Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9608/21**

Paper 2 Fundamental Problem-solving and Programming Skills **May/June 2021**

MARK SCHEME

Maximum Mark: 75

Published

This mark scheme is published as an aid to teachers and candidates, to indicate the requirements of the examination. It shows the basis on which Examiners were instructed to award marks. It does not indicate the details of the discussions that took place at an Examiners' meeting before marking began, which would have considered the acceptability of alternative answers.

Mark schemes should be read in conjunction with the question paper and the Principal Examiner Report for Teachers.

Cambridge International will not enter into discussions about these mark schemes.

Cambridge International is publishing the mark schemes for the May/June 2021 series for most Cambridge IGCSE™, Cambridge International A and AS Level components and some Cambridge O Level components.

This document consists of **17** printed pages.

Generic Marking Principles

These general marking principles must be applied by all examiners when marking candidate answers. They should be applied alongside the specific content of the mark scheme or generic level descriptors for a question. Each question paper and mark scheme will also comply with these marking principles.

GENERIC MARKING PRINCIPLE 1:

Marks must be awarded in line with:

- the specific content of the mark scheme or the generic level descriptors for the question
- the specific skills defined in the mark scheme or in the generic level descriptors for the question
- the standard of response required by a candidate as exemplified by the standardisation scripts.

GENERIC MARKING PRINCIPLE 2:

Marks awarded are always **whole marks** (not half marks, or other fractions).

GENERIC MARKING PRINCIPLE 3:

Marks must be awarded **positively**:

- marks are awarded for correct/valid answers, as defined in the mark scheme. However, credit is given for valid answers which go beyond the scope of the syllabus and mark scheme, referring to your Team Leader as appropriate
- marks are awarded when candidates clearly demonstrate what they know and can do
- marks are not deducted for errors
- marks are not deducted for omissions
- answers should only be judged on the quality of spelling, punctuation and grammar when these features are specifically assessed by the question as indicated by the mark scheme. The meaning, however, should be unambiguous.

GENERIC MARKING PRINCIPLE 4:

Rules must be applied consistently, e.g. in situations where candidates have not followed instructions or in the application of generic level descriptors.

GENERIC MARKING PRINCIPLE 5:

Marks should be awarded using the full range of marks defined in the mark scheme for the question (however; the use of the full mark range may be limited according to the quality of the candidate responses seen).

GENERIC MARKING PRINCIPLE 6:

Marks awarded are based solely on the requirements as defined in the mark scheme. Marks should not be awarded with grade thresholds or grade descriptors in mind.

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>1(a)(i)</td><td>One mark for each part statement:

