Answer

Answer 1

1(a)	It is an unplanned event // an event not wanted
1(b)	**1 mark** per example to **max 3** e.g. • Division by zero • Invalid array index • File does not exist • Run-time error • Invalid input • Invalid argument/value • Stack overflow • Memory leakage • Hardware failure/error
1(c)	**1 mark** per bullet point to **max 2** • The program will not crash // more robust // program will continue • Result does not cause further errors/problems later • Appropriate error messages/result • Exceptional conditions are identified • Improve readability

Answer 2

2(a)

1 mark for the first 3 rows
1 mark for the last 2 rows

Feature	Must be included
Incrementation	
General case	✓
Base case	✓
Selection case	
It calls itself	✓

2(b)

1 mark for each of the spaces filled in

```
PROCEDURE Count(BYVALUE Number : INTEGER)
    IF MOD(Number, 2) <> 0
        THEN
            Number ← Number - 1
    ENDIF
    OUTPUT Number
    IF Number > 0
        THEN
            CALL Count(Number - 1)
    ENDIF
ENDPROCEDURE
```

2(c)	**1 mark** per bullet point
	• Recursive call
	• ..with correct parameters
	• …in correct working place(s)
	• Removal of loop
	• Remainder of program in correct place
	
	`PROCEDURE MealsCount(BYREF MealOption1 : INTEGER, MealOption2 : INTEGER)`
	`  DECLARE MealOption : INTEGER`
	`  INPUT MealOption`
	`  IF MealOption = 1`
	`  THEN`
	`  MealOption1 ← MealOption1 + 1`
	`  CALL MealsCount(MealOption1, MealOption2)`
	`  ELSE`
	`  IF MealOption = 2`
	`  THEN`
	`  MealOption2 ← MealOption2 + 1`
	`  CALL MealsCount(MealOption1, MealOption2)`
	`  ELSE`
	`  OUTPUT MealOption1, " ", MealOption2`
	`  ENDIF`
	`  ENDIF`
	`ENDPROCEDURE`

Answer 3

3(a)	**1 mark** for each statement • `person(elle).` • `sport(rugby).` • `plays(elle, rugby).` • `will_not_play(elle, hockey).`

3(b)	`johann, jessica`

3(c)	**1 mark** per bullet point
	
	• `person(Y)`
	• `AND // ,`
	• `sport(X)`
	• `AND NOT // , NOT`
	• `will_not_play(Y, X)`
	
	`mightplay(Y, X)`
	`IF person (Y) AND sport (X) AND NOT(will_not_play(Y, X))`

Answer 4

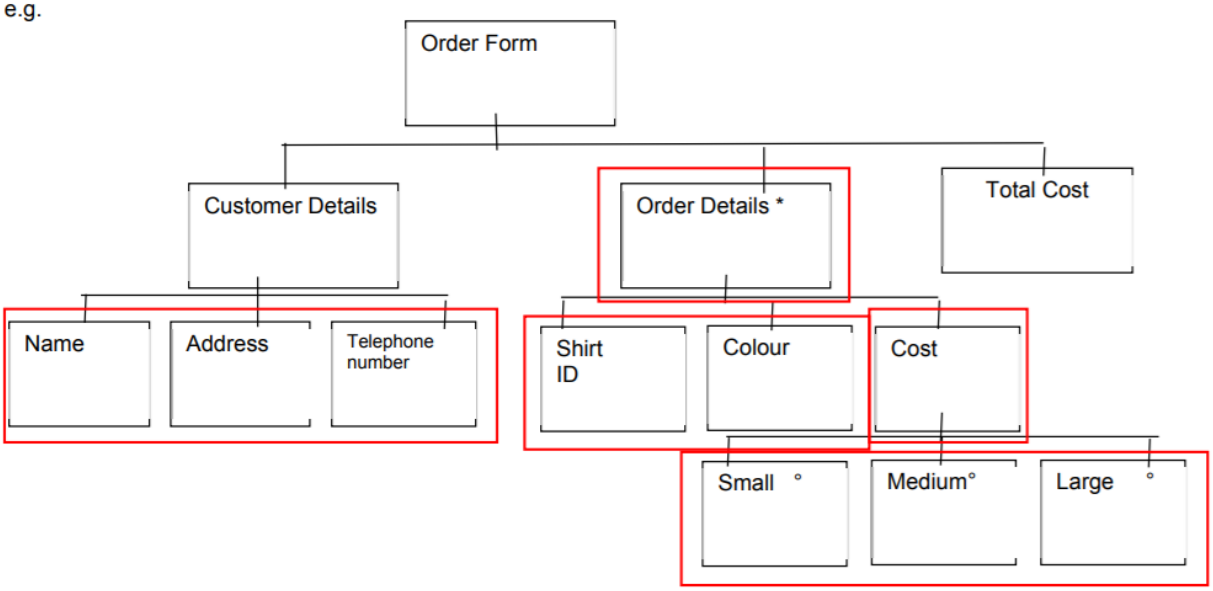
4(a)	**1 mark** per bullet point to **max 2** • Can use the properties from the parent/super class (without redeclaring them) • Can use the methods from the parent/super class (without redeclaring them) • Can extend the properties from the parent/super class • Can extend the methods from the parent/super class
4(b)	**1 mark** per feature to **max 2** • Polymorphism • Encapsulation • Containment • Aggregation • Composition

Answer 5

6(a)	**1 mark** per bullet point to **max 4**
	
	• Procedure header and end
	• Loop 6000 times
	• Access the `UserID` and `PINNumber` for each element in `CustomerDetails`
	• Correct initialisation of `UserID` to `""` and `PINNumber` to `0`
	
	`PROCEDURE InitialiseHashTable()`
	`  FOR x ← 0 TO 5999`
	`  CustomerDetails[x].UserID ← ""`
	`  CustomerDetails[x].PINNumber ← 0`
	`  ENDFOR`
	`ENDPROCEDURE`

6(b)	**1 mark** for each completed missing statement
	
	`FUNCTION InsertRecord(NewRecord) RETURNS INTEGER`
	`  DECLARE Count : INTEGER`
	`  DECLARE Index : INTEGER`
	`  Count ← 0`
	`  Index ← Hash(NewRecord.UserID)`
	`  WHILE (CustomerDetails[Index].UserID <> "") AND (Count <= 5999)`
	`  Index ← Index + 1`
	`  Count ← Count + 1`
	`  IF Index > 5999`
	`  THEN`
	`  Index ← 0`
	`  ENDIF`
	`  ENDWHILE`
	
	`  IF Count > 5999`
	`  THEN`
	`  RETURN -1`
	`  ELSE`
	`  CustomerDetails[Index] ← NewRecord`
	`  RETURN Index`
	`  ENDIF`
	`ENDFUNCTION`

Answer 6

7	**1 mark** per bullet point • Name, Address and Telephone Number beneath Customer details • Order Details below booking form • …with appropriate iteration of entering Order Details • Shirt ID and Colour beneath Order Details • Cost beneath Order Details • …with Small, Medium and Large below it • …with selection e.g. 
7(b)(i)	Serial

7(b)(ii)	**1 mark** per bullet point to **max 2** • Sequential • Random

7(c)	**1 mark per** completed statement
	
	`PROCEDURE UpdateTelephone(BYREF ThisCustomer : Customer,`
	`BYVALUE NewTelephoneNumber : STRING)`
	`ThisCustomer.TelephoneNumber ← NewTelephoneNumber`
	`ENDPROCEDURE`

7(d)	**1 mark** per column pair

Conditions	**Order over $50**	Y	Y	Y	Y	N	N	N	N
	Monday	Y	Y	N	N	Y	Y	N	N
	Loyalty card	Y	N	Y	N	Y	N	Y	N
Actions	**Additional 5% discount**	**Y**	**N**	**Y**	**N**	**Y**	**N**	**Y**	**N**
	10% discount	**Y**	**Y**	**Y**	**Y**	**N**	**N**	**N**	**N**
	Free gift	**Y**	**Y**	**N**	**N**	**N**	**N**	**N**	**N**
	Free delivery	**Y**	**N**	**Y**	**N**	**N**	**N**	**N**	**N**

Answer 7

1(a)	**1 mark** per reason to **max 3** e.g. • Division by zero • Array out of bounds • File does not exist • Stack overflow • Memory leakage • Hardware fault/failure

