Cambridge Assessment International Education

# Cambridge International AS & A Level

| | |
|---|---|
| CANDIDATE NAME | |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| CENTRE NUMBER | | | | | | CANDIDATE NUMBER | | | | | |

*3033251826*

**COMPUTER SCIENCE** **9618/22**

Paper 2 Fundamental Problem-solving and Programming Skills **May/June 2023**

**2 hours**

You must answer on the question paper.

You will need: Insert (enclosed)

**INSTRUCTIONS**

- Answer **all** questions.
- Use a black or dark blue pen.
- Write your name, centre number and candidate number in the boxes at the top of the page.
- Write your answer to each question in the space provided.
- Do **not** use an erasable pen or correction fluid.
- Do **not** write on any bar codes.
- You may use an HB pencil for any diagrams, graphs or rough working.
- Calculators must **not** be used in this paper.

**INFORMATION**

- The total mark for this paper is 75.
- The number of marks for each question or part question is shown in brackets [ ].
- No marks will be awarded for using brand names of software packages or hardware.
- The insert contains all the resources referred to in the questions.

This document has **20** pages. Any blank pages are indicated.

Refer to the **insert** for the list of pseudocode functions and operators.

**1** A program calculates the postal cost based on the weight of the item and its destination. Calculations occur at various points in the program and these result in the choice of several possible postal costs. The programmer has built these postal costs into the program.

For example, the postal cost of $3.75 is used in the following lines of pseudocode:

```
IF Weight < 250 AND ValidAddress = TRUE THEN
   ItemPostalCost ← 3.75      // set postal cost for item to $3.75
   ItemStatus ← "Valid"       // item can be sent
ENDIF
```

**(a)** **(i)** Identify a more appropriate way of representing the postal costs.

Use of constants ........................................................................................ [1]

**(ii)** Describe the advantages of your answer to **part (a)(i)** with reference to this program.

1 Postal rates are entered once only
2 Avoids input error / changing the cost accidentally // avoids different values for postal rates at different points in the program
3 When required, the constant representing the postal rate value is changed once only // easier to maintain the program when the postal rates change
4 Makes the program easier to understand

........................................................................................................................

........................................................................................................................ [3]

**(b)** The lines of pseudocode contain features that make them easier to understand.

State **three** of these features.

1 ....
- Indentation
- White space

2 ....
- Comments
- Sensible / meaningful variable names // use of Camel Case

3 ....
- Capitalised keywords

[3]

**(c)** Give the **appropriate** data types for the following variables:

`ValidAddress` .................
- BOOLEAN

`ItemPostalCost` .............
- REAL

`ItemStatus` ....................
- STRING

[3]

**2** A program stores a user's date of birth using a variable `MyDOB` of type `DATE`.

**(a)** Write a pseudocode statement, using a function from the **insert**, to assign the value corresponding to 17/11/2007 to `MyDOB`.

```
MyDOB ← SETDATE(17, 11, 2007)
```

[1]

**(b)** `MyDOB` has been assigned a valid value representing the user's date of birth.

Write a pseudocode statement to calculate the number of months from the month of the user's birth until the end of the year and to assign this to the variable `NumMonths`.

For example, if `MyDOB` contains a value representing 02/07/2008, the value 5 would be assigned to `NumMonths`.

```
NumMonths ← 12 - MONTH(MyDOB)
```

[2]

**(c)** The program will output the day of the week corresponding to `MyDOB`.

For example, given the date 22/06/2023, the program will output `"Thursday"`.

An algorithm is required. An array will be used to store the names of the days of the week.

Define the array **and** describe the algorithm in **four** steps.

Do **not** use pseudocode statements in your answer.

Array definition

- A (1D) array containing 7 elements
- of type STRING

Step 1

Step 2

Step 3

Step 4

Step1: Assign value "Sunday" to first element, "Monday" to second element etc.

