

AS computer science summative assessment 2

Time: 60 minutes Total: 49 Marks

Name: _____ Marks: _____

- 1 A programmer wants to write a program to calculate the baggage charge for a passenger's airline flight.

Two types of ticket are available for a flight:

- economy class (coded E)
- standard class (coded S)

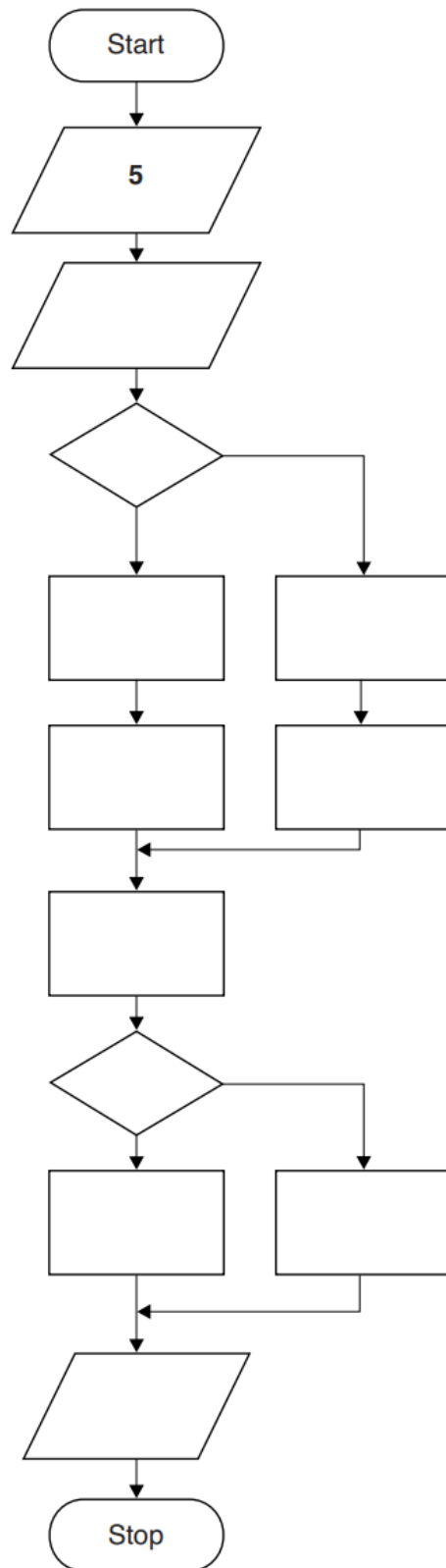
Each ticket type has a baggage weight allowance as shown below. The airline makes a charge if the weight exceeds the allowance.

Ticket type	Baggage allowance (kg)	Charge rate per additional kg (\$)
'E'	16	3.50
'S'	20	5.75

- (a) A program flowchart will document the program. The flowchart will contain the following statements:

Statement number	Statement
1	Charge \leftarrow 0
2	INPUT BaggageWeight
3	Charge \leftarrow ExcessWeight * ChargeRate
4	Is ExcessWeight > 0 ?
5	INPUT TicketType
6	ExcessWeight \leftarrow BaggageWeight - BaggageAllowance
7	BaggageAllowance \leftarrow 16
8	ChargeRate \leftarrow 3.5
9	OUTPUT Charge
10	ChargeRate \leftarrow 5.75
11	BaggageAllowance \leftarrow 20
12	Is TicketType = 'E' ?

Complete the flowchart by putting the appropriate **statement number** in each flowchart symbol. Statement 5 has been done for you.



[6]

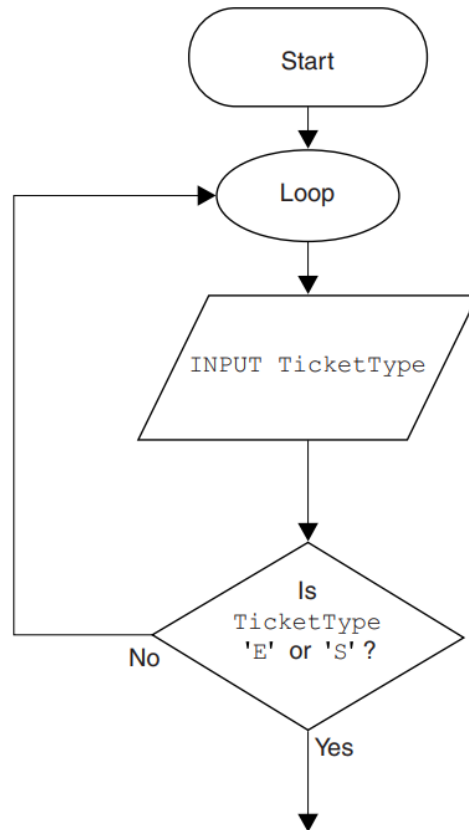
(b) The programmer needs data to test the flowchart.

