTINCTION</td></tr></table> 3 × (mark + matching grade) for abnormal data accept negative values, non-integer values, Expected Result: Error 0 and marks above 100 are still acceptable values Do not accept FAIL in expected result column for Abnormal data	3
(ii)	(The programmer is) concerned only with the input (i.e. the mark) to the function and monitoring the expected output (i.e. the grade) // can compare expected result and actual result	1
(b)	Exception: 1. situation causing a crash / run-time error / fatal error 1 Exception handling: 2. code which is called when a run-time error occurs 1 3. … to avoid the program terminating/crashing 1	3

(c)	1 Open a non-existent file 2 Directory path does not exist 3 Attempt to read past the end of the file // attempt to read an empty file 4 Array subscript is out of range 5 Non-integer value / corrupt data read 6 File already open in a different mode // wrong file permissions	Max 3
(d) (i)	09 // 9	1
(ii)	1 Line 11 catches exceptions (only) between lines 05 and 10 — 1 2 Line 11 stops the program from crashing — 1 3 Different exception types recognised — 1 4 Each exception type has an appropriate message output — 1 5 The program language has an (object) type `EXCEPTION` — 1 6 `ThisException` is the instance of `EXCEPTION` which has been raised — 1 7 `EXCEPTION` objects have a 'Message' property // the message property for `ThisException` is "Arithmetic operation resulted in an overflow" — 1	Max 3

Answer 32

1 (a) (i)



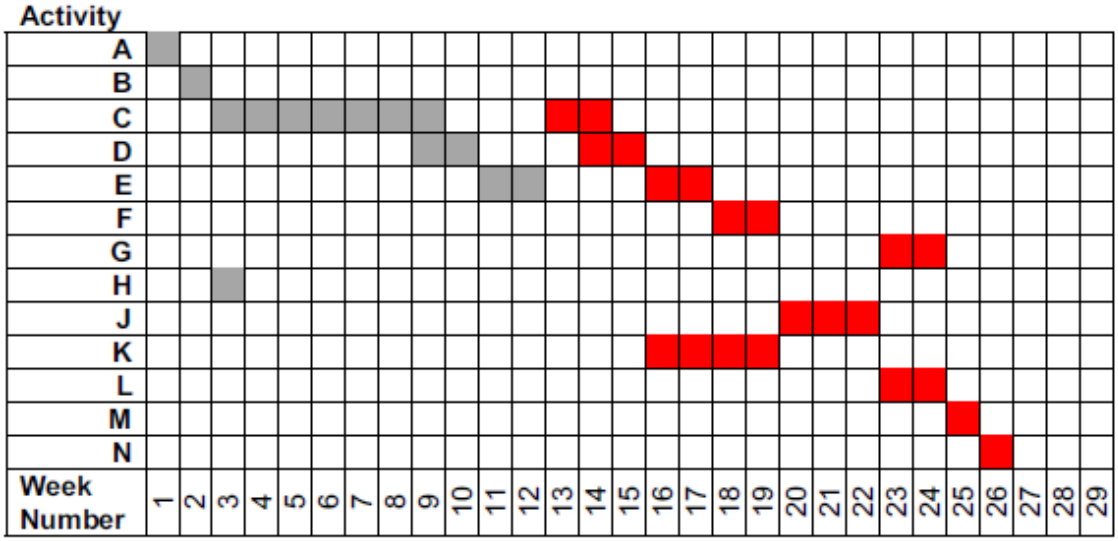
1 mark for each square [2]

(ii) week number 18

Allow follow through [1]

(b) (i)

Activity	Description	Weeks to complete
A	Write requirement specification	1
B	Produce program design	1
C	Write module code	7
D	Module testing	2
E	Integration testing	2
F	Alpha testing	2
G	Install software and carry out acceptance testing	2
H	Research and order hardware	1
J	Install delivered hardware	3
K	Write technical documentation	4
L	Write user training guide	2
M	Train users on installed hardware and software	1
N	Sign off final system	1



1 mark per activity (but 1 mark for activity M and N)
Notes:
C must be after E (1 or 2 later is ok)
D, E, F correct relative to C
J must start in week 20 (allow 21, 22)
G must come after the end of J (f.t.)
K finishes after or at same time as F
L finishes at the same time as G **and after the end of J** (or 1-2 weeks later)
M starts **when everything else has finished**. N after or at same time as M [9]