Answer 8

2(a)	**1 mark** per bullet point to **max 4** • Declaration of array with 3000 elements • … of type `CustomerRecord` • Looping 3000 times • Accessing the username and password for each field • Correct initialisation of values to “”

```
DECLARE CustomerLogIn : ARRAY[0:2999] OF CustomerRecord
FOR x ← 0 TO 2999
  CustomerLogIn[x].Username ← ""
  CustomerLogIn[x].Password ← ""
ENDFOR
```

2(b)(i)	**1 mark** for each correctly filled in space
	
	`FUNCTION SearchHashTable(BYVALUE SearchUser : STRING) RETURNS INTEGER`
	`  DECLARE Index : INTEGER`
	`  DECLARE Count : INTEGER`
	`  Index ← Hash(SearchUser)`
	`  Count ← 0`
	`  WHILE (CustomerLogIn[Index].Username <> SearchUser) AND`
	`  (CustomerLogIn[Index].Username <> "") AND (Count < 2999)`
	`  Index ← Index + 1`
	`  Count ← Count + 1`
	`  IF Index > 2999`
	`  THEN`
	`  Index ← 0`
	`  ENDIF`
	`  ENDWHILE`
	`  IF CustomerLogIn[Index].Username = SearchUser`
	`  THEN`
	`  RETURN Index`
	`  ELSE`
	`  RETURN -1`
	`  ENDIF`
	`ENDFUNCTION`

2(b)(ii)	**1 mark** per bullet point to **max 2** • Records if you have checked every record // the number of records checked in the array • … without finding the search value • … and stops infinite loop // stopping condition

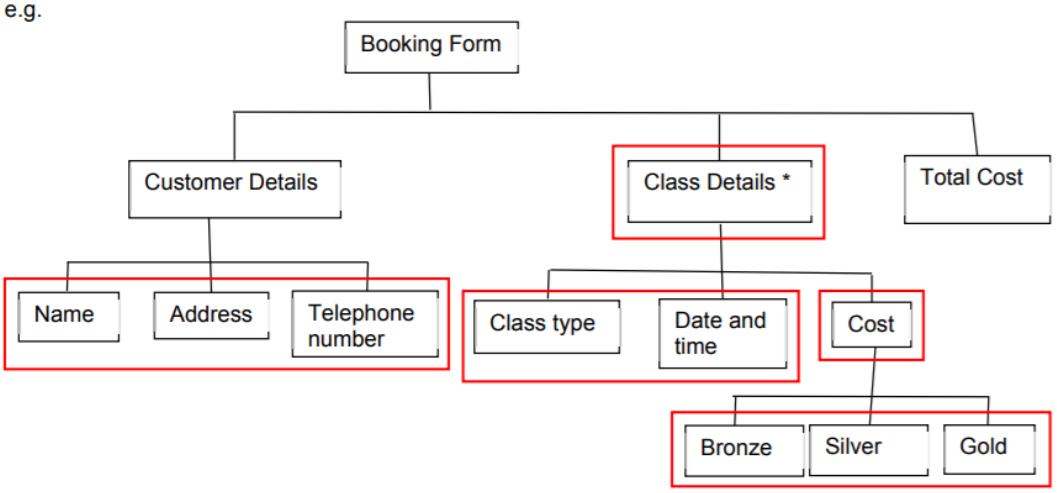
Answer 9

3(a)	1 mark per bullet point • Have a base case // stopping condition • Work toward the base case // general case // changes state • Call itself // defined in terms of itself

3(b)	1 mark for each (shown in **bold**)
	
	`FUNCTION IsPalindrome(CheckWord : STRING) RETURNS BOOLEAN`
	`  IF Length(CheckWord) <= 1`
	`  THEN`
	`  RETURN True`
	`  ENDIF`
	`  IF Substring(CheckWord, 0, 1) <>`
	`  Substring(CheckWord,Length(CheckWord)-1, 1)`
	`  THEN`
	`  RETURN False`
	`  ELSE`
	`  RETURN IsPalindrome(Substring(CheckWord,1,`
	`  Length(CheckWord)-2))`
	`  ENDIF`
	`ENDFUNCTION`

3(c)	**1 mark** per bullet point
	
	• Function is defined and returns integer with correct parameters
	• Returns 1 if exponent is 0 // returns base if exponent is 1
	• Calculates base to power with recursive call …
	• … with correct parameters
	• Returns result of calculation
	e.g.
	`FUNCTION FindPower(Base:INTEGER, Exp: INTEGER) RETURNS INTEGER`
	`  DECLARE Value : INTEGER`
	`  IF Exp = 0`
	`  THEN`
	`  RETURN 1`
	`  ELSE`
	`  Value ← Base * FindPower(Base, Exp - 1)`
	`  RETURN Value`
	`  ENDIF`
	`ENDFUNCTION`

Answer 10

4(a)	**1 mark** per bullet point: • Name, Address and Telephone number beneath Customer details • Class Details (or equivalent) below booking form • … with appropriate iteration of entering Class Details • Class type, and Date and time beneath Class Details (or equivalent) • Cost beneath Class Details (or equivalent) … • … with Bronze, Silver and Gold below it (FT for name and position of Cost) … • … with selection (FT for name and position of B, S, G) e.g. 
4(b)	**1 mark** for each feature to **max 2** • Sequence • Selection • Iteration

Answer 11

5(a)	• person(gina) • country(cyprus) • visited(gina, cyprus)

5(b)	**1 mark** for 2 correct, **2 marks** for 3 correct
	
	`william`
	`deeraj`
	`meghan`

5(c)	**1 mark** per bullet point
	• `person(P) // country(C)`
	• `AND country(C) // AND person(P)`
	• `AND NOT // , NOT`
	• `visited(P, C)`
	
	`mightvisit(P, C)`
	`IF person (P) AND country (C) AND NOT visited(P, C)`

Answer 12

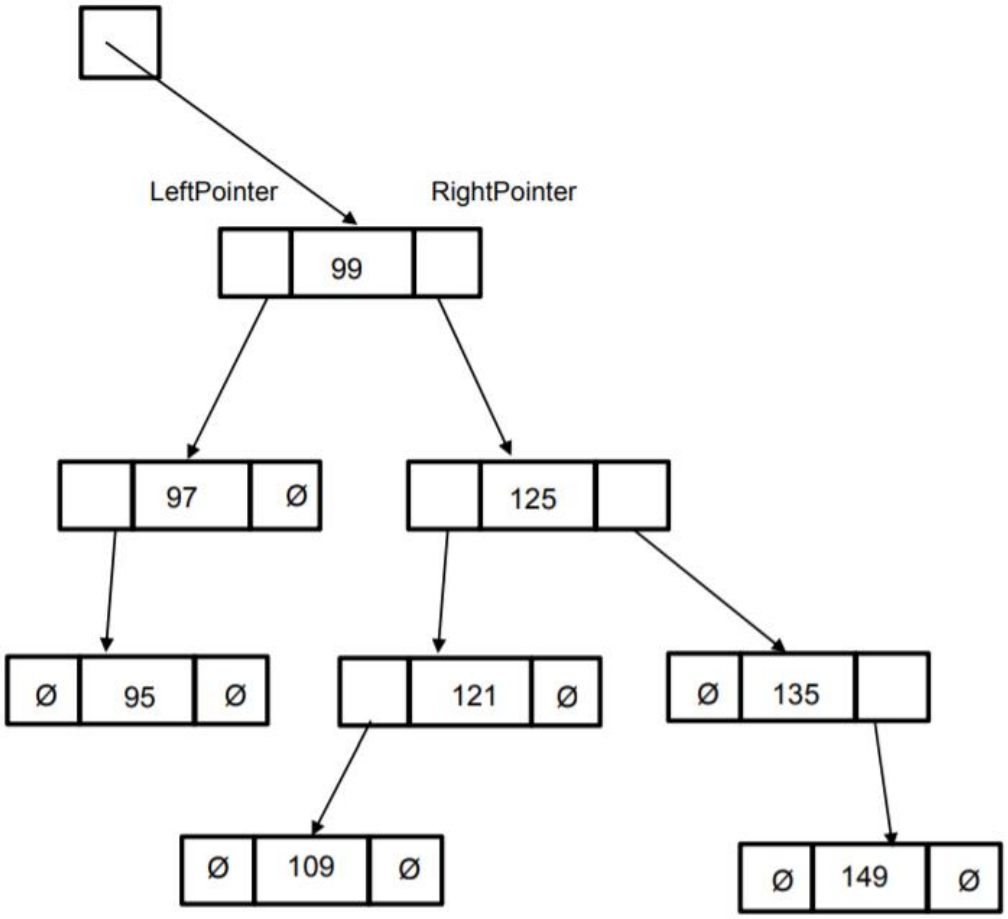
6(a)	**1 mark** per bullet point to **max 3** • A class includes an instance of another class • Aggregation • … the contained object can exist outside of its super class • Composition • … object is declared and exists within its super class // object is destroyed when super class is destroyed
6(b)	**1 mark** per feature to **max 2** e.g. • Inheritance • Polymorphism • Encapsulation • Private/public