Complete the table of test data below to show **five** tests.

TicketType	BaggageWeight	Explanation	Expected output
E	15	
		
		
		
		

[5]

- (c) The program design is to be amended. The value input by the user for the ticket type is to be validated. Part of the amended flowchart is shown below.



Write **pseudocode** to use a pre-condition loop for this validation.

.....

.....

.....

.....

.....

.....[3]

A company employs Ahmed as a programmer.

(a) At College, before joining the company, Ahmed used two items of software for programming:

- a text editor
- a compiler

Describe how he could have developed programs using these software tools.

Include in the description the terms 'object code' and 'source code'.

.....

.....

.....

.....

.....

.....

.....[3]

(b) Ahmed now uses an Integrated Development Environment (IDE) for programming.

(i) State **one** feature an IDE provides to help with the identification of syntax errors.

.....

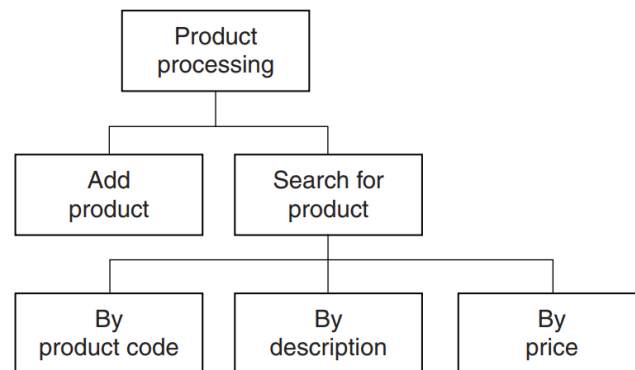
.....[1]

(ii) State **one** feature an IDE provides to carry out white box testing.

.....

.....[1]

(c) The company maintains a file of product data. Ahmed is to write a program to add a new product and search for a product based on the structure diagram shown:



The program records the following data for each product:

- product code
- product description
- product retail price

The text file `PRODUCTS` stores each data item on a separate line, as shown below:

File	PRODUCTS
0198	
	Plums (10kg)
11.50	
0202	
	Onions (20kg)
10.00	
0376	
	Mango chutney (1kg)
02.99	
0014	
	Mango (10kg)
12.75	

The program uses the variables shown in the identifier table.

Identifier	Data type	Description
<code>PRODUCTS</code>	TEXT FILE	Storing the code, description and retail price for all current products
<code>PCode</code>	ARRAY[1:1000] OF STRING	Array storing the product codes
<code>PDescription</code>	ARRAY[1:1000] OF STRING	Array storing the product descriptions
<code>PRetailPrice</code>	ARRAY[1:1000] OF REAL	Array storing the product retail prices
<code>i</code>	INTEGER	Array index used by all three arrays

- (i) The first operation of the program is to read all the product data held in file `PRODUCTS` and write them into the three 1D arrays.

Complete the pseudocode below.

```

OPEN .....

i ← 1

WHILE .....

    READFILE ("PRODUCTS", ..... )
    READFILE ("PRODUCTS", ..... )
    READFILE ("PRODUCTS", ..... )
    .....
    .....

ENDWHILE

CLOSE "PRODUCTS"



OUTPUT "Product file contents written to arrays"

```

[5]

When Ahmed designed the `PRODUCTS` file, he considered the alternative file structure shown opposite.

It stores one product per line in the text file.

File <code>PRODUCTS</code>		
0198	Plums (10kg)	11.50
0202	Onions (20kg)	10.00
		
0376	Mango chutney (1kg)	02.99
		
0014	Mango (10kg)	12.75

- (ii) State **one** benefit and **one** drawback of this file design.

Benefit

.....

