AQA

AS
COMPUTER SCIENCE
7516/1

Paper 1

Mark scheme

June 2024

Version: 1.0 Final

246A7516/1/MS

Mark schemes are prepared by the Lead Assessment Writer and considered, together with the relevant questions, by a panel of subject teachers. This mark scheme includes any amendments made at the standardisation events which all associates participate in and is the scheme which was used by them in this examination. The standardisation process ensures that the mark scheme covers the students' responses to questions and that every associate understands and applies it in the same correct way. As preparation for standardisation each associate analyses a number of students' scripts. Alternative answers not already covered by the mark scheme are discussed and legislated for. If, after the standardisation process, associates encounter unusual answers which have not been raised they are required to refer these to the Lead Examiner.

It must be stressed that a mark scheme is a working document, in many cases further developed and expanded on the basis of students' reactions to a particular paper. Assumptions about future mark schemes on the basis of one year's document should be avoided; whilst the guiding principles of assessment remain constant, details will change, depending on the content of a particular examination paper.

No student should be disadvantaged on the basis of their gender identity and/or how they refer to the gender identity of others in their exam responses.

A consistent use of 'they/them' as a singular and pronouns beyond 'she/her' or 'he/him' will be credited in exam responses in line with existing mark scheme criteria.

Further copies of this mark scheme are available from aqa.org.uk

AS Computer Science

Paper 1 (7516/1) – applicable to all programming languages A, B, C, D and E

June 2024

The following annotation is used in the mark scheme:

; - means a single mark
// - means alternative response
/ - means an alternative word or sub-phrase
A. - means acceptable creditworthy answer
R. - means reject answer as not creditworthy
NE. - means not enough
I. - means ignore
DPT. - means 'Don't penalise twice'. In some questions a specific error made by a candidate, if repeated, could result in the loss of more than one mark. The **DPT** label indicates that this mistake should only result in a candidate losing one mark, on the first occasion that the error is made. Provided that the answer remains understandable, subsequent marks should be awarded as if the error was not being repeated.

Level of response marking instructions

Level of response mark schemes are broken down into levels, each of which has a descriptor. The descriptor for the level shows the average performance for the level. There are marks in each level.

Before you apply the mark scheme to a student's answer read through the answer and annotate it (as instructed) to show the qualities that are being looked for. You can then apply the mark scheme.

Step 1 Determine a level

Start at the lowest level of the mark scheme and use it as a ladder to see whether the answer meets the descriptor for that level. The descriptor for the level indicates the different qualities that might be seen in the student's answer for that level. If it meets the lowest level then go to the next one and decide if it meets this level, and so on, until you have a match between the level descriptor and the answer. With practice and familiarity you will find that for better answers you will be able to quickly skip through the lower levels of the mark scheme.

When assigning a level you should look at the overall quality of the answer and not look to pick holes in small and specific parts of the answer where the student has not performed quite as well as the rest. If the answer covers different aspects of different levels of the mark scheme you should use a best fit approach for defining the level and then use the variability of the response to help decide the mark within the level, ie if the response is predominantly level 3 with a small amount of level 4 material it would be placed in level 3 but be awarded a mark near the top of the level because of the level 4 content.

Step 2 Determine a mark

Once you have assigned a level you need to decide on the mark. The descriptors on how to allocate marks can help with this. The exemplar materials used during standardisation will help. There will be an answer in the standardising materials which will correspond with each level of the mark scheme. This answer will have been awarded a mark by the Lead Examiner. You can compare the student's answer with the example to determine if it is the same standard, better or worse than the example. You can then use this to allocate a mark for the answer based on the Lead Examiner's mark on the example.

You may well need to read back through the answer as you apply the mark scheme to clarify points and assure yourself that the level and the mark are appropriate.

Indicative content in the mark scheme is provided as a guide for examiners. It is not intended to be exhaustive and you must credit other valid points. Students do not have to cover all of the points mentioned in the Indicative content to reach the highest level of the mark scheme.

An answer which contains nothing of relevance to the question must be awarded no marks.

Examiners are required to assign each of the candidate's responses to the most appropriate level according to **its overall quality**, and then allocate a single mark within the level. When deciding upon a mark in a level, examiners should bear in mind the relative weightings of the assessment objectives.

eg

In question **11.1**, the marks available are as follows:

AO1 (knowledge)	1 mark
AO2 (analyse)	1 mark

In question **17.1**, the marks available for the AO3 elements are as follows:

AO3 (design)	2 marks
AO3 (programming)	6 marks

In question **18.1**, the marks available for the AO3 elements are as follows:

AO3 (design)	3 marks
AO3 (programming)	9 marks

Where a candidate's answer only reflects one element of the AO, the maximum mark they can receive will be restricted accordingly.

Section A

Qu		Marks	
01		**5 marks for AO2 (application)** **1 mark** for each correct set of values in the correct sequence (boxed in red); **I.** missing quotes **I.** duplicated values in a column If, after marking according to the boxed sections, fewer than 3 marks are awarded, **1 mark** can be awarded for each of the following, up to a maximum total of 3: Column C completely correct; Column R completely correct; Columns J, X, D1, D2 and S **all** completely correct; **Max 4** if any errors	5

S1	S2	C	R	J	X	D1	D2	S
"011101"	"001100"	"0"	""					
				0	5	"1"	"0"	"1"
			"1"	1	4	"0"	"0"	"0"
		"0"	"01"	2	3	"1"	"1"	"0"
		"1"	"001"	3	2	"1"	"1"	"1"
		"1"	"1001"	4	1	"1"	"0"	"0"
		"1"	"01001"	5	0	"0"	"0"	"1"
		"0"	"101001"					

OUTPUT:	"101001"