Answer 13

8

Statement	Serial	Sequential	Random
Uses a hashing algorithm			✓
No key field is used when storing data e.g. it is in chronological order	✓		
Collisions can occur			✓
Least efficient for a very large number of records	✓		
Most efficient for a very large number of records			✓

Answer 14

2(a)(i)	1 mark per bullet point • 95 to left of 97 • 109 to left of 121 • 135 to right of 125 • 149 to right of 135 • Null points in all places and no inappropriate pointers RootPointer  	**5**

2(a)(ii)	1 mark per bullet point • FreePointer as 8 • 99 • 125 • 121 and 97 • 109 and 95 • 135 and 149	**6**

RootPointer
0

FreePointer
8

Index	LeftPointer	Data	RightPointer
[0]	3	99	1
[1]	2	125	6
[2]	4	121	null
[3]	5	97	null
[4]	null	109	null
[5]	null	95	null
[6]	null	135	7
[7]	null	149	null
[8]			

2(b)	1 mark for each completed section	**6**

```
FUNCTION FindElement(Item : INTEGER) RETURNS INTEGER
   CurrentPointer ← RootPointer
   WHILE CurrentPointer <> NullPointer
      IF List[CurrentPointer].Data <> Item
         THEN
            CurrentPointer ← List[CurrentPointer].Pointer
         ELSE
            RETURN CurrentPointer
      ENDIF
   ENDWHILE
   CurrentPointer ← NullPointer
   RETURN CurrentPointer
ENDFUNCTION
```

Answer 15

2(c)(i)	1 mark per bullet point to max 3 e.g. • A sequence of steps that change the state of the program • The steps are in the order they should be carried out • e.g. procedural programming/language • Groups code into self-contained blocks // split the program into modules • … which are subroutines // by example	**3**
2(c)(ii)	1 mark per bullet point to max 3 e.g. • Creates classes • …as a blueprint for an object // objects are instances of classes • …that have properties/attributes **and** methods • … that can be private to the class // properties can **only** be accessed by the class's methods // encapsulation • **Subclasses** can inherit from **superclasses** (child and parent) • A **subclass** can inherit the methods and properties from the **superclass** • A **subclass** can change the methods from the **superclass** // **subclass** can use polymorphism • Objects can interact with each other	**3**

Answer 16

2(f)(i)	Insertion sort	**1**
2(f)(ii)	**One** from: • Bubble sort • Merge sort	**1**

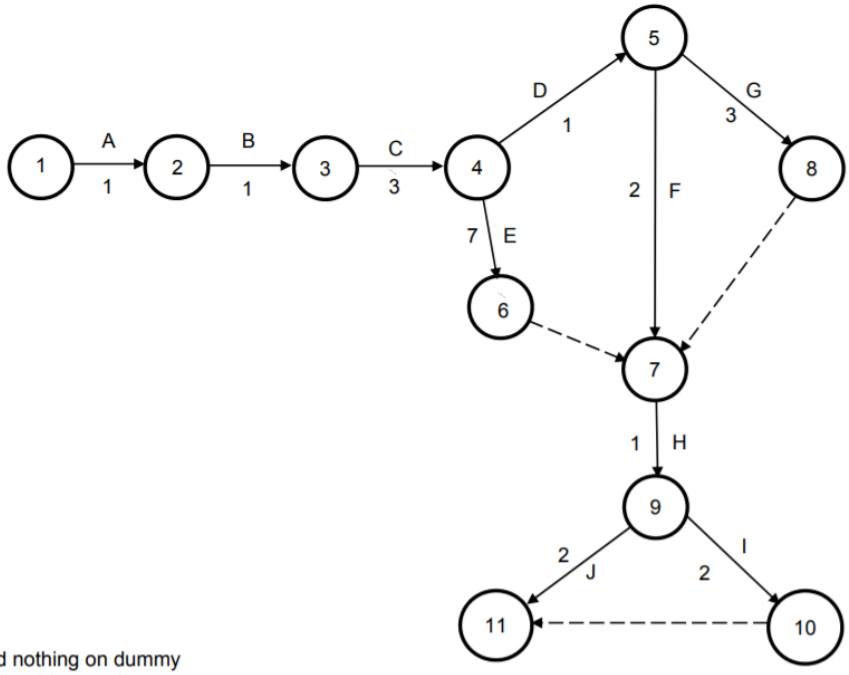
Question	Answer	Marks
2(f)(iii)	1 mark per shaded section	**7**

Item	NumberOfScores	InsertScore	Index	ArrayData				
				0	1	2	3	4
				99	125	121	109	115
1	5	125	0		(125)			
2		121	1			125		
			0		121			
3		109	2				125	
			1			121		
			0		109			
4		115	3					125
			2				121	
			1			115		

Answer 17

3(a)	1 mark per bullet point to max 2 • It is defined in terms of itself // it calls itself • It has a stopping condition // base case • It is a self-contained subroutine • It can return data to its previous call	**2**
3(b)	1 mark per bullet point to max 3 • (When the recursive call is made) all values/data are put on … • … the stack • When the stopping condition/base case is met • …the algorithm unwinds • …the last set of values are taken off the stack (in reverse order)	**3**

Answer 18

1(a)(i)	 1 mark each bullet point: • B 1 • C 3 (following B) • D 10 (following C) • G 3 (following D) and nothing on dummy • E 7 (following C) and nothing on dummy • H1 in position • J 2 (following H) and nothing on dummy	**7**
1(a)(ii)	1 mark: • The next activity is dependent on the previous but there is no activity	**1**

1(b)	1 mark per row									**3**
			Rules							
Conditions	**Public Holiday**	Y	Y	Y	Y	N	N	N	N	
	Hours >= 160	Y	Y	N	N	Y	Y	N	N	
	Pension	Y	N	Y	N	Y	N	Y	N	
Actions	**3% bonus payment**	**X**	**X**	**X**	**X**					
	5% bonus payment	**X**	**X**			**X**	**X**			
	4% Pension payment	**X**		**X**		**X**		**X**		

Answer 19

1(d)	Polymorphism	**1**

Answer 20

2(a)	1 mark per completed statement
	
	`FUNCTION AddToQueue(Number : INTEGER) RETURNS BOOLEAN`
	`  CONSTANT FirstIndex = 0`
	`  CONSTANT LastIndex = 7`
	`  TempPointer ← EndPointer + 1`
	`  IF TempPointer > LastIndex`
	`   THEN`
	`    TempPointer ← FirstIndex`
	`  ENDIF`
	`  IF TempPointer = StartPointer`
	`   THEN`
	`     RETURN FALSE`
	`   ELSE`
	`    EndPointer ← TempPointer`
	`    NumberQueue[EndPointer] ← Number`
	`    RETURN TRUE`
	`  ENDIF`
	`ENDFUNCTION`

2(b)	1 mark per bullet point 1 mark for: • … if the start pointer reaches the end of the queue, it becomes the index of the first element in the queue **Max 3** from: • Checks if the circular queue is empty // Checks if the queue has any data in it • … if it is empty it **reports** that it is empty • If not empty, return the value at the position of the **start pointer** … • … then increments the start pointer

2(c)	1 mark per bullet point to **max 3** e.g. • Stack • Linked list • Dictionary • (Binary) tree

Answer 21

3(a)

1 mark per test data type

- Normal / valid
- Abnormal / erroneous / invalid
- Boundary / extreme

3(b)(i)

1 mark for each name

Description	Name of debugging feature
A point where the program can be halted to see if the program works to this point	**Breakpoint**
One statement is executed and then the program waits for input from the programmer to move to the next statement.	**Stepping // step through/over/into**

3(b)(ii)

1 mark for name, 1 for description
e.g.