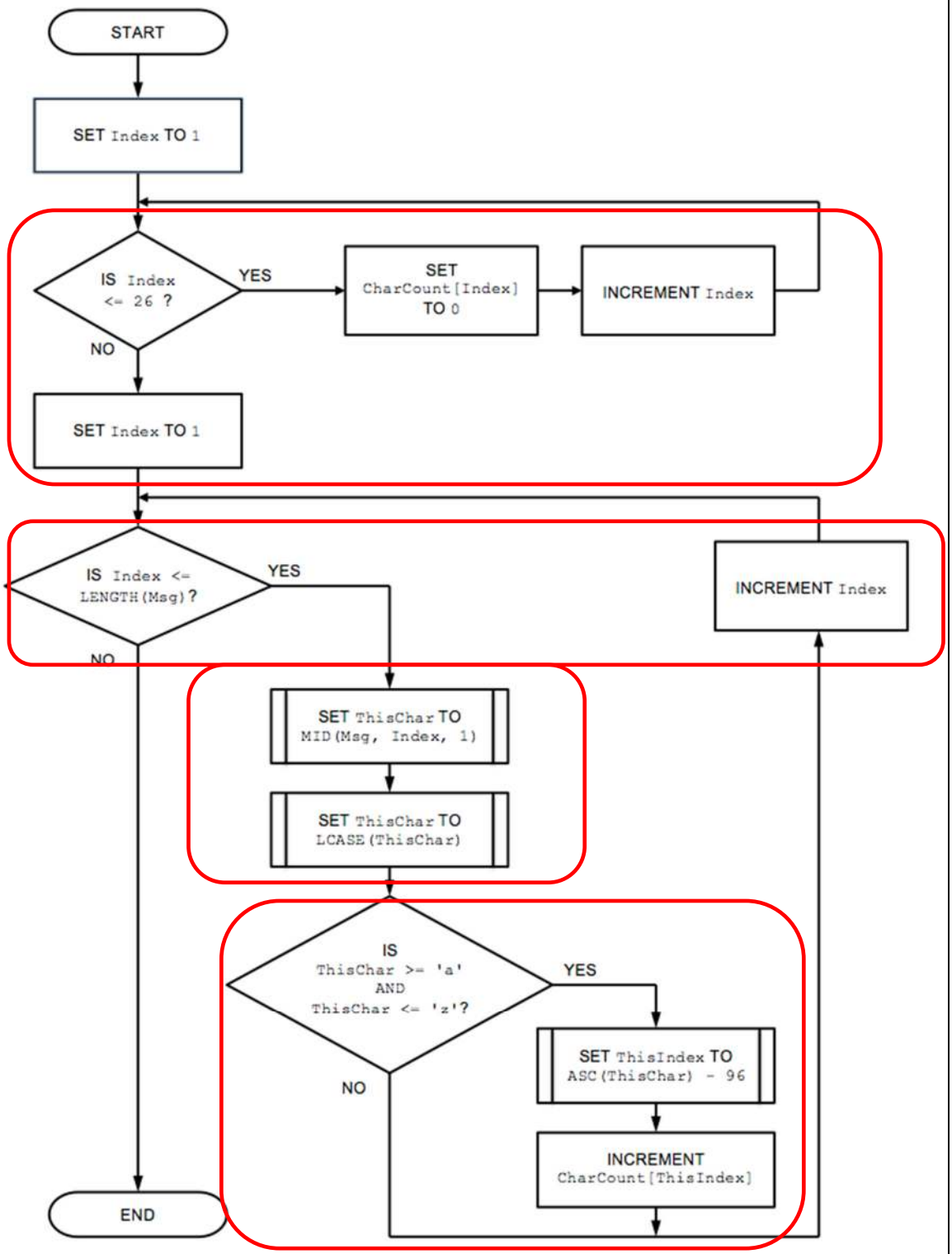
• Each character is assigned...
• ...a unique value
• …using 7 bits

Max 2 marks</td><td>2</td></tr>
<tr><td>1(a)(ii)</td><td>One mark for two correct, 2 marks for all correct

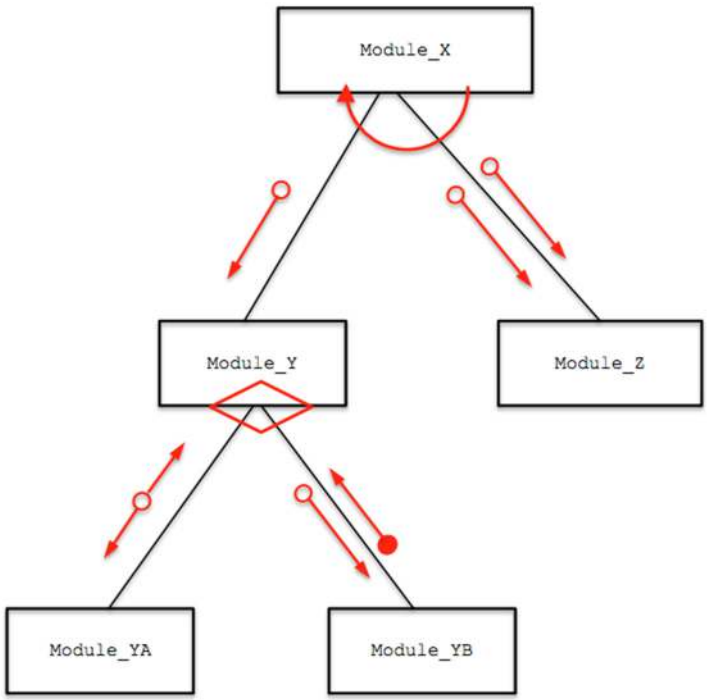
<table><tr><th>Memory location</th><th>ASCII character value</th></tr><tr><td>100</td><td>70</td></tr><tr><td>101</td><td>65</td></tr><tr><td>102</td><td>68</td></tr><tr><td>103</td><td>69</td></tr><tr><td>104</td><td>68</td></tr></table></td><td>2</td></tr>
<tr><td>1(b)(i)</td><td>One mark for both answers correct.
Exact terms only.