Qu		Marks	
02	1	**2 marks for AO1 (understanding)** Global variables accessible to all parts of the program // local variables accessible only in the program block/subroutine in which it was declared; Local variables declared in subroutine // global variable declared in main program block/outside subroutines; Local variables only use memory/exist while the program block/subroutine (in which they are declared) is executing; **Max 2**	**2**
02	2	**2 marks for AO1 (understanding)** Using local variables makes subroutine self-contained; Using local variables aids modularisation; Local variables use less memory // memory allocated to local variables can be reused when subroutine not in use; Variable names can be reused (in other parts of the program); Code is easier to re-use in other programs; **A.** Prevents unintended side-effects **A.** Easier debugging/maintenance/testing **Max 2**	**2**
03		**3 marks for AO1 (understanding)** Can get an overview of (the structure of) the program // code is easier to understand; Can break problem down into sub-tasks; Can re-use subroutines/modules; **A.** less duplication of code Can distribute (implementation of) subroutines/modules among team; Can test subroutines/modules independently // quicker/easier to debug/maintain // easier to locate errors; **Max 3**	**3**

Qu		Marks	
04	1	**8 marks for AO3 (programming)** **Mark as follows:** 1. Correct variable declarations for `Number1`, `Number2`, `Number`, `Count` and initialisation; **Note to examiners:** If a language allows variables to be used without explicit declaration, (eg Python), then this mark should be awarded if the correct variables exist in the program code and the first value they are assigned is of the correct data type. 2. Correct prompts `"Enter an integer: "` and `Number1` assigned integer value entered by user and `"Enter another integer: "` and `Number2` assigned integer value entered by user; 3. Correct `IF THEN ELSE` statement syntax allowed by the programming language and correct condition; 4. Correct assignments to `Number` in `THEN` and `ELSE` part; 5. Loop iterates correct number of times; 6. Correct condition to output X; 7. Correct condition to output V; 8. Correct output within loop without line feed; **I.** case and minor typos **Max 7** if code does not function correctly	**8**

04	2	**Mark is for AO3 (evaluate)**
		
		****** SCREEN CAPTURE ******
		Must match code from **04.1**.
		Code for **04.1** must be sensible.
		
		Screen capture showing:
		
		`Enter an integer: 4`
		`Enter another integer: 99`
		`////V////X////V////X////`
			1

Section B

Qu		Marks	
05	**1**	**Mark is for AO1 (knowledge)** `real/float/single/double/decimal`;	**1**
05	**2**	**Mark is for AO2 (analyse)** `AverageWaitingTime/AverageQLength`; **R.** if any additional code **R.** if spelt incorrectly **I.** case and spacing	**1**
06	**1**	**Mark is for AO1 (knowledge)** `String`; **A.** `str` **A.** an array/list of characters **I.** case	**1**
06	**2**	**Mark is for AO2 (analyse)** `Answer` / `DataString` / `ThisBuyerID`; **A.** `BLANK` (Python only) **A.** `input`, `output`, `format` (Java only) **A.** `BuyerID` **R.** `self.BuyerID` **R.** if any additional code **R.** if spelt incorrectly **I.** case and spacing **Max 1**	**1**
07		**Mark is for AO2 (analyse)** `FindFreeTill/ChangeSettings`; **R.** if any additional code **R.** if spelt incorrectly **I.** case and spacing	**1**

Qu		Marks	
08		**Mark is for AO2 (analyse)** Otherwise there could be a serving time of 0; Should round up to the (whole) number of time units needed; **A.** by example such as: if speed = 3, items = 8 then this will take more than 2 time units. **Max 1**	**1**
09		**2 marks for AO2 (analyse)** To finish serving all (waiting) buyers // there might be buyers left who haven't been served; … and include them in the statistics; **A**. example of any statistic listed in the `simulation statistics` output.	**2**
10	1	**Mark is for AO2 (analyse)** `BuyerQ`; **R.** if any additional code **R.** if spelt incorrectly **I.** case and spacing	**1**
10	2	**Mark is for AO2 (analyse)** `Stats` / `Tills` / `Data`; **A**. `buyerInfo` (Java only) **R.** if any additional code **R.** if spelt incorrectly **I.** case and spacing **Max 1**	**1**

Qu		Marks	
11	1	**Procedural composition** **1 mark for AO1 (knowledge):** Combining subroutines to form compound subroutines // a subroutine that calls other subroutines; **A.** another subroutine **1 mark for AO2 (analyse):** Several subroutines are combined to form the compound subroutine `Serving` // `FindFreeTill`, `ServeBuyer`, `UpdateStats`, `CalculateServingTime`, `IncrementTimeWaiting`, `UpdateTills`, `OutputTillAndQueueStates` are combined into one subroutine `Serving` (**Note**: 2 or more subroutines **must** be named); Several subroutines are combined to form the compound subroutine `QueueSimulator` // `ResetDataStrucutures`, `ChangeSettings`, `ReadInSimulationData`, `OutputHeading`, `BuyerArrives`, `Serving`, `TillsBusy`, `OutputTillAndQueueStates`, `OutputStats`, `UpdateTills` are combined into one subroutine `QueueSimulator` (**Note**: 2 or more subroutines **must** be named); **Max 1** **Data composition** **1 mark for AO1 (knowledge):** Combining data objects to form compound data; **1 mark for AO2 (analyse):** Several records of type `Q_Node` are combined to form the compound data structure `BuyerQ` // (three) values/data items are combined to make a `Q_Node` // `BuyerID`, `WaitingTime` and `ItemsInBasket` are combined to make a `Q_Node` // arrays;	**2**