- variable watch window
- observe how variables change during execution // view current status of variables

- Error list
- describes the error // gives line number of error

Answer 22

5(a)

1 mark for each shaded section

- `LDD NUMBER`
- `LSL`
- `#2`
- `STO NUMBER`
- `END`

Label	Op code	Operand	Comment
	LDD	**NUMBER**	`// load contents of NUMBER`
	LSL	**#2**	`// perform shift to multiply by 4`
	STO	**NUMBER**	`// store contents of ACC in NUMBER`
	END		`// end program`
`NUMBER:`	`B00110110`		

5(b)

Label	Op code	Operand	Comment	
	LDR	#0	// initialise index register to 0	
START:	**LDX**	**STRING**	// load the next value from STRING	**1**
	AND	**MASK**	// bitwise AND operation with MASK	**1**
	CMP	**MASK**	// check if result equals MASK	**1**
	JPN	**UPPER**	// if FALSE jump to UPPER	**1**
	LDD	**COUNT**	// increment COUNT	**1**
	INC	**ACC**		
	STO	**COUNT**		
UPPER:	INC	IX	// increment Index Register	**1**
	LDD	**LENGTH**	// decrement LENGTH	**1**
	DEC	**ACC**		
	STO	**LENGTH**		
	CMP	**#0**	// is LENGTH = 0 ?	**1**
	JPN	**START**	// if FALSE, jump to START	**1**
	END		// end program	

5(b)

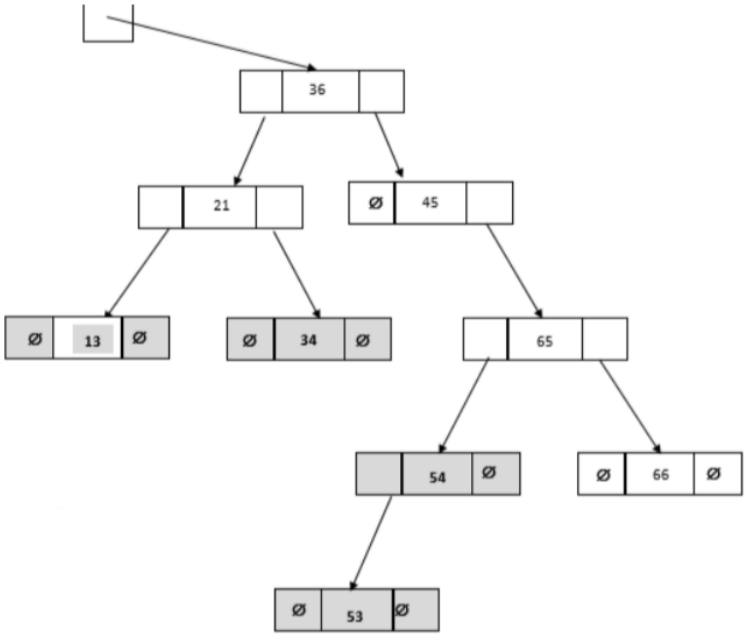
MASK:	B00100000	// if bit 5 is 1, letter is lower case	
COUNT:	0		
LENGTH:	5		
STRING:	B01001000	// The ASCII code for 'H'	
	B01100001	// The ASCII code for 'a'	
	B01110000	// The ASCII code for 'p'	
	B01110000	// The ASCII code for 'p'	
	B01011001	// The ASCII code for 'Y'	

Answer 23

1(a)(i)

1 mark per bullet point

- 13 to left of 21
- 34 to right of 21
- 54 to left of 65
- 53 to left of 54
- Null and other pointers on all boxes written **and** no other pointers filled in



1(a)(ii)

1 mark per bullet point

- FreePointer
- 36
- 45
- 21 and 65
- 66 and 13
- 54, 53 and 34

RootPointer
0

FreePointer
9

Index	LeftPointer	Data	RightPointer
[0]	2	36	1
[1]	null	45	3
[2]	5	21	8
[3]	6	65	4
[4]	null	66	null
[5]	null	13	null
[6]	7	54	null
[7]	null	53	null
[8]	null	34	null
[9]			

Answer 24

1(b)(ii)	1 mark per bullet point to max 3 • Used to access/change the properties/attributes • …only using the get/set methods • …that are set to private • Provide encapsulation • Prevents accidental change • To make sure data is valid // act as validation • Hides data • The get methods allow the data to be accessed/returned • The set methods allow the data to be changed/written to

Answer 25

1(c)	1 mark per bullet point to max 4 • There are a large number of objects/records to store // hash is better for a large number of objects/records • A hashing **algorithm/hash** performed on **key field/record** (to form the address) • …to allow direct access to the object • …so it is likely to be faster in finding the object // linked list is slower in finding the object • In a linked list, each object needs to be checked until found // sequentially/linear accessed • …the left/right/next pointer is followed // have to trace pointers

Answer 26

2(a)	1 mark for correct tick **Statement** / **Tick (✓)** Last in first out / ✓ First in first out / Last in last out /	**1**
2(b)(i)	1 mark for correct stack 10 35 20	**1**
2(b)(ii)	1 mark for correct stack 65 50 35 20	**1**

2(b)(iii)	1 mark for each bullet
	
	• Function `Push` ...
	• ...taking parameter (returning Boolean)
	• Checking if `Top` = 7 ...
	• ...returning `FALSE` if full
	• ...returning `TRUE` otherwise
	• if not full, increment `Top`
	• ... add parameter to `Top` of `ArrayStack`
	
	`FUNCTION Push (BYVALUE DataItem : Integer) (RETURNS Boolean)`
	`  IF Top = 7`
	`  THEN`
	`  RETURN FALSE`
	`  ELSE`
	`  Top ← Top + 1`
	`  ArrayStack[Top] ← DataItem`
	`  RETURN TRUE`
	`  ENDIF`
	`ENDFUNCTION`

Answer 27

3(a)	1 mark for name of feature; 1 mark for description e.g. • Colouring code//Pretty printing • This is how the code is presented in the IDE e.g. colour coding and indentation • Context-sensitive prompts • Displays keywords or hints at the point of insertion e.g. drop-down list of commands • Auto-indent • Automatically indent your code for selection/iteration/procedures/methods • Auto-complete • Avoid typing errors // speeds up process of typing • Expand/collapse subroutines/code • To make it easier to view code currently working on
3(b)	1 mark per each correct bullet point • Normal • Abnormal / erroneous / invalid • Boundary / extreme

Answer 28

4(a)(i)

1 mark for error and correction

Error 1 – `IF List[LowerBound] = SearchValue`
Correction – `IF List[MidPoint] = SearchValue`

Error 2 – `UpperBound ← MidPoint + 1`
Correction – `LowerBound ← MidPoint + 1`

Error 3 – `IF LowerBound > MidPoint`
Correction - `IF LowerBound > UpperBound`

Error 4 – `IF ValueFound = FALSE`
Correction – `IF ValueFound = TRUE`

4(a)(ii)

Linear search

4(b)(i)

1 mark per shaded section

Count	TempValue	Sorted	ArrayData					
			0	1	2	3	4	5
0	""	TRUE	5	20	12	25	32	29
1	12	FALSE		12	20			
2								
3								
4	29	(FALSE)					29	32
0		TRUE						
1								
2								
3								
4								

4(b)(ii)

1 mark per bullet point

- Initialising a counter variable to 0 **and** must be the same variable used to access array elements
- While loop checking counter is < 5 or <= 4
- Incrementing counter inside the loop and outside IF, **and** remainder of algorithm completed (The IF to ENDIF)

e.g.

```
Count ← 0
WHILE Count < 5
  IF ArrayData[Count] > ArrayData[Count + 1]
    THEN
       TempValue ← ArrayData[Count + 1]
       ArrayData[Count + 1] ← ArrayData[Count]
       ArrayData[Count] ← TempValue
       Sorted ← False
  ENDIF
  Count ← Count + 1
ENDWHILE
```

4(b)(iii)

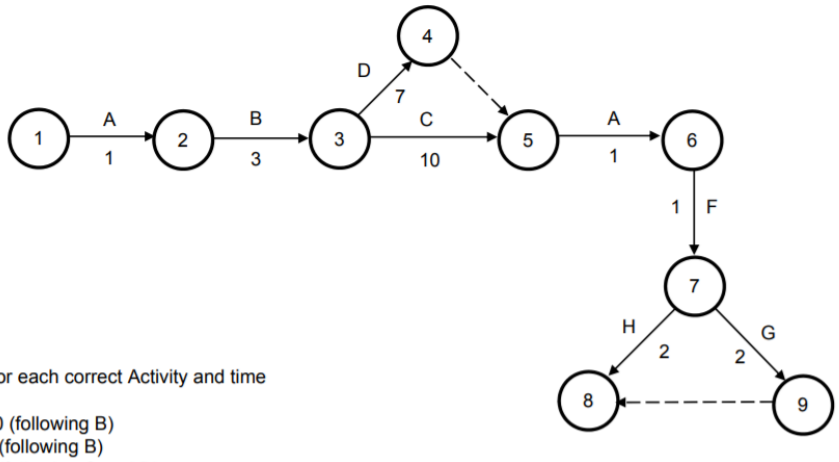
Bubble sort

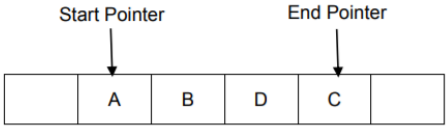
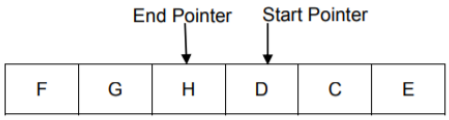
4(b)(iv)