• Lower bound
• Upper bound</td><td>1</td></tr>
<tr><td>1(b)(ii)</td><td>index / subscript</td><td>1</td></tr>
<tr><td>1(c)</td><td>One mark for each error

<table><tr><th>Statement</th><th>Error</th></tr><tr><td><code>Code ← LEFT("Cat", 4)</code></td><td>Only 3 characters in string</td></tr><tr><td><code>Status ← MID("Aardvark", 0, 5)</code></td><td>Second parameter should start from 1</td></tr><tr><td><code>Size ← LENGTH("Password)</code></td><td>Missing closing quote / Opening quote should be removed</td></tr><tr><td><code>Stock[n] ← Stock[n+1]</code></td><td>NO ERROR / n may not be integer value / n out of bound</td></tr><tr><td><code>Result ← 3 OR 4</code></td><td>Not Boolean types</td></tr></table></td><td>5</td></tr>
</table>

Question	Answer	Marks
2	Mark as follows: First mark for START, Initialisation of `Index` **and** END Then one mark per area outlined, in correct place. At least one decision box label (YES / NO) must be present.  	**5**

Question	Answer	Marks
2(b)	One mark per point. Algorithm should mention: 1 Initialise variable to hold Max value 2 Loop through 26 elements of array 3 Test if element > Max ….and if so set new Max value 4 Method of checking for duplicates 5 Output a messge giving alphabetic char with largest count value - needs use of `CHR()` 6 Output a suitable message if largets count value is shared	**6**

Question	Answer	Marks
3(a)	 1 mark for each of: 1 Iteration arrow 2 Selection diamond 3 Both sets of parameters from `Module_X` 4 Parameter ByReferene to `Module_YA` 5 Parameter (ByValue) **and** return Boolean from `Module_YB`	**5**
3(b)(i)	One mark for each statement: • it is a function • because it returns a value	**2**

Question	Answer	Marks
3(b)(ii)	`PROCEDURE Module ZB (BYVALUE ParX : REAL, BYREF ParZ : STRING)` One mark for: • Procedure declaration • `ParX : REAL` and `ParZ : STRING` • `ByRef` for `ParZ` Condone missing `BYVALUE` for `ParX`	**3**

Question	Answer	Marks

4(a)	'Pseudocode' solution included here for development and clarification of mark scheme. Programming language solutions appear in the Appendix.
	
	`PROCEDURE ScanArray(SearchString STRING)`
	
	`  DECLARE Index, Total : INTEGER`
	`  DECLARE Error : BOOLEAN`
	
	`  Index ← 1`
	`  Total ← 0`
	`  Error ← FALSE`
	
	`  WHILE Index  <= 1000 AND Error <> TRUE`
	`  IF LENGTH(ThisArray[Index]) > 5`
	`  THEN`
	`  IF LEFT(ThisArray[Index], 4) = SearchString`
	`  THEN`
	`  Total ← Total + LENGTH(ThisArray[Index]) - 5`
	`  ENDIF`
	`  Index ← Index + 1`
	`  ELSE`
	`  Error ← TRUE`
	`  ENDIF`
	`  ENDWHILE`
	
	`  IF Index > 1`
	`  THEN`
	`  ArrayResult ← INT(Total / (Index - 1))`
	`  ENDIF`
	
	`ENDPROCEDURE`
	
	Mark as follows:
	1 Procedure header including parameter and end (where required)
	2 Local variable declarations and initialisation of `Index`, `Total` and `Error` but no local declaration of `ArrayResult`
	3 `WHILE / ENDWHILE` loop
	4 Nested `IF` statement comparing first four character of array element with `SearchString`
	5 Summation of `Total` using appropriate `LENGTH` function and subtracting 5
	6 Assignment to `ArrayResult` using appropriate `INT` function **AND** check for division by zero
		6

4(b)	One mark for each: • The IDE displays hints / choice of keywords / available identifiers • (Appropriate to) the current cursor position / insertion point	**2**