Qu		Marks	
11	2	**2 marks for AO2 (analyse)** **<u>Procedural composition</u>** Some groups of subroutines need to be called in two places/more than one place // the group of subroutines in `Serving` need to be called during the main simulation time and also after buyers stop arriving; Less code is required if only one compound subroutine needs to be called; It improves understanding of code; **<u>Data composition</u>** Array elements are easier to address than individual variables; The grouped data items/record can be manipulated as one unit; **Max 2** Award marks for **either** procedural composition **or** data composition **or** both.	**2**
12		**2 marks for AO2 (analyse)** As index into (elements of) the `Tills` data structure // to specify which array element should be used; Don't need to remember which element of the data structure is used when referring to it // code is easier to understand // the data structure can be changed / reordered and just by changing this value the program will still work;	**2**
13	1	**2 marks for AO2 (analyse)** All queue records (apart from the first one); are moved one location (forward);	**2**
13	2	**Mark is for AO2 (analyse)** Buyers moving forward in/towards the front of/up the queue;	**1**
14	1	**Mark is for AO2 (analyse)** Divides `Stats[TOTAL_Q]` by `Stats[TOTAL_Q_OCCURENCE]`;	**1**

Qu		Marks	
14	2	**2 marks for AO2 (analyse)** `Stats[TOTAL_Q]` has the length of the queue added onto it (in each time unit that there is a non-empty queue); `Stats[TOTAL_Q_OCCURRENCE]` is incremented in each time unit that there is a (non-empty) queue; **DPT** within 14.2 reference to an index such as `TOTAL_Q` rather than the data to which it points, such as `Stats[TOTAL_Q]`	**2**
15		**2 marks for AO3 (design)** Need a 2D data structure/list of lists to store the queues // add a field to `Q_Node` to store which till the buyer is queuing for // add a queue/array/list to each `Till`/element of the `Tills` data structure; **R.** Use one array per `Till` Need code to allocate buyers to different queues // the code that moves someone out of the queue to be served would need changing // the code that moves everyone in the queue up would need changing // the code that displays the contents of the queue would need changing;	**2**

Section C

Qu		Marks	
16	**1**	**7 marks for AO3 (programming)** **Marking guidance:** **Evidence of AO3 programming – 7 marks:** Evidence of programming to look for in response: 1. Constant declared and used as index (any index between 6 and 9) for `Stats`; 2. If queue length not less than 5 … // if queue length equals 5 …; 3. … Increment count of shuns in `Stats` data structure; **R**. if not within a selection structure. 4. … Output buyer number and message; **R**. if not within a selection structure. 5. … If number of tills is less than `MAX_TILLS` …; **R**. if not within a selection structure. 6. …. Increment `NoOfTills`; **R**. if not within a nested selection structure. 7. In `OutputStats` output number of total shuns with suitable message; **Max 6** if any errors	**7**

16	**2**	**Mark is for AO3 (evaluate)**
		
		****** SCREEN CAPTURE ******
		Must match code from **16.1**, including prompts on screen capture matching those in code.
		Code for **16.1** must be sensible.
		
		Screen capture showing:
		
		`The simulation statistics are:`
		`==========`
		`The maximum queue length was: 5 buyers`
		`The maximum waiting time was: 12 time units`
		`33 buyers arrived during 50 time units`
		`The average waiting time was: 4.2 time units`
		`The average queue length was: 3.26 buyers`
		`4 buyers did not need to queue`
		`4 buyers turned away because the queue was too long`
			1

Qu		Marks	
17	1	**2 marks for AO3 (design) and 6 marks for AO3 (programming)** **Marking guidance:** **Evidence of AO3 design – 2 marks:** Evidence of design to look for in response: 1. Identify the need for a loop or equivalent to initialise the till speeds/re-use loop in `ResetDataStructures`; 2. Recognise need to use `Tills` data structure to calculate serving time in `CalculateServingTime`; **Note: AO3** (design) points are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works. **Evidence of AO3 programming – 6 marks:** Evidence of programming to look for in response: 3. Correctly calculate and store the default till speeds; 4. Correctly change the size of each element in `Tills`; 5. Add `Tills` to parameter list of `ChangeSettings` definition and call // add `Tills` to return value (Python) and assign in call to `ChangeSettings` in `QueueSimulator`; 6. Correctly set up loop to set the speed for each till; 7. Output suitable message to user including till number and default till speed; 8. Store till speed entered by user in correct element of `Tills`; **Max 7** if any errors	**8**

Qu		Marks	
17	2	**Mark is for AO3 (evaluate)** ****** SCREEN CAPTURE ****** Must match code from **17.1**, including prompts on screen capture matching those in code. Code for **17.1** must be sensible. Screen capture showing: `---------- 5 B5(2) B5 3 3 1 0 6 4 2 1 5 5 3 5 1 2 ** Start of queue ** *** End of queue *** ----------`	**1**

<table>
<tr><th>Qu</th><th></th><th>Marks</th><th></th></tr>
<tr><td>18</td><td>1</td><td>

3 marks for AO3 (design) and 9 marks for AO3 (programming)

<table>
<tr><th>Level</th><th>Description</th><th>Mark Range</th></tr>
<tr><td>3</td><td>A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution. All of the appropriate design decisions have been taken.</td><td>9–12</td></tr>
<tr><td>2</td><td>There is evidence that a line of reasoning has been partially followed. There is evidence of some appropriate design work. This is a partially working programmed solution.</td><td>5–8</td></tr>
<tr><td>1</td><td>An attempt has been made to amend the subroutine <code>Serving</code> and/or <code>OutputTillAndQueueStates</code>. Some appropriate programming statements have been written. There is little evidence to suggest that a line of reasoning has been followed or that the solution has been designed. The statements written may or may not be syntactically correct and the subroutines will have very little or none of the extra required functionality. It is unlikely that any of the key design elements of the task have been recognised.</td><td>1–4</td></tr>
</table>

Marking guidance:

Evidence of AO3 design – 3 marks:

Evidence of design to look for in response:

1. Attempt to test for conditions to be served at express till (in `Serving`).
2. Recognise the need for a loop to find next buyer with < 10 items (in `ServeBuyerExpress`).
3. Recognise the need to move buyer records in `BuyerQ`.

Note: AO3 (design) points are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works.