One from:

- Insertion sort
- Merge sort
- Quick sort

Answer 29

1(a)(i)	 1 mark for each correct Activity and time • B 3 • C 10 (following B) • D 7 (following B) • E 3 (following C and D) • G 2 (following F) • H 2 (following F)	**6**
1(a)(ii)	The shortest time to complete the project // the sequence of activities that must be completed to avoid delaying the project	**1**
1(a)(iii)	1 mark for identify, max 1 for description • GANTT • A table that has time across the top and activities on the left, boxes are coloured to show dependencies and find critical path // colour in the boxes to show the length of time for each task	**2**

1(b)(i)	A, B, C and D in correct places with no alteration to start and end pointer 	**1**
1(b)(ii)	1 mark per bullet point • correct jobs in correct order… • … correct location of start pointer • … correction location of new end pointer 	**3**
1(b)(iii)	1 mark from: • An error **message** would be generated	**1**

1(b)(iv)	1 mark for each correct line
	
	`FUNCTION Remove RETURNS STRING`
	`  DECLARE PrintJob : STRING`
	`  IF StartPointer = EndPointer`
	`   THEN`
	`     RETURN "Empty"`
	`   ELSE`
	`     PrintJob ← Queue[StartPointer]`
	`     IF StartPointer = 5`
	`       THEN`
	`        StartPointer ← 0`
	`       ELSE`
	`        StartPointer ← StartPointer + 1`
	`      ENDIF`
	`     RETURN PrintJob`
	`   ENDIF`
	`ENDFUNCTION`
		4

1(b)(v)	1 mark per bullet point • A stack is Last In First Out (LIFO) while a queue is First In First Out (FIFO) • The queue removes and returns the element at **start pointer** // item is removed from the **start/head** // • A stack would remove and return the element at **end pointer** // item is removed from the **end**	2

Answer 30

1(d)(i)	1 mark per bullet • Way of modelling logic • Show **all** possible outputs // shows **every** possible outcome // **all** outcomes … • … based on the inputs • Determine which action to take in specific conditions // how different conditions affect the actions/outcomes	2
1(d)(ii)	1 mark for each row Accept Y/X/ticks as long as clear which are Y. Accept N/X/– for empty spaces	4

		Rules							
Conditions	Document printed, but quality is poor	Y	Y	Y	Y	N	N	N	N
	Error light is flashing on printer	Y	Y	N	N	Y	Y	N	N
	Document printed, but paper size is incorrect	Y	N	Y	N	Y	N	Y	N
Actions	Check connection from computer to printer						**X**		
	Check ink status	**X**	**X**	**X**	**X**				
	Check if there is a paper jam						**X**		
	Check paper size selected	**X**		**X**		**X**		**X**	

1(d)(iii)	1 mark each for each correct column Accept –/X for empty spaces. Accept Y/X/Ticks as long as clear which are used	5

		Rules							
Conditions	Document printed, but quality is poor	**Y**	**Y**	**N**	**N**	**N**			
	Error light is flashing on printer				**Y**	**N**			
	Document printed, but paper size is incorrect	**Y**	**N**	**Y**	**N**	**N**			
Actions	Check connection from computer to printer				**X**				
	Check ink status	**X**	**X**						
	Check if there is a paper jam				**X**				
	Check paper size selected	**X**		**X**					

Answer 31

2	1 mark for each highlighted section

Label	Op Code	Operand	Comment	
LOOP:	LDD	ANSWER	// Load the value from ANSWER	
	ADD	NUMONE	// Add the value from NUMONE	[1]
	STO	ANSWER		[1]
	LDD	COUNT	// Load the value from COUNT	[1]
	INC	ACC	// Increment the Accumulator	[1]
	STO	COUNT		[1]
	CMP	NUMTWO	// Is NUMTWO = COUNT?	[1]
	JPN	LOOP	// If false, jump to LOOP	[1]
	LDD	ANSWER	// Load the value from ANSWER	[1]
	OUT		// output ANSWER to the screen	[1]
	END		// End of program	
NUMONE:	2			
NUMTWO:	4			
COUNT:	0			
ANSWER:	0			

Answer 32

3	1 mark per bullet point to **max 3** • Logic error // it is programmed incorrectly • There was an error in the design // the correct requirements were not stated • Run-time error // division by 0 // stack overflow // end of file reached // library not available // linking/loading error • Not adequately/correctly tested	**3**

Answer 33

4	1 mark for each term. **Definition** / **Term** Software is tested by an in-house team of dedicated testers. — **Alpha testing/black-box/white-box** Software is tested by the customer before it is signed off. — **Acceptance testing** Software is tested by a small selection of users before general release — **Beta testing**	**3**

Answer 35

1(a)	1 mark per fact
	
	`14 direct(london, rome).`
	`15 flies(rome, british_air).`

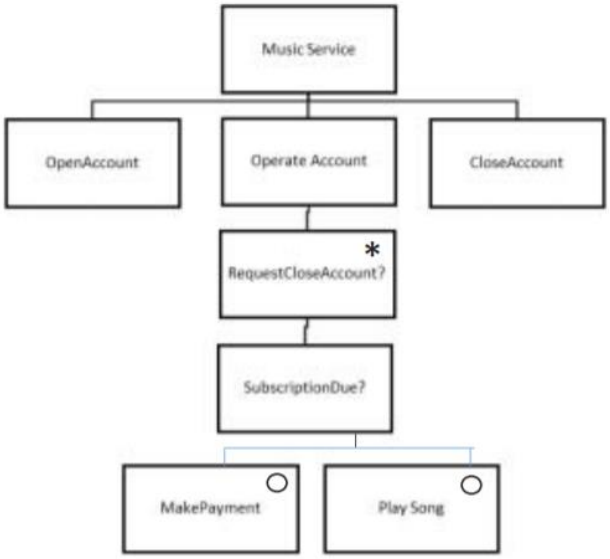
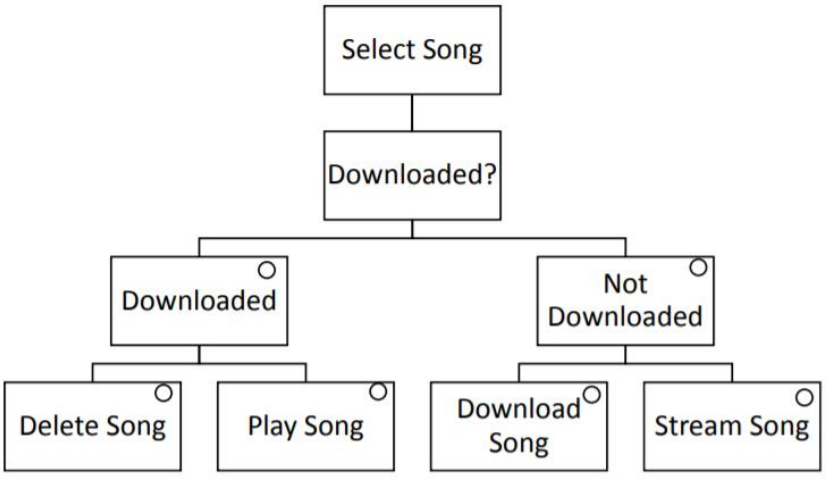
1(b)	1 mark per bullet: • palma • salzburg `K = palma, salzburg`
1(c)	1 mark per bullet: • direct • glasgow, M `direct(glasgow, M).`
1(d)	1 mark per bullet: • `flies(Z,X)` • `AND` • `direct(Z, Y)` `flies(Z, X) AND direct(Z, Y)`
1(e)	`YES`