Step2: Use the function `DAYINDEX()` to return / find the day number from `MyDoB`

Step3: Use the returned value as the array index / to access the element that contains the name / string

Step4: Output the element / name / string

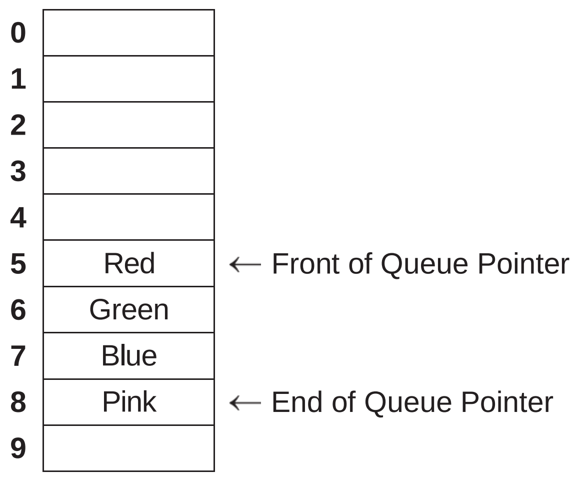
[6]

**3** A program stores data in a text file. When data is read from the file, it is placed in a queue.

**(a)** The diagram below represents an Abstract Data Type (ADT) implementation of the queue. Each data item is stored in a separate location in the data structure. During initial design, the queue is limited to holding a maximum of 10 data items.

The operation of this queue may be summarised as follows:

1 The Front of Queue Pointer points to the next data item to be removed.
2 The End of Queue Pointer points to the last data item added.
3 The queue is circular so that locations can be reused.



**(i)** Describe how the data items Orange and Yellow are added to the queue shown in the diagram.

1 Check that the queue is not full
2 EoQ pointer will move to point to location 9
3 Data item Orange will be stored in location referenced by EoQ pointer
4 EoQ pointer will move to point to location 0
5 Data item Yellow will be stored in location referenced by EoQ pointer

[4]

**(ii)** The following diagram shows the state of the queue after several operations have been performed. All queue locations have been used at least once.

| | | |
|---|---|---|
| **0** | D4 | |
| **1** | D3 | ← End of Queue Pointer |
| **2** | D27 | |
| **3** | D8 | |
| **4** | D33 | |
| **5** | D17 | ← Front of Queue Pointer |
| **6** | D2 | |
| **7** | D1 | |
| **8** | D45 | |
| **9** | D60 | |

State the number of data items in the queue.

7 [1]

**(b)** The design of the queue is completed and the number of locations is increased.

A function `AddToQueue()` has been written. It takes a string as a parameter and adds this to the queue. The function will return `TRUE` if the string was added successfully.

A procedure `FileToQueue()` will add each line from the file to the queue. This procedure will end when all lines have been added or when the queue is full.

Describe the algorithm for procedure `FileToQueue()`.

Do **not** use pseudocode in your answer.

1 Open file in `READ` mode
2 Loop to `EOF()` // read / process all the lines in file
3 Loop will end when return value from `AddToQueue()` is `FALSE` / queue is full
4 Read a line from the file in a loop
5 Pass string to `AddToQueue()` // `AddToQueue()` is executed with line as parameter

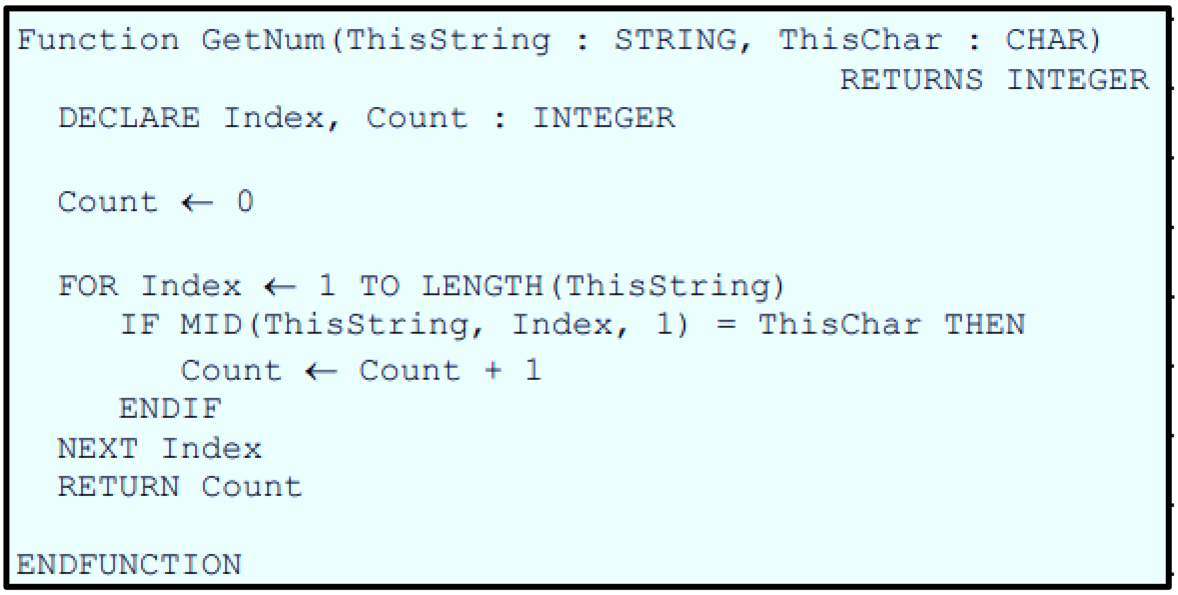
[5]

**4** A function `GetNum()` will:

1. take two parameters: a string and a character
2. count the number of times that the character occurs in the string
3. return the count.