Evidence of AO3 programming – 9 marks:

Evidence of programming to look for in response:

4. Correct parameters and return values for `ServeBuyerExpress`.
5. Correct conditions for finding a buyer eligible for express till within loop. **R**. if multiple buyers would be found in a single method call.
6. Extract buyer data only if a buyer with less than 10 items has been found.
7. Correctly move buyer records in `BuyerQ`.
8. Output buyer ID.
9. Call `UpdateStats`.
10. Call `CalculateServingTime` with till 0 parameter.
11. Call `ServeBuyerExpress` under correct conditions (QLength > 0 and till 0 free).
12. Till 0 stats included in `OutputTillAndQueueStates`.

Max 11 if code does not function correctly

</td><td>12</td></tr>
</table>

Qu		Marks	

18	2	**Mark is for AO3 (evaluate)**
		
		****** SCREEN CAPTURE ******
		Must match code from **18.1**, including prompts on screen capture matching those in code.
		Code for **18.1** must be sensible.
		
		If Question 17 has been implemented the output should be:
		
		`----------`
		`8 B7(7)`
		`B7 0 2`
		`0 5 4 1`
		`1 2 7 3`
		`2 6 3 0`
		`** Start of queue **`
		`*** End of queue ***`
		
		If Question 17 has **not** been implemented the output should be:
		
		`----------`
		`8 B7(7)`
		`B7 0 3`
		`0 4 5 2`
		`1 1 8 1`
		`2 4 5 0`
		`** Start of queue **`
		`B6 3 25`
		`*** End of queue ***`
		`----------`
			1

VB.Net

04 **1** **8**

```
Sub Main()
   Dim Number1, Number2, Number, Count As Integer
   Console.Write("Enter an integer: ")
   Number1 = Console.ReadLine()  'MP1
   Console.Write("Enter another integer: ")
   Number2 = Console.ReadLine()   'MP2
   If Number1 > Number2 Then  'MP3
      Number = Number1 \ Number2
   Else
      Number = Number2 \ Number1  'MP4
   End If
   Count = 0
   While Count <> Number  'MP5
      Count = Count + 1
      If (Count Mod 10) = 0 Then   'MP6
         Console.Write("X")
      Else
         If (Count Mod 5) = 0 Then  'MP7
               Console.Write("V")
         Else
            Console.Write("/")   'MP8
         End If
      End If
   End While
   Console.ReadLine()
End Sub
```

16 **1** **7**

```
    ' indices for Stats data structure
     Const MAX_Q_LENGTH As Integer = 0
     Const MAX_WAIT As Integer = 1
     Const TOTAL_WAIT As Integer = 2
     Const TOTAL_Q As Integer = 3
     Const TOTAL_Q_OCCURRENCE As Integer = 4
     Const TOTAL_NO_WAIT As Integer = 5
     Const TOTAL_SHUNS As Integer = 6    'Q16 MP1

  Sub BuyerArrives(Data(,) As Integer, BuyerQ() As Q_Node, ByRef QLength As
Integer, BuyerNumber As Integer, ByRef NoOfTills As Integer, Stats() As
Integer)
     If QLength < 5 Then  'Q16 MP2
       Console.WriteLine($"  B{BuyerNumber}({Data(BuyerNumber, ITEMS)})")
       BuyerJoinsQ(Data, BuyerQ, QLength, BuyerNumber)
     Else
       Stats(TOTAL_SHUNS) += 1         'Q16 MP3
       If NoOfTills < MAX_TILLS Then  'Q16 MP5
         NoOfTills += 1               'Q16 MP6
       End If
       Console.WriteLine($"  B{BuyerNumber} has shunned the queue.") 'Q16 MP4
     End If
  End Sub

  Sub OutputStats(Stats() As Integer, BuyerNumber As Integer, SimulationTime
As Integer)
     Console.WriteLine("The simulation statistics are:")
     Console.WriteLine("==============================")
     Console.WriteLine($"The maximum queue length was: {Stats(MAX_Q_LENGTH)}
buyers")
     Console.WriteLine($"The maximum waiting time was: {Stats(MAX_WAIT)} time
units")
     Console.WriteLine($"{BuyerNumber} buyers arrived during {SimulationTime}
time units")
```

		Console.WriteLine($"The average waiting time was: {Math.Round(Stats(TOTAL_WAIT) / BuyerNumber, 1)} time units")
		If Stats(TOTAL_Q_OCCURRENCE) > 0 Then
		Console.WriteLine($"The average queue length was: {Math.Round(Stats(TOTAL_Q) / Stats(TOTAL_Q_OCCURRENCE), 2)} buyers")
		End If
		Console.WriteLine($"{Stats(TOTAL_NO_WAIT)} buyers did not need to queue")
		Console.WriteLine($"{Stats(TOTAL_SHUNS)} buyers shunned the queue") 'Q16 MP7
		End Sub	

17	**1**	Sub ResetDataStructures(Stats() As Integer, Tills(,) As Integer, BuyerQ() As Q_Node)
		For i As Integer = 0 To 9
		Stats(i) = 0
		Next
		For Count As Integer = 0 To MAX_TILLS
		For i As Integer = 0 To 2
		Tills(Count, i) = 0
		Next
		Tills(Count, 3) = 7 – Count ' Q17 MP1 MP3
		Next
		For i As Integer = 0 To MAX_Q_SIZE - 1
		BuyerQ(i).BuyerID = BLANK
		BuyerQ(i).WaitingTime = 0
		BuyerQ(i).ItemsInBasket = 0
		Next
		End Sub
		