Answer 36

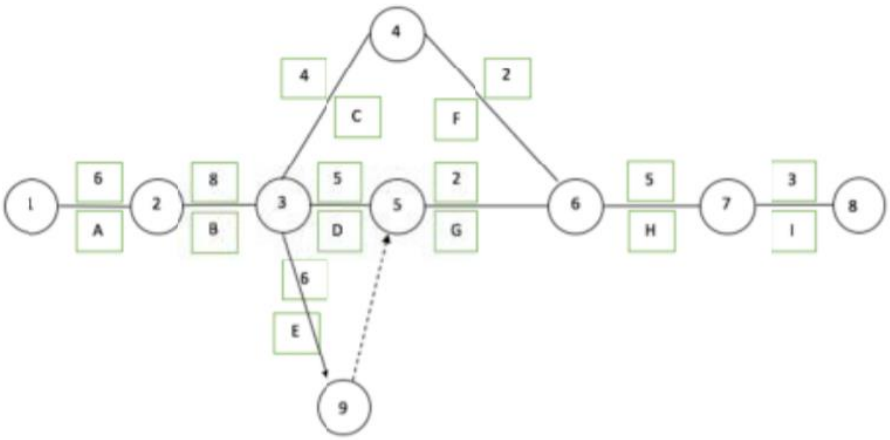
2(a)	1 mark for each completed statement 01 MaxIndex ← 20 02 NumberItems ← **MaxIndex - 1 // 19** 03 FOR Outer ← 1 TO **MaxIndex - 1 // 19** 04 FOR Inner ← 1 to NumberItems 05 IF ItemList[Inner] > **ItemList[Inner + 1]** 06 THEN 07 Temp ← ItemList[**Inner**] 08 ItemList[Inner] ← ItemList[**Inner + 1**] 09 ItemList[Inner + 1] ← **Temp** 10 ENDIF 11 ENDFOR 12 NumberItems ← **NumberItems - 1** 13 ENDFOR
2(b)(i)	1 mark per bullet • Iterations continue // it continues doing comparisons • …after the array is sorted
2(b)(ii)	1 mark per bullet to max 3 • Use of a flag to indicate if any swaps have taken place • If the inner loop has made all comparisons with no changes • …flag/value set accordingly • A comparison checks the flag/value at the end of each inner loop • …if it is sorted it breaks out/stops

2(c)	1 mark per bullet to max 4 e.g. • When the list is **almost** sorted … • …because it will stop as soon as it is sorted • When there are a large number of data items … • …because it will perform fewer comparisons/loops

Answer 37

3(a)	1 mark per bullet • OpenAccount, OperateAccount and Close Account on same level • RequestCloseAccount under OperateAccount • …SubscriptionDue under RequestCloseAcount • …Make Payment and PlaySong under SubscriptionDue • Correct selection and iteration throughout 
3(b)	1 mark per bullet • Downloaded? • Download and Not downloaded beneath downloaded • Delete song and play song beneath Downloaded (2) **and** Download song and stream song beneath not downloaded • All selections correct 

Answer 38

4(a)	1 mark per bullet • C, D and E all coming from 3 • G following D and E • F following C • H from 6 to 7 • I from 7 to 8 
4(b)	1 mark per bullet • A→B→E→G→H→I • 30 days
4(c)	1 mark per bullet • Earliest start time: 19 days • Latest finish time: 22 days

Answer 39

<table>
<tr><td>6(a)(i)</td><td>1 mark per bullet:

• TYPE ListNode declaration and ENDTYPE
• DECLARE Player : String
• DECLARE Pointer : INTEGER

<pre>TYPE ListNode
   DECLARE Player : STRING
   DECLARE Pointer : INTEGER
ENDTYPE</pre></td><td>3</td></tr>
<tr><td>6(a)(ii)</td><td>1 mark per bullet:

• DECLARE Scorers : ARRAY[0:9]
• OF ListNode

<pre>DECLARE Scorers : ARRAY[0:9] OF ListNode</pre></td><td>2</td></tr>
<tr><td>6(b)</td><td>1 mark for each completed statement

<pre>FUNCTION SearchList(Find, Position) RETURNS INTEGER
  IF Scorer[Position].Player = Find
    THEN
      RETURN Position
    ELSE
      IF Scorer[Position].Player <> -1
        THEN
           Position ← SearchList(Find, Scorer[Position].Pointer)
           RETURN Position
         ELSE
           RETURN 99
      ENDIF
  ENDIF
ENDPROCEDURE</pre></td><td>5</td></tr>
</table>

Answer 40

1(a)

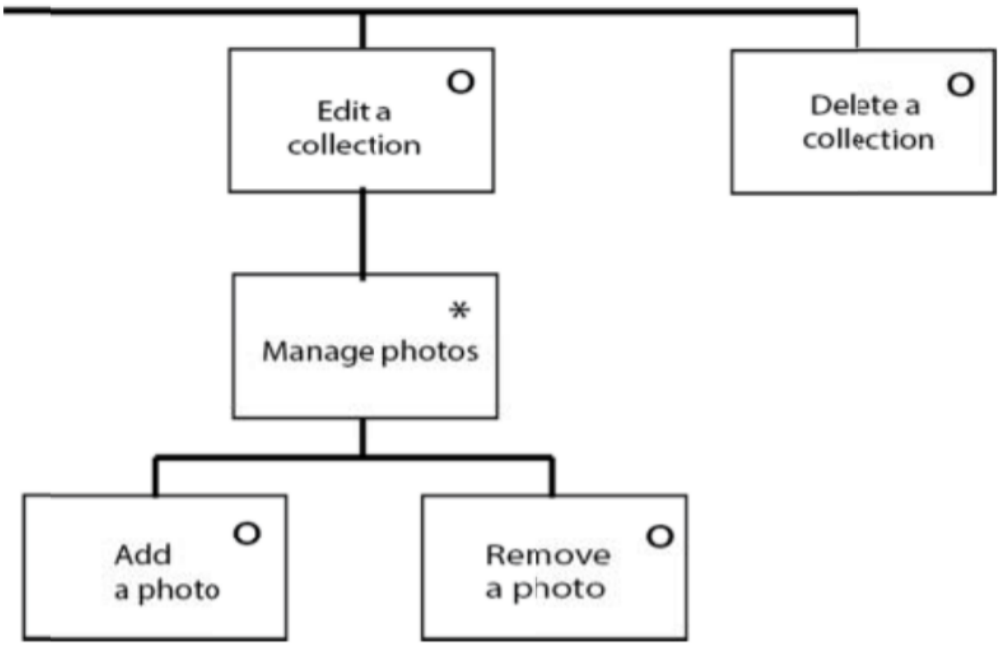
1 mark for each correctly completed pseudocode line to max 4

```
01        REPEAT
02           CALL TakePhoto
03           CALL AddPhotoToCollection
04           OUTPUT "Do you want to take another photo?"
05           INPUT AddPhoto
06        UNTIL AddPhoto = "No"
07        REPEAT
08           CALL AddUser
09           OUTPUT "Do you want to add another user?"
10           INPUT NewUser
11        UNTIL NewUser = "No"
12        OUTPUT "Do you want to create another collection?"
13        INPUT NewCollection
14     UNTIL NewCollection = "No"
```

1(b)

1 mark per bullet:

- Edit a collection // Choose a collection // Select a collection
- Manage photos
- Delete a collection
- both add and remove a photo // add to collection, delete to collection
- appropriate selection and iteration in all boxes



Answer 41

2(a)	1 mark for each statement
	
	`15  is_a(gecko, lizard).`
	`16  maxsize(gecko, 182).`

2(b)	1 mark for 2 results 2 marks for 3 correct results `green_iguana, cayman, smooth_iguana`
2(c)	1 mark per bullet • `is_a` used with brackets () • `squamata, X` in correct order `is_a(squamata, X).`
2(d)	1 mark for each bullet to max 3 • `is_a(X, Z)` • `and // , has(Z, Y).` `is_a(X, Z) AND has(Z, Y).`
2(e)	`YES`

Answer 42

3(a)	`CardData` is **partially** sorted/ordered // more items in order/sorted

3(b)	1 mark for each correct statement
	
	`01 ArraySize ← 10`
	`02 FOR Pointer ← 2 TO` **`ArraySize // 10`**
	`03  ValueToInsert ← CardData[Pointer]`
	`04  HolePosition ←` **`Pointer`**
	`05  WHILE(HolePosition>1 AND(`**`CardData[HolePosition - 1]`** `>` **`ValueToInsert`**`))`
	`06  CardData[HolePosition] ← CardData[`**`HolePosition - 1`**`]`
	`07  HolePosition ←` **`HolePosition - 1`**
	`08  ENDWHILE`
	`09  CardData[HolePosition] ←` **`ValueToInsert`**
	`10 ENDFOR`