Drawback

.....[2]

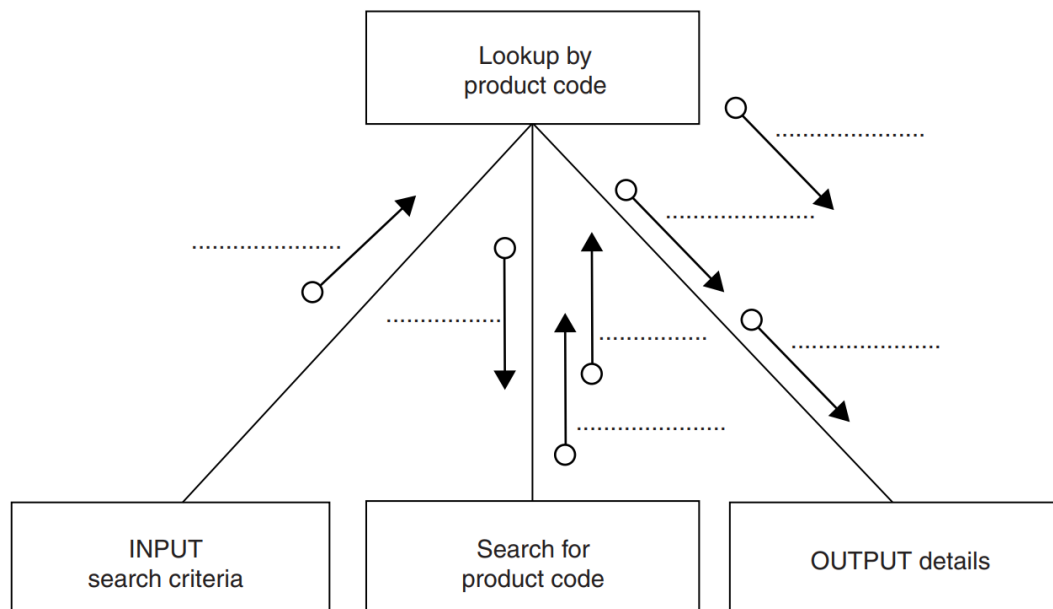
- (d) To code the 'Search by product code' procedure, Ahmed draws a structure chart showing the different stages.

The procedure uses the variables shown in the identifier table.

Identifier	Data type	Description
SearchCode	STRING	Product code input by the user
ThisIndex	INTEGER	Array index position for the corresponding product
ThisDescription	STRING	Product description found
ThisRetailPrice	REAL	Product retail price found

You can assume that before the procedure is run, all the product data is read from file **PRODUCTS** and then stored in three 1D arrays as described in **part (c)(i)**.

Label the structure chart to show the input(s) and output(s).



[4]

- (e) A first attempt was made at writing the ‘Search for product code’ module. Ahmed designs this as a function `ProductCodeSearch`.

The function returns an integer value as follows:

- if the product code is found, it returns the index position of the 1D array `PCode` being searched
- if the product code is not found, the function returns `-1`

Write **program code** for function `ProductCodeSearch`.

Visual Basic and Pascal: You should include the declaration statements for variables.

Python: You should show a comment statement for each variable used with its data type.

Programming language

[6]

3.

A firm employs workers who assemble amplifiers. Each member of staff works an agreed number of hours each day.

The firm records the number of completed amplifiers made by each employee each day.

Management monitor the performance of all its workers.

Production data was collected for 3 workers over 4 days.

Daily hours worked		Production data			
		Worker 1	Worker 2	Worker 3	
Worker 1	5				
Worker 2	10				
Worker 3	10				
		Day 1	10	20	9
		Day 2	11	16	11
		Day 3	10	24	13
		Day 4	14	20	17

A program is to be written to process the production data.

- (a) The production data is to be stored in a 2-dimensional array `ProductionData`, declared as follows:

```
DECLARE ProductionData ARRAY[1:4, 1:3] : INTEGER
```

- (i) Describe **two** features of an array.

1

 2
[2]

- (ii) Give the value of `ProductionData[3, 2]`.