Question	Answer	Marks
4(c)(i)	One mark for Name, max 2 for Tasks (one per underlined term): Name: Design Tasks: To define the data structures and algorithms (of the solution) ALTERNATIVE Name: Analysis Tasks: Feasibility study // Problem definition / investigation // Requirement spec	**3**
4(c)(ii)	Coding / Implementation / Programming	**1**

Question	Answer	Marks

5(a)(i)	`PROCEDURE GuessNum()`
	`  DECLARE Count: INTEGER`
	`  DECLARE RndNumber : INTEGER`
	`  DECLARE MyGuess : INTEGER`
	
	`  RndNumber ← 1 + INT(RAND(20))`
	`  Count ← 1`
	
	`  REPEAT`
	`  OUTPUT "Input your guess"`
	`  INPUT MyGuess`
	`  IF MyGuess <> RndNumber`
	`  THEN`
	`  Count ← Count + 1`
	`  OUTPUT "Incorrect – try again"`
	`  ENDIF`
	`  UNTIL MyGuess = RndNumber`
	
	`  OUTPUT "You took ", Count, " guesses."`
	
	`ENDPROCEDURE`
	
	1 mark for each of the following:
	
	1 Use of `RAND()` to generate an integer between 1 and 20
	2 Conditional loop until random number is guessed
	3 Prompt and input a guess …**in a loop**
	4 Comparison and increment `Count` and 'Try again' output message…**in a loop**
	5 Final output message…**not in a loop**
		5

Question	Answer	Marks
5(a)(ii)	One mark per point. Check for: • Integer / number out of range <1 OR > 20 • Real number entered • Non-numeric value entered **Max 2 marks**	**2**
5(b)(i)	• Stub testing	**1**
5(b)(ii)	One mark for each: • A simplified version of `Status()` / a dummy function is written • that returns a typical / expected value.	**2**
5(b)(iii)	A compiler is used to translate / convert the source code / program / high-level language code into object code / machine code / an executable file	**1**

Question	Answer	Marks

6(a)	'Pseudocode' solution included here for development and clarification of the mark scheme.
	Programming language example solutions appear in the Appendix.
	
	`FUNCTION Check(Index : INTEGER) RETURNS BOOLEAN`
	
	`  IF LENGTH(StockID[Index]) <> 8 OR __`
	`  Description[Index]) = "" OR __`
	`  Quantity[Index] < 0`
	`  THEN`
	`  RETURN FALSE`
	`  ELSE`
	`  RETURN TRUE`
	`  ENDIF`
	
	`ENDFUNCTION`
	
	One mark for each of the following:
	
	1 Function heading and ending (where appropriate)
	2 Three comparisons...
	3 ... connected by logical `OR // AND` / correct nested IF
	4 `RETURN` value in both cases
		4

Question	Answer	Marks
6(b)		**8**

```
FUNCTION Backup() RETURNS BOOLEAN

  DECLARE Index : INTEGER
  DECLARE FileName, FileLine : STRING
  DECLARE AllOK : BOOLEAN

  CONSTANT ASTERISK = '*'
  AllOK ← TRUE

  FileName ← GetValidFileName()
  OPENFILE Filename FOR WRITE
  OPENFILE "ERRORLOG.TXT" FOR WRITE

  FOR Index ← 1 TO 10000
     IF StockID[Index]<> “”
       THEN
         FileLine ← StockID[Index] & ASTERISK
          FileLine ← FileLine & Description[Index] & ASTERISK
          FileLine ← FileLine & NUM_TO_STRING(Quantity[Index]) &
                                                ASTERISK
          FileLine ← FileLine & NUM_TO_STRING(Cost[Index])
          WRITEFILE FileName, FileLine

        //now check for sensible data
         IF Check(Index) <> TRUE
           THEN
             WRITEFILE, "ERRORLOG.TXT", FileLine
             AllOK ← FALSE
         ENDIF
    ENDIF
  ENDFOR

  CLOSEFILE FileName
  CLOSEFILE "ERRORLOG.TXT"
  RETURN AllOK
ENDFUNCTION
```

1 mark for each of the following:

1 Declare local variable for backup filename and index
2 Call to function `GetValidFileName()`
3 `OPEN` and `CLOSE` **both** files
4 Loop all 10 000 elements
5 Form FileLine using at least one correct array index expression and asterisk
6 Use of `NUM_TO_STRING()` to convert at least one of `QUANTITY` or `COST`
7 Write line to backup file
8 Call `Check()` to determine whether values are valid and if not, write to `ERRORLOG.TXT`
9 Return AllOK

Max 8 marks from possible 10 mark points

Question	Answer	Marks
6(c)	'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix.	**8**

```
PROCEDURE Unpack(Index : INTEGER, FileLine : STRING)

  DECLARE Pointer : INTEGER
  DECLARE NextChar : CHAR
  DECLARE TempString : STRING
  CONSTANT ASTERISK = '*'

  StockID[Index] ← LEFT(FileLine, 8) // the only fixed length
                                        field

  Pointer ← 10 // Point to start of Description (skip the '*')
  NextChar ←  MID(FileLine, Pointer, 1)
  TempString ← ""
  WHILE NextChar <> ASTERISK
     TempString ← TempString & NextChar
     Pointer ← Pointer + 1
     NextChar ←  MID(FileLine, Pointer, 1)
  ENDWHILE

  Description[Index] ← TempString
  Pointer ← Pointer + 1
  NextChar ←  MID(FileLine, Pointer, 1)

  TempString ← ""
  WHILE NextChar <> ASTERISK
     TempString ← TempString & NextChar
     Pointer ← Pointer + 1
     NextChar ←  MID(FileLine, Pointer, 1)
  ENDWHILE

  Quantity[Index] ← STRING_TO_NUM(TempString)
  TempString ← RIGHT(FileLine, LENGTH(FileLine) - Pointer)
  Cost[Index] ← STRING_TO_NUM(TempString)

ENDFUNCTION
```

1 mark for each of the following:

1 Procedure heading with parameters
2 Extract first 8 chars of `FileLine`
3 Assign to `StockID`
4 Search for asterisk for place separator
5 Extract Description string and assign to `Description` array
6 Extract Quantity string, and assign to `Quantity` array...
7 Extract Cost string and assign to `Cost` array
8 Type conversion for Cost and Quantity

*** End of Mark Scheme – example program code solutions follow ***

Program Code Example Solutions

Q4 (a): Visual Basic

```
Sub ScanArray(SearchString As String)

  Dim Index, Total As Integer
  Dim Error As Boolean

  Index = 1
  Total = 0
  Error = FALSE

  While Index  <= 1000 And Error <> TRUE
     If Len(ThisArray(Index)) > 5 Then
        If Left(ThisArray(Index), 4) = SearchString Then
           Total = Total + Len(ThisArray(Index)) - 5
        End If
        Index = Index + 1
     Else
        Error = TRUE
     End If
  End While

  If Index > 1  Then
     ArrayResult = Int(Total / (Index - 1))
  End If

End Sub
```

Q4 (a): Pascal

```
procedure ScanArray(SearchString : string);

var
  Index, Total : integer;
  Error : boolean;

begin
  Index := 1;
  Total := 0;
  Error := FALSE;

  while Index  <= 1000 And Error <> TRUE do
  begin
     if Length(ThisArray[Index]) > 5 then
     begin
        if LeftStr(ThisArray[Index], 4) = SearchString then
           Total := Total + Length(ThisArray[Index]) - 5;

        Index := Index + 1;
     else
        Error := TRUE;
     end;
  end;

  if Index > 1 then
     ArrayResult := int(Total / (Index - 1));

end;
```

Q4(a): Python

```
def ScanArray(SearchString):

  ## Index, Total As Integer
  ## Error As Boolean

  Index = 1
  Total = 0
  Error = FALSE

  while Index  <= 1000 and Error <> TRUE:
     if len(ThisArray[Index]) > 5:
        ThisElement = ThisArray[Index]
        if ThisElement[:4] == SearchString:
           Total = Total + len(ThisArray[Index]) - 5
        Index = Index + 1
     else:
        Error = TRUE


  if Index > 1:
     ArrayResult = int(Total / (Index - 1))
```

Q6 (a): Visual Basic

```
Function Check(Index As Integer) As Boolean

  If Len(StockID(Index)) <> 8 Or _
     Description(Index)) = "" Or _
     Quantity(Index) < 0 Then
        Return FALSE
     Else
        Return TRUE
  End If

End Function
```