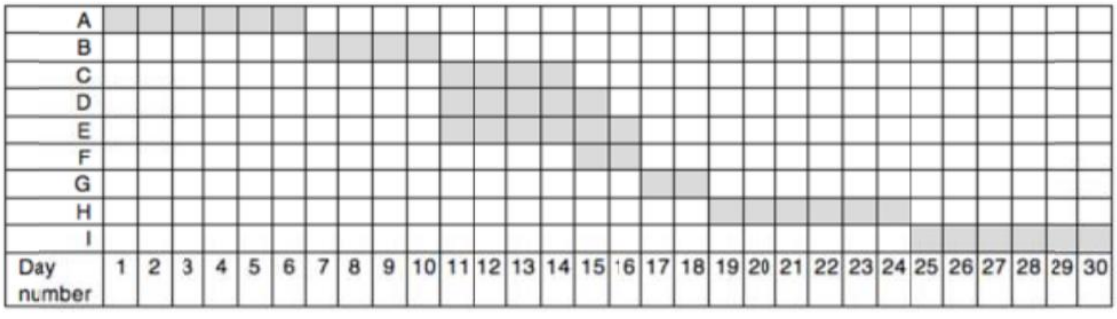
3(c)(i)	1 mark per bullet to max 2 • It doesn't check every value • The midpoint is the middle element, not the middle numerical value • When the higher/lower elements are discarded they will not be the higher/lower elements • It might discard the value you are looking for

3(c)(ii)	1 mark per bullet to max 4. Max 2 marks if no relation to `CardData` values. • Find mid-point **and** comparison // 25 is smaller than/compared to 52/56 • Discard/ignore greater // change upper bound to 33/52/midpoint - 1 //e.g. right hand side // only use array elements 1 - 4/5 • Find **and** compare to mid-point of new list e.g. 12/25 • Value is the midpoint // Continue until value found

3(d)	1 mark for each complete statement

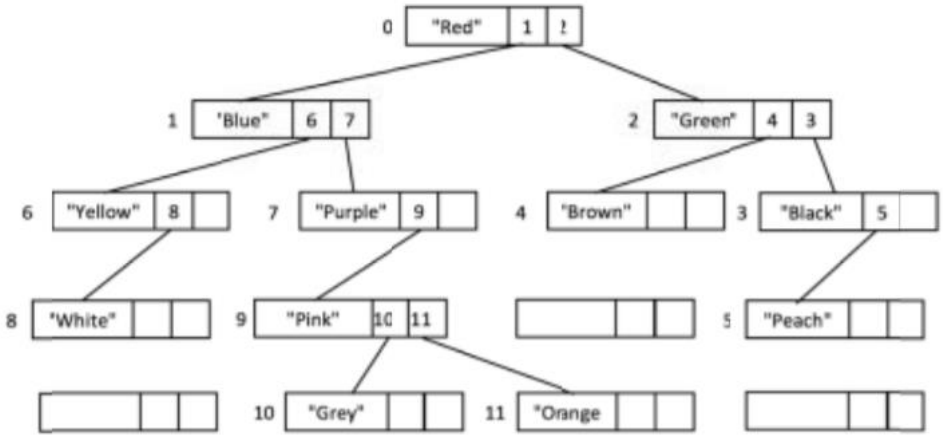
```
PROCEDURE BinarySearch(CardData, SearchValue)
   DECLARE Midpoint : INTEGER
   First ← 1
   Last ← ARRAYLENGTH(CardData)
   Found ← FALSE
   WHILE (First <= Last) AND NOT(Found)
      Midpoint ← (First + Last) \ 2
      IF CardData[Midpoint] = SearchValue
         THEN
            Found ← TRUE
        ELSE
            IF SearchValue < CardData[Midpoint]
               THEN
                  Last ← Midpoint - 1
               ELSE
                  First ← Midpoint + 1
            ENDIF
         ENDIF
    ENDWHILE
ENDPROCEDURE
```

Answer 43

5(a)	1 mark per bullet • **C**/**D**/**E** in parallel starting after **B**, with correct durations. • **F** with dependency on **C** and correct duration • **G** with dependency on **D** and **E** with correct duration • **H** with correct dependency on **F** and **G** • **I** with dependency on **H** 	**5**
5(b)(i)	**C, D, E**	**1**
5(b)(ii)	**E, F**	**1**

5(c)	1 mark per bullet to max 2 For example: • Check if the project is on track • …so the project manager can intervene if behind • lets you identify slack time to • …reallocate resources to support the process • find critical path • …to ensure activities are given correct priority • see when tasks end • ...to plan the next tasks • see which tasks can run in parallel • set milestones/goals • check correct tasks are being carried out on current day • Calculate latest start time • Calculate earliest finish time • Calculate latest start time for a task

Answer 44

6(a)	1 mark per bullet • Brown left and black right from node 2 • Yellow left and Purple right from node 1 • Peach left comes from 3 • White left from 6 **and** Pink left from 7 • Grey left from 9 and orange right from 9 

Answer 45

1(a)(i)	1 mark for each correct statement: • bird(lays_egg). • bird(has_wings).
1(a)(ii)	1 mark for each correct line: • feature(eagle, lays_eggs). • feature(eagle, has_wings).
1(b)(i)	1 mark for each animal: tuna, crab
1(b)(ii)	1 mark per bullet point: • feature() • tuna, C feature(tuna, C)

1(c)	1 mark per bullet point to max 3:
	
	• feature(X,Y) AND bird(Y) // feature(X, has_wings)
	• AND
	• feature(X,Z) AND bird(Z) // feature(X, lays_eggs)
	
	(feature(X, Y) AND bird(Y)) AND (feature(X, Z) AND bird(Z))

1(d)(i)	A programming style/classification // characteristics/features that programming language has/uses
1(d)(ii)	1 mark for each: • Low-level • Imperative // Procedural

Answer 46

3(a)

- LIFO / last in first out

3(b)(i)

Points to the **next** free space on the stack

3(b)(ii)

1 mark per bullet to max 3

- Correct stack contents
- StackPointer = 4

StackPointer	4	StackContents
	0	"Screw 1"
	1	"Screw 2"
	2	"Back case"
	3	"Light 1"
	4	
	5	
	6	
	7	

3(c)(i)

1 mark for each correct statement:

```
PROCEDURE POP
   IF StackPointer = 0
        THEN
            OUTPUT ("The stack is empty")
        ELSE
           StackPointer ← StackPointer - 1
           OUTPUT Parts[StackPointer]
            Parts(StackPointer) ← "*"
   ENDIF
ENDPROCEDURE
```

3(c)(ii)

1 mark for each completed statement:

```
PROCEDURE PUSH (BYVALUE Value : String)
   IF StackPointer > 19
        THEN
           OUTPUT "Stack full"
        ELSE
           Parts[StackPointer] ← Value
           StackPointer ← StackPointer + 1
   ENDIF
ENDPROCEDURE
```

Answer 47

<table>
<tr><td>4(a)(i)</td><td>A function/subroutine defined in terms of itself // a function/subroutine that calls itself</td><td>1</td></tr>
<tr><td>4(a)(ii)</td><td>06</td><td>1</td></tr>
<tr><td>4(b)</td><td>1 mark for each bullet point:

• –60 as final return value
• 3*2*1*–10

1 mark for each row in table

<table>
<tr><th>Call Number</th><th>Function call</th><th>Number = 0 ?</th><th>Return value</th></tr>
<tr><td>1</td><td>Calculate(3)</td><td>False</td><td>3*Calculate(2)</td></tr>
<tr><td>2</td><td>Calculate(2)</td><td>False</td><td>2*Calculate(1)</td></tr>
<tr><td>3</td><td>Calculate(1)</td><td>False</td><td>1*Calculate(0)</td></tr>
<tr><td>4</td><td>Calculate(0)</td><td>TRUE</td><td>–10</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
</table></td><td>6</td></tr>
<tr><td>4(c)(i)</td><td>1 mark per bullet point:

• Each time it calls itself the variables are put onto the stack // The function call itself too many times
• … it runs out of stack space // stack overflow</td><td>2</td></tr>
<tr><td>4(c)(ii)</td><td>1 mark per bullet point to max 5:

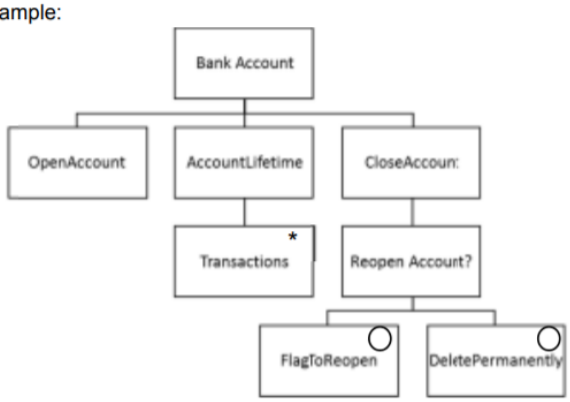
• Function header with parameter and Returning calculated value
• Loop parameter times (up to number, or down from number)…
• …Multiplying by loop counter
• Multiplying by –10
• Dealing with starting value correctly

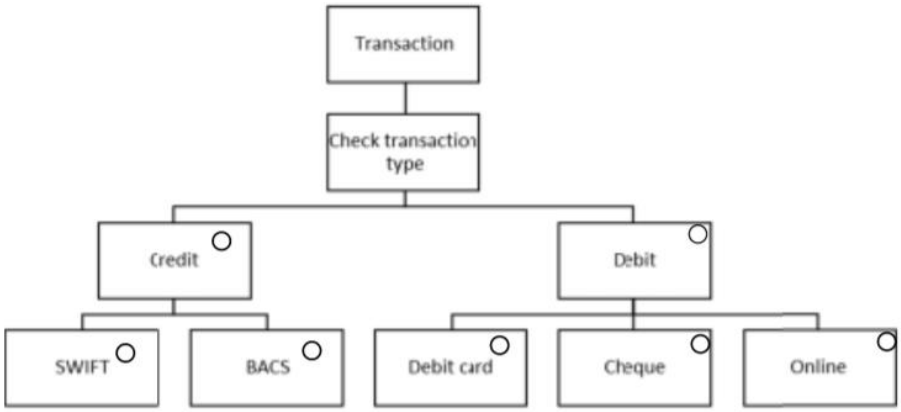
For example:

<pre>
FUNCTION Calculate(Number : INTEGER) RETURNS INTEGER
  DECLARE Count : INTEGER
  DECLARE Value : INTEGER
  Value ← -10
  FOR Count ← 1 to Number
    Value ← Value * Count
  ENDFOR

  RETURN Value
ENDFUNCTION
</pre></td><td>5</td></tr>
</table>

Answer 48

1(a)	1 mark per bullet point: • `OpenAccount`, `AccountLifetime` and `CloseAccount` below `Bank Account Transactions` below `AccountLifetime` • …with iteration • `Reopen Account?` below `Close Account` • `FlagToReopen` and `DeletePermanently` below `ReopenAccount?` • …with selection on both Example: 

1(b)	1 mark per bullet point: • `Credit` and `debit` below `Transaction` • `Swift` and `BACS` below `Credit` • `Debit`, `Cheque` and `Online` below `Debit` • Correct selections where needed and no additional selection/iteration 

Answer 49

2(a)	1 mark for each fact:
	
	`type(waterdog, gundog).`
	`is_a(standardpoodle, waterdog).`

2(b)	1 mark for each result: `H = english_setter, irish_setter`
2(c)	1 mark per bullet point to max 2: • `is_a` • `(irish_setter, W)` `is_a(irish_setter, W)`
2(d)	1 mark per bullet point to max 3: • `is_a(X,Z)` • `AND` • `fav_bird(Z, Y).` `fav_bird(X, Y) IF is_a(X, Z) AND fav_bird(Z, Y).`
2(e)	NO

Answer 50

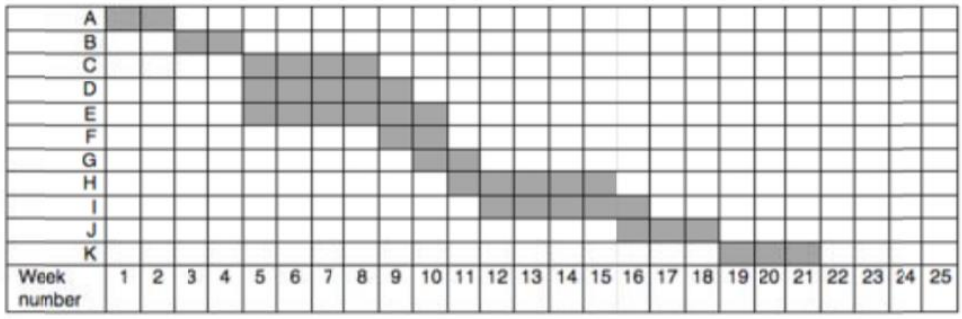
3(a)	1 mark for each completed statement:
	
	`01 FOR Outer ← LENGTH(List)-1 TO 0 STEP -1`
	`02  FOR Inner ← 0 TO (Outer - 1)`
	`03  IF List[Inner] > List[Inner + 1]`
	`04  THEN`
	`05  Temp ← List[Inner]`
	`06  List[Inner] ← List[Inner + 1]`
	`07  List[Inner + 1] ← Temp`
	`08  ENDIF`
	`09  ENDFOR`
	`10 ENDFOR`

3(b)(i)	Ascending (must match answer to 3(a))

3(b)(ii)	Line 03 Change the operator in the IF statement to < or <= rather than >

3(c)	1 mark per bullet
	
	• Use of a (Boolean) flag…
	• …Remainder of bubble correct
	• Set flag when a swap has been made…
	• …Loop until a swap has not been made **and** then exit all loops
	
	`Outer ← LENGTH(List)-1`
	`REPEAT`
	`  Inner ← 0`
	`  Swap ← FALSE`
	`  REPEAT`
	`  IF List[Inner] > List[Inner + 1]`
	`  THEN`
	`  Temp ← List[Inner]`
	`  List[Inner] ← List[Inner + 1]`
	`  List[Inner + 1] ← Temp`
	`  Swap = TRUE`
	`  ENDIF`
	`  Inner ← Inner + 1`
	`  UNTIL Inner = Outer - 1`
	`  Outer ← Outer - 1`
	`UNTIL Swap = FALSE OR Outer = 0`

Answer 51

5(a)	1 mark per bullet • C, D and E start at same point after B • F follows C • G follows D **and** H follows F • I follows G **and** J follows H • K follows J  	**5**
5(b)	1 mark per bullet point to max 2: Example: • Teams can work on simultaneous/concurrent/parallel tasks • Any example of two teams working simultaneously e.g. From step 3, each team can work on a different task in week 4, 5 and 6 // tasks C, D, E can be split between different teams • Any example of working on same activities //e.g. all teams can work together on 1–2 / A, 2–3/B • Any example of working on dependent activities // e.g. if all teams work on B, then they can split up for CDE	**2**
5(c)(i)	A,B,E,H,J,K	**1**

5(c)(ii)	1 mark per bullet point to max 2: Example: • If there is any delay to a task which is part of the critical path • … then the overall project will be delayed • Gives the earliest possible completion time • … this allows you to organise/allocate resources (efficiently) • Frequently recalculate the critical path … • … to see if there are any delays/new critical path has arisen • Can identify where there is slack in activities • … so they can start later without affecting the critical path	**2**

Answer 52

6(a)(i)

1 mark per bullet point:

- Declaring a record type // class etc.
- Declaring Country as a string
- Declaring Pointer as an integer

Example:

```
TYPE ListElement
   DECLARE Country : STRING
   DECLARE Pointer : INTEGER
ENDTYPE
```

6(a)(ii)

1 mark per bullet point:

- Declaring `CountryList` as an array with 15 elements
- Of type `ListElement`

Example:

```
DECLARE CountryList : ARRAY[1 : 15] OF ListElement
```

6(b)

1 mark for each completed statement

```
PROCEDURE DeleteNode(NodeValue: STRING, ThisPointer: INTEGER, PreviousPointer: INTEGER)
 IF CountryList[ThisPointer].Value = NodeValue
   THEN
     CountryList[ThisPointer].Value ← ""
     IF ListHead = ThisPointer
      THEN
         ListHead ← CountryList[ThisPointer].Pointer
      ELSE
         CountryList[PreviousPointer].Pointer ← CountryList[ThisPointer].Pointer
     ENDIF
   CountryList[LastNode].Pointer ← ThisPointer
   LastNode ← ThisPointer
       CountryList[ThisPointer].Pointer ← -1
   ELSE
      IF CountryList[ThisPointer].Pointer <> -1
        THEN
          CALL DeleteNode(NodeValue,CountryList[ThisPointer].Pointer, ThisPointer)
        ELSE
          OUTPUT "DOES NOT EXIST"
      ENDIF
  ENDIF
ENDPROCEDURE
```