.....[1]

- (iii) Describe the information produced by the expression:

```
ProductionData[2, 1] + ProductionData[2, 2] + ProductionData[2, 3]
```

.....
[2]

(b) Complete the trace table for the pseudocode algorithm below.

```

FOR WorkerNum ← 1 TO 3
    WorkerTotal[WorkerNum] ← 0
ENDFOR

FOR WorkerNum ← 1 TO 3
    FOR DayNum ← 1 TO 4
        WorkerTotal[WorkerNum] ← WorkerTotal[WorkerNum] +
                                ProductionData[DayNum, WorkerNum]
    ENDFOR
ENDFOR

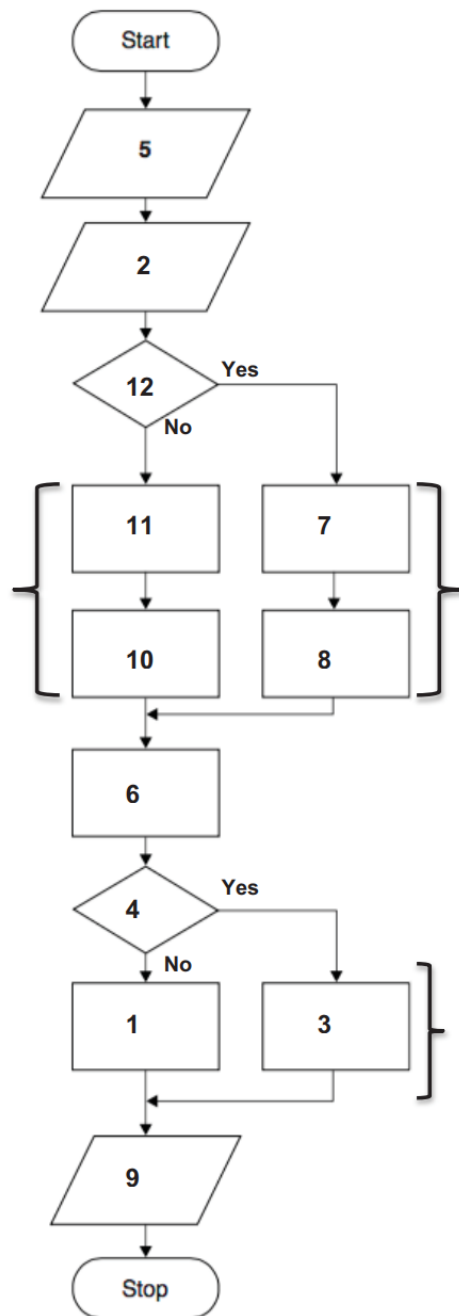
FOR WorkerNum ← 1 TO 3
    WorkerAverage ← WorkerTotal[WorkerNum]/
                    (4 * DailyHoursWorked[WorkerNum])
    IF WorkerAverage < 2
        THEN
            OUTPUT "Investigate", WorkerNum
        ENDFIF
ENDFOR

```

[illegible][illegible]

[8]

1 (a)



Note: Order of 11, 10
and 7,8 may be
reversed.

One mark for each of the following symbols / symbol combinations:

- 2
- 7 and 8 from YES
- 10 and 11
- 6
- 1 and 3 (1 from NO, 3 from YES)
- 9
- 12 and 4

Max [6]

(b) Rows 2 to 7 are examples only

TicketType	BaggageWeight	Explanation	Expected output
E	15	Under the allowance	0
E	> 16	Under the allowance	Charge
S	<= 20	Under the allowance	0
S	> 20	Under the allowance	Charge
E	16	Boundary weight for a type E ticket	0
S	20	Boundary weight for a type S ticket	0
E or S	negative or non-numeric	Invalid weight	Error message

Ticket type	Baggage allowance (kg)	Charge rate per additional kg (\$)
'E'	16	3.50
'S'	20	5.75

One mark for each different test (examples above)

Max [5]

(c) `INPUT TicketType`
`WHILE NOT (TicketType = 'E') OR (TicketType = 'S')`
`INPUT TicketType`
`ENDWHILE`

One mark for each of:

- `WHILE ... ENDWHILE`
- Correct condition in a loop
- `INPUT` within loop plus one before loop // alternative arrangement leading to correct exit from loop

[3]

2.