Any comparison between characters needs to be case sensitive. For example, character '`a`' and character '`A`' are not identical.

Write pseudocode for function `GetNum()`.



```
Function GetNum(ThisString : STRING, ThisChar : CHAR)
                                          RETURNS INTEGER
  DECLARE Index, Count : INTEGER

  Count ← 0

  FOR Index ← 1 TO LENGTH(ThisString)
     IF MID(ThisString, Index, 1) = ThisChar THEN
        Count ← Count + 1
     ENDIF
  NEXT Index
  RETURN Count

ENDFUNCTION
```

[6]

**BLANK PAGE**

**5** A programmer has produced the following pseudocode to output the square root of the numbers from 1 to 10.

Line numbers are for reference only.

```
10  DECLARE Num : REAL
11  Num ← 1.0
...

40  REPEAT
41     CALL DisplaySqrt(Num)
42     Num ← Num + 1.0
43  UNTIL Num > 10
...

50  PROCEDURE DisplaySqrt(BYREF ThisNum : REAL)
51     OUTPUT ThisNum
52     ThisNum ← SQRT(ThisNum)        // SQRT returns the square root
53     OUTPUT " has a square root of ", ThisNum
54  ENDPROCEDURE
```

The pseudocode is correctly converted into program code.

Function `SQRT()` is a library function and contains no errors.

The program code compiles without errors, but the program gives unexpected results. These are caused by a design error.

**(a)** Explain why the program gives unexpected results.

- parameter / `Num` has been passed by reference // should have been passed by value
- so when the value / `ThisNum` is modified (in procedure `DisplaySqrt()`)
- the new value will be used in the loop (lines 40–43) // `Num` will be changed to modified value

[3]

**(b)** Explain why the compiler does **not** identify this error.

- The rules of the language have not been broken // there are no syntax errors

[1]

**(c)** Describe how a typical Integrated Development Environment (IDE) could be used to identify this error.

- Set a breakpoint to stop the program at a certain line / statement / point
- Step through the program line by line / statement by statement
- checking the value of `'num'` / a variable using a report / watch window

[3]

**(d)** The pseudocode is converted into program code as part of a larger program.

During compilation, a complex statement generates an error.

The programmer does not want to delete the complex statement but wants to change the statement so that it is ignored by the compiler.

State how this may be achieved.

- Change the statement into a comment
- Change the statement to a string representing a literal value and assign it to a variable / output it

[1]

**6** A procedure `Square()` will take an integer value in the range 1 to 9 as a parameter and output a number square.

The boundary of a number square is made up of the character representing the parameter value. The inside of the number square is made up of the asterisk character (`*`).

| **Parameter value** | **1** | **2** | **3** | **4** | **...** | **9** |
|---|---|---|---|---|---|---|
| **Output** | `1` | `22`<br>`22` | `333`<br>`3*3`<br>`333` | `4444`<br>`4**4`<br>`4**4`<br>`4444` | ... | `999999999`<br>`9*******9`<br>`9*******9`<br>`9*******9`<br>`9*******9`<br>`9*******9`<br>`9*******9`<br>`9*******9`<br>`999999999` |

The pseudocode `OUTPUT` command starts each output on a new line. For example, the following three `OUTPUT` statements would result in the outputs as shown:

```
OUTPUT "Hello"
OUTPUT "ginger"
OUTPUT "cat"
```

Resulting output:

```
Hello
ginger
cat
```

Write pseudocode for procedure `Square()`.

Parameter validation is not required.