Q6(a): Pascal

```
function Check(Index : Integer) : boolean;

begin

  if Length(StockID[Index]) <> 8 Or
     Description[Index]) = "" Or
     Quantity[Index] < 0 then
       Check := FALSE // result := FALSE
  else
       Check := TRUE // result := TRUE
  end;

end
```

Q6(a): Python

```
def Check(Index):

  if len(StockID[Index]) <> 8 or \
     Description[Index]) == "" or \
     Quantity[Index] < 0:
        return FALSE
   else:
        return TRUE
```

Q6(c): Visual Basic

```
Sub Unpack(Index As Integer, FileLine As String)

  Dim Pointer As Integer
  Dim NextChar As Char
  Dim TempString As String
  Const ASTERISK = '*'

  StockID(Index) = LEFT(FileLine, 8)

  Pointer = 10       'point to start of Description (skip the '*')
  NextChar =  Mid(FileLine, Pointer, 1)
  TempString = ""
  While NextChar <> ASTERISK
     TempString = TempString & NextChar
     Pointer = Pointer + 1
     NextChar =  Mid(FileLine, Pointer, 1)
  End While

  Description(Index) = TempString
  Pointer = Pointer + 1
  NextChar =  Mid(FileLine, Pointer, 1)

  TempString = ""
  While NextChar <> ASTERISK
     TempString = TempString & NextChar
     Pointer = Pointer + 1
     NextChar =  Mid(FileLine, Pointer, 1)
  End While

  Quantity(Index) = CInt(TempString)
  TempString = Right(FileLine, Len(FileLine) - Pointer)
  Cost(Index) = CDec(TempString)

End Sub
```

Q6(c): Pascal

```
procedure Unpack(Index : Integer, FileLine : String);

var
  Pointer : integer;
  NextChar : char;
  TempString : string;

const
  ASTERISK = '*';

begin
  StockID[Index] := LeftStr(FileLine, 8);

  Pointer := 10;       //point to start of Description (skip the '*')
  NextChar :=  MidStr(FileLine, Pointer, 1);
  TempString := "";
  while NextChar <> ASTERISK do
```

```
  begin
     TempString := TempString & NextChar;
     Pointer := Pointer + 1;
     NextChar :=  MidStr(FileLine, Pointer, 1);
   end;

  Description[Index] := TempString;
  Pointer := Pointer + 1;
  NextChar :=  MidStr(FileLine, Pointer, 1);

  TempString := "";
  while NextChar <> ASTERISK do
  begin
     TempString := TempString & NextChar;
     Pointer := Pointer + 1;
     NextChar :=  MidStr(FileLine, Pointer, 1);
  end;

  Quantity[Index] := StrToInt(TempString);
  TempString := RightStr(FileLine, Length(FileLine) - Pointer);
  Cost[Index] := StrToFloat(TempString);

end;
```

Q6(c): Python

```
def Unpack(Index, FileLine):

  ## Pointer As Integer
  ## NextChar As Char
  ## TempString As String
  ASTERISK = '*'

  StockID[Index] = FileLine[:8] #characters 0 to 7

  Pointer = 9       #point to start of Description (skip the '*')
  NextChar =  FileLine[Pointer]
  TempString = ""

  while NextChar <> ASTERISK:
     TempString = TempString + NextChar
     Pointer = Pointer + 1
     NextChar =  FileLine[Pointer]

  Description[Index] = TempString
  Pointer = Pointer + 1
  NextChar =  FileLine[Pointer]

  TempString = ""

  while NextChar <> ASTERISK:
     TempString = TempString + NextChar
     Pointer = Pointer + 1
     NextChar =  FileLine[Pointer]

  Quantity[Index] = int(TempString)
  TempString = FileLine[Len(FileLine) - Pointer - 1:)
```

```
  Cost[Index] = float(TempString)
```

Alternative

```
def Unpack(Index, FileLine):
   ## TempString As String

   StockID[Index] = FileLine[:8]
   TempString = FileLine[8:]   // remove first 8 characters
   Description[Index], Quantity[Index], Cost[Index] =
(TempString.split('*'))
```