		Sub ChangeSettings(ByRef SimulationTime As Integer, ByRef NoOfTills As Integer, **Tills(,) As Integer**) **'Q17 MP5 part**
		SimulationTime = 10
		NoOfTills = 2
		Console.WriteLine("Settings set for this simulation:")
		Console.WriteLine("================================")
		Console.WriteLine($"Simulation time: {SimulationTime}")
		Console.WriteLine($"Tills operating: {NoOfTills}")
		Console.WriteLine("================================")
		Console.WriteLine()
		Console.Write("Do you wish to change the settings? Y/N: ")
		Dim Answer As String = Console.ReadLine()
		If Answer = "Y" Then
		Console.WriteLine($"Maximum simulation time is {MAX_TIME} time units")
		Console.Write("Simulation run time: ")
		SimulationTime = Convert.ToInt32(Console.ReadLine())
		While SimulationTime > MAX_TIME Or SimulationTime < 1
		Console.WriteLine($"Maximum simulation time is {MAX_TIME} time units")
		Console.Write("Simulation run time: ")
		SimulationTime = Convert.ToInt32(Console.ReadLine())
		End While
		Console.WriteLine($"Maximum number of tills is {MAX_TILLS}")
		Console.Write("Number of tills in use: ")
		NoOfTills = Convert.ToInt32(Console.ReadLine())
		While NoOfTills > MAX_TILLS Or NoOfTills < 1
		Console.WriteLine($"Maximum number of tills is {MAX_TILLS}")
		Console.Write("Number of tills in use: ")
		NoOfTills = Convert.ToInt32(Console.ReadLine())
		End While
		For count = 1 To NoOfTills ' Q17 MP6
		Console.WriteLine($"Enter operator speed for till {count}.")
		Console.Write($"Default value is {Tills(count, 3)}:") ' Q17 MP7
		Tills(count, 3) = Console.ReadLine() ' Q17 MP8
		Next
			8

```
    End If
  End Sub

  Sub CalculateServingTime(Tills(,) As Integer, ThisTill As Integer, NoOfItems
As Integer)
    Dim ServingTime As Integer = (NoOfItems \ Tills(ThisTill, 3)) + 1  ' Q17
MP2
    Tills(ThisTill, TIME_SERVING) = ServingTime
    Console.WriteLine($"{ThisTill,6}{ServingTime,6}")
  End Sub

  Sub QueueSimulator()
    Dim BuyerNumber As Integer = 0
    Dim QLength As Integer = 0
    Dim Stats(9) As Integer
    Dim Tills(MAX_TILLS, 3) As Integer ' Q17 MP4
    Dim Data(MAX_TIME, 1) As Integer
    Dim BuyerQ(MAX_Q_SIZE - 1) As Q_Node
    Dim SimulationTime, NoOfTills, TimeToNextArrival, ExtraTime, TimeUnit As
Integer
    ResetDataStructures(Stats, Tills, BuyerQ)
    ChangeSettings(SimulationTime, NoOfTills, Tills) ' MP5 part
……
```

18 **1**

```
Sub ServeBuyerExpress(ByRef BuyerQ() As Q_Node, ByRef QLength As Integer,
ByRef Stats() As Integer, ByVal Tills(,) As Integer)
   Dim BuyerID As String
   Dim WaitingTime, Items As Integer
   Dim pos As Integer = 0
   While pos < QLength - 1 And BuyerQ(pos).ItemsInBasket > 9  ' Q18 MP2
MP5
      pos += 1
   End While
   If BuyerQ(pos).ItemsInBasket < 10 Then
      BuyerID = BuyerQ(pos).BuyerID   ' Q18 MP6
      WaitingTime = BuyerQ(pos).WaitingTime
      Items = BuyerQ(pos).ItemsInBasket
      For Count = pos To QLength – 1    ' Q18 MP3  MP7
         BuyerQ(Count).BuyerID = BuyerQ(Count + 1).BuyerID
         BuyerQ(Count).WaitingTime = BuyerQ(Count + 1).WaitingTime
         BuyerQ(Count).ItemsInBasket = BuyerQ(Count + 1).ItemsInBasket
      Next
      BuyerQ(QLength).BuyerID = BLANK
      BuyerQ(QLength).WaitingTime = 0
      BuyerQ(QLength).ItemsInBasket = 0
      QLength -= 1
      Console.Write($"{BuyerID,17}")      ' Q18 MP8
      UpdateStats(Stats, WaitingTime)         ' Q18 MP9
      CalculateServingTime(Tills, 0, Items)   ' Q18 MP10
   End If
End Sub

  Sub Serving(Tills(,) As Integer, NoOfTills As Integer, BuyerQ() As
Q_Node, ByRef QLength As Integer, Stats() As Integer)
    Dim TillFree As Integer
    Dim BuyerID As String = ""
    Dim WaitingTime As Integer = 0
    Dim ItemsInBasket As Integer = 0
    If QLength > 0 And Tills(0, TIME_SERVING) = 0 Then  ' Q18 MP1 MP11
      ServeBuyerExpress(BuyerQ, QLength, Stats, Tills)  ' Q18 MP4
    End If
    TillFree = FindFreeTill(Tills, NoOfTills)
```

12

```
    While TillFree <> -1 And QLength > 0
      ServeBuyer(BuyerQ, QLength, BuyerID, WaitingTime, ItemsInBasket)
      UpdateStats(Stats, WaitingTime)
      CalculateServingTime(Tills, TillFree, ItemsInBasket)
      TillFree = FindFreeTill(Tills, NoOfTills)
    End While
    IncrementTimeWaiting(BuyerQ, QLength)
    UpdateTills(Tills, NoOfTills)
    If QLength > 0 Then
      Stats(TOTAL_Q_OCCURRENCE) += 1
      Stats(TOTAL_Q) += QLength
    End If
    If QLength > Stats(MAX_Q_LENGTH) Then
      Stats(MAX_Q_LENGTH) = QLength
    End If
    OutputTillAndQueueStates(Tills, NoOfTills, BuyerQ, QLength)
  End Sub

  Sub OutputTillAndQueueStates(Tills(,) As Integer, NoOfTills As Integer,
BuyerQ() As Q_Node, QLength As Integer)
    For i As Integer = 0 To NoOfTills   ' Q18 MP12
      Console.WriteLine($"{i,36}{Tills(i, TIME_IDLE),5}{Tills(i,
TIME_BUSY),5}{Tills(i, TIME_SERVING),6}")
    Next
    Console.WriteLine("
** Start of queue **")
    For i As Integer = 0 To QLength - 1

Console.WriteLine($"{BuyerQ(i).BuyerID,57}{BuyerQ(i).WaitingTime,7}{Buyer
Q(i).ItemsInBasket,6}")
    Next
    Console.WriteLine("
*** End of queue ***")
    Console.WriteLine("----------------------------------------------------
----------------------")
  End Sub
```