Example of iterative solution:

```
PROCEDURE Square(Dim : INTEGER)
   DECLARE Count : INTEGER
   DECLARE ThisChar : CHAR
   DECLARE StringA, StringB : STRING
   CONSTANT FILLER = '*'

   StringA ← ""

   ThisChar ← NUM_TO_STR(Dim)

   FOR Count ← 1 TO Dim
      StringA ← StringA & ThisChar //build up first &
                                     last line
   NEXT Count

   StringB ← ThisChar
   FOR Count ← 1 TO Dim - 2
      StringB ← StringB & FILLER //build up
                                  intermediate line
   NEXT Count
   StringB ← StringB & ThisChar // add final digit

   OUTPUT StringA
   FOR Count ← 1 TO Dim - 2
      OUTPUT StringB
   NEXT Count

   IF Dim <> 1 THEN
      OUTPUT StringA
   ENDIF

ENDPROCEDURE
```

[6]

```
PROCEDURE Square(Dim : INTEGER)
   DECLARE Count : INTEGER

   CASE OF Dim
      1  : OUTPUT "1"
      2  : OUTPUT "22"
           OUTPUT "22"
      3  : OUTPUT "333"
           OUTPUT "3*3"
           OUTPUT "333"
      4  : OUTPUT "4444"
           FOR Count ← 1 TO 2
              OUTPUT "4**4"
           NEXT Count
           OUTPUT "4444"
      5  : OUTPUT "55555"
           FOR Count ← 1 TO 3
              OUTPUT "5***5"
           NEXT Count
           OUTPUT "55555"
      6  : OUTPUT "666666"
           FOR Count ← 1 TO 4
              OUTPUT "6****6"
           NEXT Count
           OUTPUT "666666"
      7  : OUTPUT "7777777"
           FOR Count ← 1 TO 5
              OUTPUT "7*****7"
           NEXT Count
           OUTPUT "7777777"
      8  : OUTPUT "88888888"
           FOR Count ← 1 TO 6
              OUTPUT "8******8"
           NEXT Count
           OUTPUT "88888888"
      9  : OUTPUT "999999999"
           FOR Count ← 1 TO 7
              OUTPUT "9*******9"
           NEXT Count
           OUTPUT "999999999"
   ENDCASE

ENDPROCEDURE
```

BLANK PAGE

**7** A computer system for a shop stores information about each customer. The items of information include name and address (both postal and email) together with payment details and order history. The system also stores the product categories they are interested in and how they would like to be contacted.

**(a)** The shop wants to add a program module that will generate emails to be sent to customers who may be interested in receiving details of new products.

**(i)** State **three** items of information that the new module would need. Justify your choice in each case.

Information ..

Justification ..

Information ..

Justification ..

Information ..

Justification ..

- Information: customer name
  Justification: to personalise / address the email
- Information: email address
  Justification: so that the email can be delivered correctly
- Information: product category preference
  Justification: to check that the customer would be interested in the product
- Information: contact preference
  Justification: to check that the customer will accept contact via email
- Information: order history
  Justification: to send details of product similar to that already bought // to identify frequent shoppers
- Information: new product information
  Justification: to include information about the new product in the email

[3]

**(ii)** Identify **two** items of customer information that would **not** be required by the new module. Justify your choice in each case.

Information ..

Justification ..

Information ..

Justification ..

- postal address
  Justification: sending an email, not a letter
- payment details
  Justification: Nothing being bought / sold at this stage
- order history
  Justification: Customer preference used to decide if new product is relevant

[2]

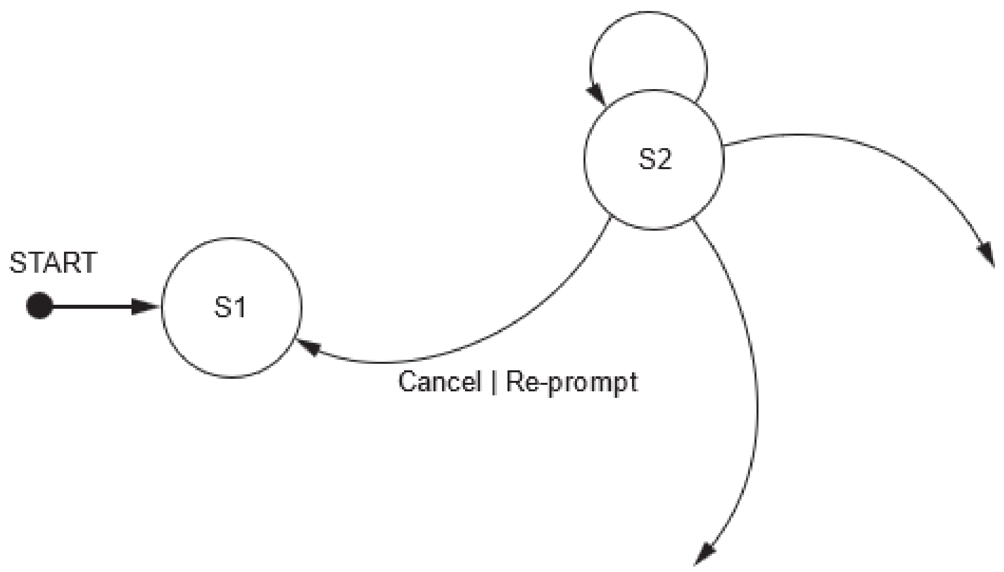
**(b)** The program includes a module to validate a Personal Identification Number (PIN). This is used when customers pay for goods using a bank card.