(ii) week number: 26

Allow f.t. [1]

Answer 33

2 **(a)**
```
parent(ali, ahmed).
parent(meena, ahmed).
```

Accept statements in either order
Wrong capitalisation minus 1 mark [2]

(b)
```
P =
ahmed
aisha
```

Ignore capitalisation
Deduct 1 mark for every extra result [2]

(c)
```
mother(M, gina).
```

Accept `parent(M, gina) AND female(M).` Accept a comma instead of `AND`
Reject `mother(M, gina) IF female(M) AND parent(M, gina).`
Ignore capitalisation [1]

(d)
```
father(F, C)
IF
male(F) AND parent(F, C).
```

(1) (1) [2]

(e)
```
brother(X, Y)
IF
male(X)AND                    [1]
parent(A, X)  AND             [1]
parent(A, Y)                  [1]
AND NOT X=Y.                  [1]
```

Accept any variable for A, but it must be the same in both places
Accept father/mother instead of parent
Ignore capitalisation

Answer 34

4 **(a)**
```
FUNCTION Hash(Key : STRING) RETURNS INTEGER
DECLARE Number : INTEGER
Number ← ASCII(LEFTSTRING(Key,1))
// Number ← ASCII(Key[1])
Number ← Number - 64
RETURN Number
// Result ← Number   // Hash ← Number
ENDFUNCTION
```

Accept `ASC` instead of `ASCII`
Accept `LEFT` instead of `LEFTSTRING`
Key can be a different identifier but must be the same in both places [5]

(b) (i)

Index	Dictionary Key	Value
1		
2		
3	Computer	Rechner
4	Disk	Platte
5	Error	Fehler
6	File	Datei
7		
8		
:	:	
:	:	
1999		
2000		

Ignore spelling mistakes
1 mark for 2 correct pairs entered in correct slots [2]

(ii) Collision / synonym / space already occupied / same index in array
Overwrites previous key-value pair

reject error [Max 2]

(iii) Create an overflow area
The 'home' record has a pointer to others with the same key // linked list
OR
Store the overflow record at the next available address …
in sequence (= next available)
OR
Re-design the hash function …. // write a different/another algorithm
to generate a wider range of indexes // enlarging storage space // to create fewer collisions [2]

(iv) Mark as follows:

Check whether slot is empty:

```
IF Dictionary[Index,1] <>"" //  != ''    //  > NULL   // >
NONE
```

If not: update index: `THEN Index ← <some value>`
...to find an empty slot (loop / follow pointer / go to overflow area) reject FOR loop
Insert code between lines 20 and 30

```
21 WHILE Dictionary[Index,1] > ""
22   Index ← Index + 1
23   IF Index > 2000
24     THEN
25       Index ← 1
26   ENDIF
27 ENDWHILE
```

[4]

Answer 35

5 (a) (i)

Accumulator	Memory Address 509	510	511	512
0	7	3	0	0
7				7
0				
1			1	
7				
14				14
1				
2			2	
14				
21				21
2				
3			3	
3 marks			*1 mark*	*1 mark*

If values changed in column 509 or 510 don't give marks for 511/512 [5]

(ii) <u>stores</u> the counter value for …..// acts as a control variable/counter
How many times the loop has been performed // control the loop

Ignore re-stating the steps [2]

(b) LDM #12 (must be instruction before storage)
STO 509 (must be final instruction)

1 mark for each instruction [2]

Answer 36

2 **(a)**
```
parent(philippe, meena).
parent(gina, meena).
```
[2]

(b) **ahmed, aisha, raul** [2]

(c)
```
father(F, ahmed).
```
[1]

(d)
```
mother(X, Y)
IF
female(X) AND parent(X, Y).
```
[2]

(e)
```
grandparent(W, Z)
IF
parent(W,X)
AND parent(X,Z).
```
[2]