Python 3

04 **1**

```
Number1 = int(input("Enter an integer: "))      # MP1
Number2 = int(input("Enter another integer: ")) # MP2
if Number1 > Number2:    # MP3
  Number = Number1 // Number2
else:
  Number = Number2 // Number1  # MP4
Count = 0
while Count != Number: # MP5
  Count += 1
  if Count % 10 == 0:  # MP6
    print('X', end='')
  elif Count % 5 == 0: # MP7
    print('V', end='')
  else:
    print('/', end='') # MP8
```

8

16 **1**

```
# indices for Stats data structure
MAX_Q_LENGTH = 0
MAX_WAIT = 1
TOTAL_WAIT = 2
TOTAL_Q = 3
TOTAL_Q_OCCURENCE = 4
TOTAL_NO_WAIT = 5
TOTAL_SHUNS = 6  # Q16 MP1

def BuyerArrives(Data, BuyerQ, QLength, BuyerNumber, NoOfTills, Stats):
  if QLength < 5:# Q16 MP2
    print(f"  B{BuyerNumber}({Data[BuyerNumber][ITEMS]})")
    BuyerQ, QLength = BuyerJoinsQ(Data, BuyerQ, QLength, BuyerNumber)
  else:
    Stats[TOTAL_SHUNS] += 1   # Q16 MP3
    print(f"Buyer {BuyerNumber} shunned the queue") # Q16 MP4
    if NoOfTills < MAX_TILLS: # Q16 MP5
      NoOfTills += 1  # Q16 MP6
  return BuyerQ, QLength, NoOfTills, Stats

def OutputStats(Stats, BuyerNumber, SimulationTime):
  print("The simulation statistics are:")
  print("==============================")
  print(f"The maximum queue length was: {Stats[MAX_Q_LENGTH]} buyers")
  print(f"The maximum waiting time was: {Stats[MAX_WAIT]} time units")
  print(f"{BuyerNumber} buyers arrived during {SimulationTime} time units")
  AverageWaitingTime = round(Stats[TOTAL_WAIT] / BuyerNumber, 1)
  print(f"The average waiting time was: {AverageWaitingTime} time units")
  if Stats[TOTAL_Q_OCCURRENCE] > 0:
    AverageQLength = round(Stats[TOTAL_Q] / Stats[TOTAL_Q_OCCURRENCE], 2)
    print(f"The average queue length was: {AverageQLength} buyers")
  print(f"{Stats[TOTAL_NO_WAIT]} buyers did not need to queue")
  print(f"{Stats[TOTAL_SHUNS]} buyers turned away because the queue was too
long") # Q16 MP7
```

7

17 **1**

```
def ResetDataStructures():
  Stats = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
  Tills = [[0, 0, 0, 0] for i in range(MAX_TILLS + 1)] # Q17 MP4
  for Count in range(MAX_TILLS + 1): # Q17 MP1
    Tills[Count][3] = 7 - Count      # Q17 MP3
```

8

```
  BuyerQ = [Q_Node() for i in range(MAX_Q_SIZE)]
  return Stats, Tills, BuyerQ

def ChangeSettings(Tills): # Q17 MP5 part
  SimulationTime = 10
  NoOfTills = 2
  print("Settings set for this simulation:")
  print("================================")
  print(f"Simulation time: {SimulationTime}")
  print(f"Tills operating: {NoOfTills}")
  print("================================")
  print()
  Answer = input("Do you wish to change the settings?  Y/N: ")
  if Answer == 'Y':
    print(f"Maximum simulation time is {MAX_TIME} time units")
    SimulationTime = int(input("Simulation run time: "))
    while SimulationTime > MAX_TIME:
      print(f"Maximum simulation time is {MAX_TIME} time units")
      SimulationTime = int(input("Simulation run time: "))
    print(f"Maximum number of tills is {MAX_TILLS}")
    NoOfTills = int(input("Number of tills in use: "))
    while NoOfTills > MAX_TILLS:
      print(f"Maximum number of tills is {MAX_TILLS}")
      NoOfTills = int(input("Number of tills in use: "))
    for Count in range(1, NoOfTills + 1): # Q17 MP6
      Tills[Count][3] = int(input(f"Enter the till speed (items processed
per time unit (default {Tills[Count][3]}) for till {Count}): ")) # Q17 MP7
MP8
  return Tills, SimulationTime, NoOfTills

def CalculateServingTime(Tills, ThisTill, NoOfItems):
  ServingTime = (NoOfItems // Tills[ThisTill][3]) + 1  # Q17 MP2
  Tills[ThisTill][TIME_SERVING] = ServingTime
  print(f"{ThisTill:>6d}{ServingTime:>6d}")
  return Tills

def QueueSimulator():
  BuyerNumber = 0
  QLength = 0
  Stats, Tills, BuyerQ = ResetDataStructures()
  Tills, SimulationTime, NoOfTills = ChangeSettings(Tills) # Q17 MP5 part
```

A. TILL_SPEED instead of 3 as index

Alternative answer

```
def ResetDataStructures():
  Stats = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
  Tills = [[0, 0, 0, 7 - i] for i in range(MAX_TILLS + 1)] # Q17 MP1 MP4
                    # Q17 MP3
  BuyerQ = [Q_Node() for i in range(MAX_Q_SIZE)]
  return Stats, Tills, BuyerQ
```