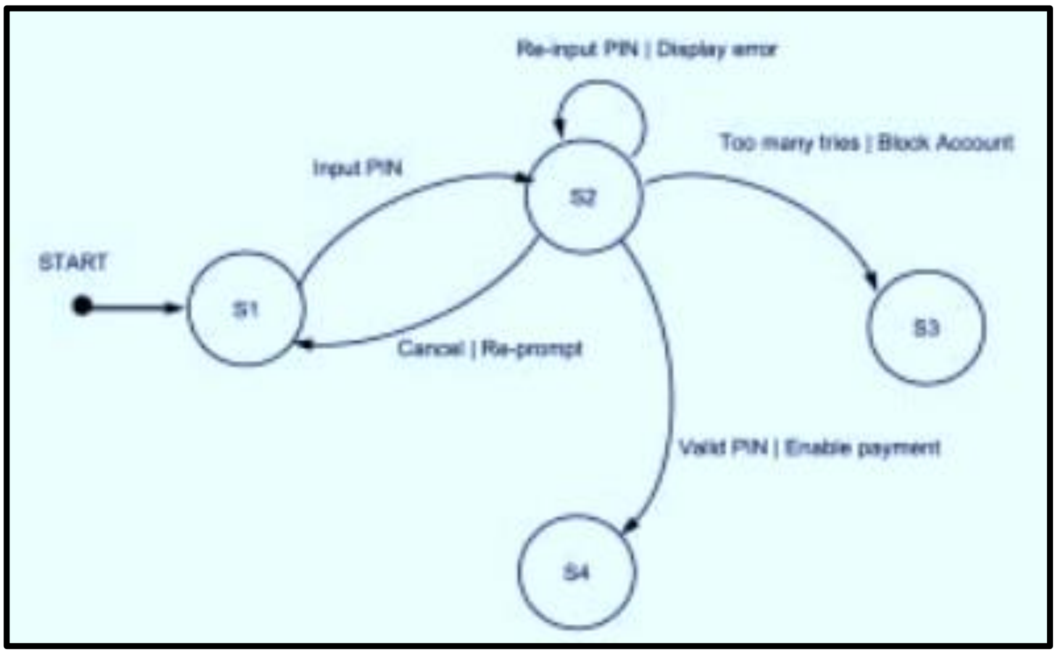
A state-transition diagram has been produced for this module.

The table show the inputs, outputs and states for this part of the program:

| **Current state** | **Input** | **Output** | **Next state** |
|---|---|---|---|
| S1 | Input PIN | | S2 |
| S2 | Re-input PIN | Display error | S2 |
| S2 | Cancel | Re-prompt | S1 |
| S2 | Valid PIN | Enable payment | S4 |
| S2 | Too many tries | Block Account | S3 |

Complete the state-transition diagram to represent the information given in the table.





Re-input PIN | Display error
Too many tries | Block Account
Input PIN
S2
START
S1
Cancel | Re-prompt
S3
Valid PIN | Enable payment
S4

[4]

**8** A computer shop assembles computers using items bought from several suppliers. A text file `Stock.txt` contains information about each item.

Information for each item is stored as a single line in the `Stock.txt` file in the format:

```
<ItemNum><SupplierCode><Description>
```

Valid item information is as follows:

| | **Format** | **Comment** |
|---|---|---|
| `ItemNum` | 4 numeric characters | unique number for each item in the range "0001" to "5999" inclusive |
| `SupplierCode` | 3 alphabetic characters | to identify the supplier of the item |
| `Description` | a string | a minimum of 12 characters |

The file is organised in ascending order of `ItemNum` and does **not** contain all possible values in the range.