- (a) • Create / modify the source code using the text editor
 • Compiler translates the source code
 • Compiler produces the object code

[Max 3]

- (b) (i) • Errors in keywords are highlighted // before the compilation process
 • Provides line-by-line syntax checking as code is typed in
 • Provides line number of the error
 • Display of known identifier names
 • Auto-complete
 • Colour-coding
 • Auto-indent
 • type checking
 • Subroutine parameter checking

[Max 1]

- (ii) • Set break-points
 • Single step / step into/over subroutine
 • Window to watch the changing value of variables

[Max 1]

(c) (i) OPEN "PRODUCTS" FOR READ
 i ← 1
 WHILE NOT EOF("PRODUCTS")

 READFILE ("PRODUCTS", PCode[i])
 READFILE ("PRODUCTS", PDescription[i])
 READFILE ("PRODUCTS", Temp // PRetailPrice[i])

 PRetailPrice[i] ← TONUM(Temp)

 i ← i + 1
 ENDWHILE

 CLOSE "PRODUCTS"
 OUTPUT "Product file contents written to arrays"

One mark per bold phrase (three READFILE() counts as a single mark)

[5]

(ii) Benefit:

- The number of file read operations is reduced (by 2/3rds)
- It may use less storage / space in the file if strings are NOT fixed length
- All the data related to a single product is read at once / in one file operation / grouped together

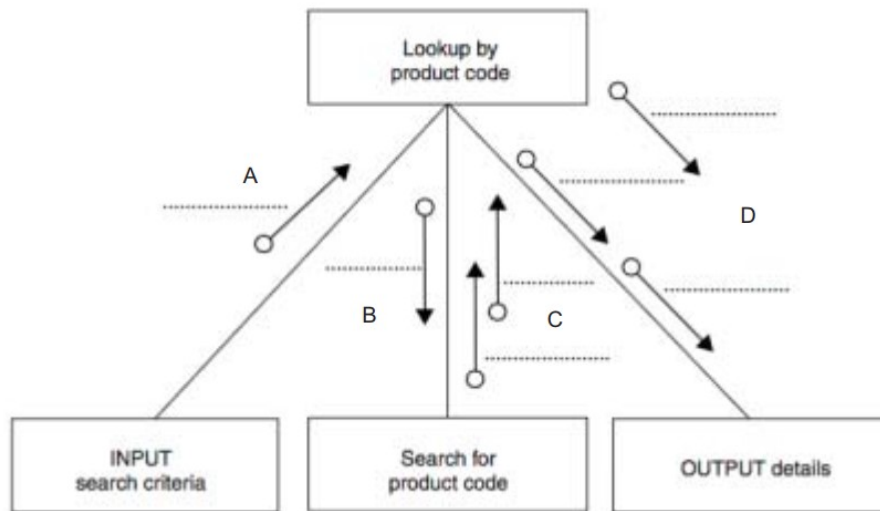
Drawback:

- The program will need to use the string handling functions to isolate each of the three items of data
- Difficult to isolate data items if the format is not consistent
- More difficult to search

Max one benefit and one drawback

[2]

(d)



One mark per group (one or more names) as follows:

- A: SearchCode
- B: SearchCode // ThisIndex
- C: ThisRetailPrice, ThisDescription
- D: SearchCode, ThisDescription, ThisRetailPrice

[4]

(e) 'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix.

```
FUNCTION ProductCodeSearch(AnyName : String) RETURNS : Integer
    DECLARE FoundPos : Integer
    DECLARE i : Integer

    i ← 1
    FoundPos ← -1

    REPEAT
        IF AnyName = PCode[i]
            THEN
                FoundPos ← i
            ELSE
                i ← i + 1
            ENDIF
    UNTIL (i = 1001) OR (FoundPos <> -1)

    RETURN FoundPos

ENDFUNCTION
```

Mark as follows:

- Function header returns INTEGER
- Initialisation of index variable
- Loop through array PCode (including exit when found)
- Comparison of AnyName with PCode[i] in a loop
- Increment index variable in a loop
- Return index if AnyName found AND return -1 if AnyName not found

[Max 6]

3.

- (a) (i) • Set of data items have a common name (1 mark)
 • Items are referenced using a subscript/index (1 mark)
 • Accept: all data items are of the same data type (1 mark) [max 2]
- (ii) 24 [1]
- (iii) • The total number of amplifiers 'produced' by workers 1, 2 and 3/three workers (1 mark)
 • on day 2_ (1 mark) [2]

(b)

				WorkerTotal		
WorkerNum	DayNum	WorkerAverage	OUTPUT	1	2	3
1				0		
2					0	
3						0
1	1			10		
	2			21		
	3			31		
	4			45		
2	1				20	
	2				36	
	3				60	
	4				80	
3	1					9
	2					20
	3					33
	4					50
1		2.25				
2		2				
3		1.25	INVESTIGATE 3			

[8]