18 **1**

```
def OutputTillAndQueueStates(Tills, NoOfTills, BuyerQ, QLength):
  for i in range(0, NoOfTills + 1): # Q18 MP12

print(f"{i:>36d}{Tills[i][TIME_IDLE]:>5d}{Tills[i][TIME_BUSY]:>5d}{Tills[i][TIME_SERVING]:>6d}")
  print("                                                ** Start of queue **")
  for i in range(QLength):
    print(f"{BuyerQ[i].BuyerID:>57s}{BuyerQ[i].WaitingTime:>7d}{BuyerQ[i].ItemsInBasket:>6d}")
  print("                                                *** End of queue ***")
  print("-----------------------------------------------------------------------")
```

12

```
def ServeBuyerExpress(Tills, BuyerQ, QLength, Stats):
  # find a buyer with fewer than 10 items
  Found = False
  EndOfQ = False
  NextInQ = 0
  while not Found and not EndOfQ: # Q18 MP2  # MP5
    if BuyerQ[NextInQ].ItemsInBasket < 10:
      Found = True
    else:
      NextInQ += 1
      if NextInQ == QLength:
        EndOfQ = True
  if Found:
    # move buyer out  # Q18 MP6
    ThisBuyerID = BuyerQ[NextInQ].BuyerID
    WaitingTime = BuyerQ[NextInQ].WaitingTime
    ItemsInBasket = BuyerQ[NextInQ].ItemsInBasket
    # close gap in queue
    for Count in range(NextInQ, QLength): # Q18 MP3 # Q18 MP7
      BuyerQ[Count].BuyerID = BuyerQ[Count + 1].BuyerID
      BuyerQ[Count].WaitingTime = BuyerQ[Count + 1].WaitingTime
      BuyerQ[Count].ItemsInBasket = BuyerQ[Count + 1].ItemsInBasket
    # blank last element
    BuyerQ[QLength].BuyerID = BLANK
    BuyerQ[QLength].WaitingTime = 0
    BuyerQ[QLength].ItemsInBasket = 0
    QLength -= 1
    print(f"{ThisBuyerID:>17s}", end='') # Q18 MP8
    # update stats #Q18 MP9
    Stats = UpdateStats(Stats, WaitingTime)
    # serve buyer at till 0 # Q18 MP10
    Tills = CalculateServingTime(Tills, 0, ItemsInBasket)
  return Tills, BuyerQ, QLength, Stats

def Serving(Tills, NoOfTills, BuyerQ, QLength, Stats):
  if QLength > 0: # Q18 MP1
    if Tills[0][TIME_SERVING] == 0: # Q18 MP11
      Tills, BuyerQ, QLength, Stats = ServeBuyerExpress(Tills, BuyerQ, QLength,
Stats) # Q18 MP4
  TillFree = FindFreeTill(Tills, NoOfTills)
  while TillFree != -1 and QLength > 0:
    BuyerQ, QLength, BuyerID, WaitingTime, ItemsInBasket = ServeBuyer(BuyerQ,
QLength)
    Stats = UpdateStats(Stats, WaitingTime)
    Tills = CalculateServingTime(Tills, TillFree, ItemsInBasket)
    TillFree = FindFreeTill(Tills, NoOfTills)
  BuyerQ = IncrementTimeWaiting(BuyerQ, QLength)
  Tills = UpdateTills(Tills, NoOfTills)
  if QLength > 0:
    Stats[TOTAL_Q_OCCURRENCE] += 1
    Stats[TOTAL_Q] += QLength
  if QLength > Stats[MAX_Q_LENGTH]:
    Stats[MAX_Q_LENGTH] = QLength
  OutputTillAndQueueStates(Tills, NoOfTills, BuyerQ, QLength)
  return  Tills, NoOfTills, BuyerQ, QLength, Stats
```

Python 2

04 **1** **8**

```
import sys
Number1 = int(raw_input("Enter an integer: "))      # MP1
Number2 = int(raw_input("Enter another integer: ")) # MP2
if Number1 > Number2:    # MP3
  Number = Number1 // Number2
else:
  Number = Number2 // Number1  # MP4
Count = 0
while Count != Number: # MP5
  Count += 1
  if Count % 10 == 0:  # MP6
    sys.stdout.write('X')
  elif Count % 5 == 0: # MP7
    sys.stdout.write('V')
  else:
    sys.stdout.write('/') # MP8
```

16 **1** **7**

```
# indices for Stats data structure
MAX_Q_LENGTH = 0
MAX_WAIT = 1
TOTAL_WAIT = 2
TOTAL_Q = 3
TOTAL_Q_OCCURRENCE = 4
TOTAL_NO_WAIT = 5
TOTAL_SHUNS = 6  # Q16 MP1

def BuyerArrives(Data, BuyerQ, QLength, BuyerNumber, NoOfTills, Stats):
  if QLength < 5:# Q16 MP2
    sys.stdout.write('  B{0}({1})'.format(BuyerNumber,
Data[BuyerNumber][ITEMS]))
    print
    BuyerQ, QLength = BuyerJoinsQ(Data, BuyerQ, QLength, BuyerNumber)
  else:
    Stats[TOTAL_SHUNS] += 1   # Q16 MP3
    print "Buyer ", BuyerNumber, " shunned the queue" # Q16 MP4
    if NoOfTills < MAX_TILLS: # Q16 MP5
      NoOfTills += 1  # Q16 MP6
  return BuyerQ, QLength, NoOfTills, Stats

def OutputStats(Stats, BuyerNumber, SimulationTime):
  print "The simulation statistics are:"
  print "=============================="
  print "The maximum queue length was: ", Stats[MAX_Q_LENGTH], " buyers"
  print "The maximum waiting time was: ", Stats[MAX_WAIT], " time units"
  print BuyerNumber, " buyers arrived during ", SimulationTime, " time
units"
  print "The average waiting time was:", round(Stats[TOTAL_WAIT]*1.0/
BuyerNumber, 1), "time units"
  if Stats[TOTAL_Q_OCCURRENCE] > 0:
    print "The average queue length was:", round(Stats[TOTAL_Q]*1.0 /
Stats[TOTAL_Q_OCCURRENCE], 2), "buyers"
  print Stats[TOTAL_NO_WAIT], " buyers did not need to queue"
  print Stats[TOTAL_SHUNS], " buyers turned away because the queue was too
long" # Q16 MP7
```