A programmer has started to define program modules as follows:

| **Module** | **Description** |
|---|---|
| `OnlyAlpha()` (already written) | • called with a parameter of type string<br>• returns `TRUE` if the string contains only alphabetic characters, otherwise returns `FALSE` |
| `CheckInfo()` | • called with a parameter of type string representing a line of item information<br>• checks to see whether the item information in the string is valid<br>• returns `TRUE` if the item information is valid, otherwise returns `FALSE` |

**(a)** Write pseudocode for module `CheckInfo()`.

Module `OnlyAlpha()` should be used as part of your solution.

```
FUNCTION CheckInfo(NewLine: STRING) RETURNS BOOLEAN
  DECLARE ThisNum : STRING
  DECLARE Index : INTEGER

  IF LENGTH(NewLine) < 19 THEN
     RETURN FALSE
  ENDIF

  FOR Index ← 1 TO 4
    IF NOT IS_NUM(MID(NewLine, Index, 1)) THEN
       RETURN FALSE
    ENDIF
  NEXT Index

  ThisNum ← LEFT(Newline, 4)

  IF ThisNum < "0001" OR ThisNum > "5999" THEN
     RETURN FALSE
  ENDIF

  IF NOT OnlyAlpha(MID(Newline, 5, 3)) THEN
     RETURN FALSE
  ENDIF

  RETURN TRUE

ENDFUNCTION
```

[7]

**(b)** A new module is defined as follows:

| Module | Description |
|---|---|
| `AddItem()` | • called with a parameter of type string representing valid information for a new item that is not currently in the `Stock.txt` file<br>• creates a new file `NewStock.txt` from the contents of the file `Stock.txt` and adds the new item information at the appropriate place in the `NewStock.txt` file |

As a reminder, the file `Stock.txt` is organised in ascending order of `ItemNum` and does not contain all possible values in the range.

Write pseudocode for module `AddItem()`.

```
PROCEDURE AddItem(NewLine : STRING)
  DECLARE NewItemNum, ThisItemNum : STRING

  OPENFILE "Stock.txt" FOR READ
  OPENFILE "NewStock.txt" FOR WRITE
  NewItemNum ← LEFT(NewLine, 4)

  WHILE NOT EOF("Stock.txt")
     READFILE("Stock.txt", ThisLine)
     ThisItemNum ← LEFT(ThisLine, 4)
     IF ThisItemNum > NewItemNum THEN
        WRITEFILE("NewStock.txt", NewLine) // write New
                                              Line...
        NewItemNum ← "9999" // ...once only
     ENDIF
     WRITEFILE("NewStock.txt", ThisLine)
  ENDWHILE

  IF NewItemNum <> "9999" THEN
     WRITEFILE("NewStock.txt", NewLine) //New last line
                                          in the file
  ENDIF

  CLOSEFILE "Stock.txt"
  CLOSEFILE "NewStock.txt"
ENDPROCEDURE
```

Example of array-based solution:

```
PROCEDURE AddItem(NewLine : STRING)
   DECLARE ThisItemNum, ThisLine : STRING
   DECLARE Temp : ARRAY [1:5999] OF STRING
   DECLARE Index : INTEGER

   FOR Index ← 1 TO 5999
      Temp[Index] ← "" //Initialise array
   NEXT Index

   Index ← STR_TO_NUM(LEFT(NewLine, 4))
   Temp[Index] ← NewLine   //Add new line to array

   OPENFILE "Stock.txt" FOR READ

   WHILE NOT EOF("Stock.txt")
      READFILE("Stock.txt", ThisLine)
      Index ← STR_TO_NUM(LEFT(ThisLine, 4))
      Temp[Index] ← ThisLine //Add line from file to
array
   ENDWHILE
   CLOSEFILE "Stock.txt"

   OPENFILE "NewStock.txt" FOR WRITE
   FOR Index ← 1 TO 5999
      IF Temp[Index] <> "" THEN  //Write non-blank
element...
         WRITEFILE("NewStock.txt", Temp[Index]) //...to
new file
      ENDIF
   NEXT Index
   CLOSEFILE "NewStock.txt"

ENDPROCEDURE
```

[7]

**(c)** The program contains modules `SuppExists()` and `CheckSupplier()`. These have been written but contain errors. These modules are called from several places in the main program and testing of the main program (integration testing) has had to stop.

Identify a method that can be used to continue testing the main program before the errors in these modules have been corrected **and** describe how this would work.

Method: Method: Stub testing

Description:

- The modules `SuppExists()` and `CheckSupplier()` are replaced by dummy modules
- ...which return a known result / contain an output statement to show they have been called

[3]

**BLANK PAGE**