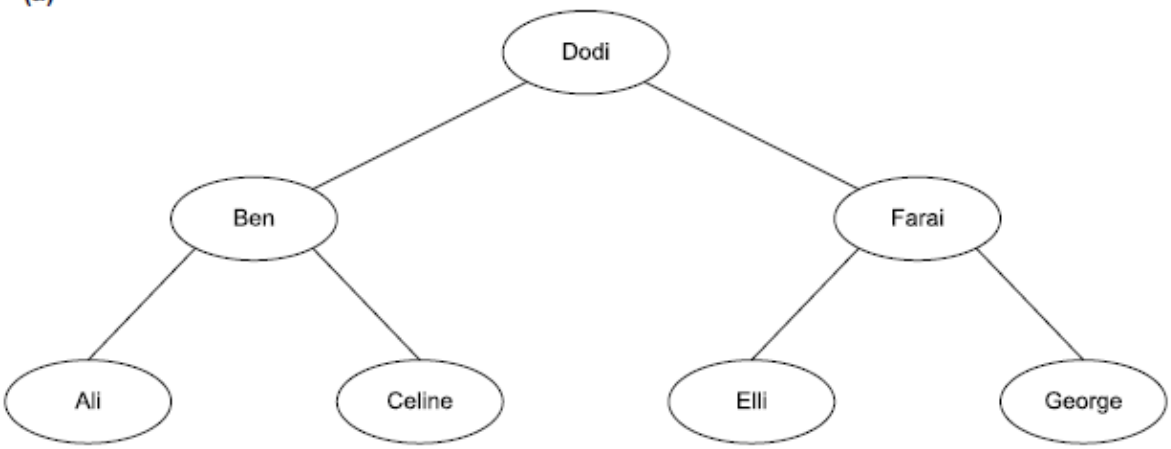
(f)
```
grandfather(G, K)
IF
male(G) AND
grandparent(G, K).

alternative:

father(G, X) AND
parent(X, K).
```
[2]

Answer 37

4 **(a)**



[4]

(b)

RootPointer	FreePointer
1	8

Tree

	Name	LeftPointer	RightPointer
[1]	Dodi	5	2
[2]	Farai	3	4
[3]	Elli	0	0
[4]	George	0	0
[5]	Ben	7	6
[6]	Celine	0	0
[7]	Ali	0	0
[8]		9	0
[9]		10	0
[10]		0	0

[7]

(c) **(i)**

```
01 PROCEDURE TraverseTree(BYVALUE Root : INTEGER)
02    IF Tree[Root].LeftPointer < > 0
03       THEN
04          TraverseTree(Tree[Root].LeftPointer)
05    ENDIF
06    OUTPUT Tree[Root].Name
07    IF Tree[Root].RightPointer < > 0
08       THEN
09          TraverseTree(Tree[Root].RightPointer)
10    ENDIF
11 ENDPROCEDURE
```

[5]

(ii) A procedure that calls itself // is defined in terms of itself
Line number: 04/09 [2]

(iii) `TraverseTree(RootPointer)` [1]

Answer 38

5 (a)

MembershipFile

Address	MemberID	other member data
0	0	
1	1001	
2	7002	
3	0	
4	0	
5	3005	
6	0	
7	0	
8	0	
:	:	
:	:	
96	4096	
97	0	
98	2098	
99	0	

1001 and 7002 and 3005 1
4096 and 2098 1 [2]

(b) (i)

```
10 // generate record address
20 NewAddress ← Hash(NewMember.MemberID)
30 // move pointer to the disk address for the record
40 SEEK NewAddress
50 PUTRECORD "MembershipFile", NewMember
```

[4]

(ii)

```
01 TRY
02    OPENFILE "MembershipFile" FOR RANDOM
03 EXCEPT
04    OUTPUT "File does not exist"
05 ENDTRY
```

[2]

(iii) collisions/synonyms
The previous record will be overwritten [2]

(iv) Create an overflow area
The 'home' record has a pointer to others with the same key
OR
Store the overflow record at the next available address
in sequence
OR
Re-design the hash function ….
to generate a wider range of indexes // to create fewer collisions [2]

(v)

```
41 GETRECORD "MembershipFile", CurrentRecord
42 WHILE CurrentRecord.MemberID <> 0
43    NewAddress ← NewAdress + 1
44    IF NewAddress > 99 THEN NewAddress ← 0
45    SEEK NewAddress
46    GETRECORD "MembershipFile", CurrentRecord
47 ENDWHILE
```

[max. 4]