17 **1** **8**

```
def ResetDataStructures():
  Stats = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
  Tills = [[0, 0, 0, 0] for i in range(MAX_TILLS + 1)] # Q17 MP4
  for Count in range(MAX_TILLS + 1): # Q17 MP1
```

```
    Tills[Count][3] = 7 - Count      # Q17 MP3
  BuyerQ = [Q_Node() for i in range(MAX_Q_SIZE)]
  return Stats, Tills, BuyerQ

def ChangeSettings(Tills): # Q17 MP5 part
  SimulationTime = 10
  NoOfTills = 2
  print "Settings set for this simulation:"
  print "================================"
  print "Simulation time: ", SimulationTime
  print "Tills operating: ", NoOfTills
  print "================================"
  print
  Answer = raw_input("Do you wish to change the settings?  Y/N: ")
  if Answer == 'Y':
    print "Maximum simulation time is ", MAX_TIME, " time units"
    SimulationTime = int(raw_input("Simulation run time: "))
    while SimulationTime > MAX_TIME or SimulationTime < 1:
      print "Maximum simulation time is ", MAX_TIME, " time units"
      SimulationTime = int(raw_input("Simulation run time: "))
    print "Maximum number of tills is ", MAX_TILLS
    NoOfTills = int(raw_input("Number of tills in use: "))
    while NoOfTills > MAX_TILLS or NoOfTills < 1:
      print "Maximum number of tills is ", MAX_TILLS
      NoOfTills = int(raw_input("Number of tills in use: "))
    for Count in range(1, NoOfTills + 1): # Q17 MP6
      sys.stdout.write('Enter the till speed (items processed per time
unit (default {0}) for till {1}: '.format(Tills[Count][3], Count)) # Q17
MP7
      Tills[Count][3] = int(raw_input()) # Q17 MP8
  return Tills, SimulationTime, NoOfTills

def CalculateServingTime(Tills, ThisTill, NoOfItems):
  ServingTime = (NoOfItems // Tills[ThisTill][3]) + 1  # Q17 MP2
  Tills[ThisTill][TIME_SERVING] = ServingTime
  sys.stdout.write('{0:>6}{1:>6}'.format(ThisTill, ServingTime))
  print
  return Tills

def QueueSimulator():
  BuyerNumber = 0
  QLength = 0
  Stats, Tills, BuyerQ = ResetDataStructures()
  Tills, SimulationTime, NoOfTills = ChangeSettings(Tills) # Q17 MP5 part
```

18	**1**	**def ServeBuyerExpress (Tills, BuyerQ, QLength, Stats):** ** # find a buyer with fewer than 10 items** ** Found = False** ** EndOfQ = False** ** NextInQ = 0** ** while not Found and not EndOfQ: # Q18 MP2 # Q18 MP5** ** if BuyerQ[NextInQ].ItemsInBasket < 10:** ** Found = True** ** else:** ** NextInQ += 1** ** if NextInQ == QLength:** ** EndOfQ = True** ** if Found:** ** # move buyer out # Q18 MP6** ** ThisBuyerID = BuyerQ[NextInQ].BuyerID** ** WaitingTime = BuyerQ[NextInQ].WaitingTime**	**12**

```
    ItemsInBasket = BuyerQ[NextInQ].ItemsInBasket
    # close gap in queue
    for Count in range(NextInQ, QLength): # Q18 MP3 # Q18 MP7
      BuyerQ[Count].BuyerID = BuyerQ[Count + 1].BuyerID
      BuyerQ[Count].WaitingTime = BuyerQ[Count + 1].WaitingTime
      BuyerQ[Count].ItemsInBasket = BuyerQ[Count + 1].ItemsInBasket
    # blank last element
    BuyerQ[QLength].BuyerID = BLANK
    BuyerQ[QLength].WaitingTime = 0
    BuyerQ[QLength].ItemsInBasket = 0
    QLength -= 1
    sys.stdout.write('{0:>17}'.format(ThisBuyerID)) # Q18 MP8
    # update stats # Q18 MP9
    Stats = UpdateStats(Stats, WaitingTime)
    # serve buyer at till 0 # Q18 MP10
    Tills = CalculateServingTime(Tills, 0, ItemsInBasket)
  return Tills, BuyerQ, QLength, Stats

def Serving(Tills, NoOfTills, BuyerQ, QLength, Stats):
  if QLength > 0: # Q18 MP1
    if Tills[0][TIME_SERVING] == 0: # Q18 MP11
      Tills, BuyerQ, QLength, Stats = ServeBuyerExpress (Tills, BuyerQ,
QLength, Stats) # Q18 MP4
  TillFree = FindFreeTill(Tills, NoOfTills)
  while TillFree != -1 and QLength > 0:
    BuyerQ, QLength, BuyerID, WaitingTime, ItemsInBasket =
ServeBuyer(BuyerQ, QLength)
    Stats = UpdateStats(Stats, WaitingTime)
    Tills = CalculateServingTime(Tills, TillFree, ItemsInBasket)
    TillFree = FindFreeTill(Tills, NoOfTills)
  BuyerQ = IncrementTimeWaiting(BuyerQ, QLength)
  Tills = UpdateTills(Tills, NoOfTills)
  if QLength > 0:
    Stats[TOTAL_Q_OCCURRENCE] += 1
    Stats[TOTAL_Q] += QLength
  if QLength > Stats[MAX_Q_LENGTH]:
    Stats[MAX_Q_LENGTH] = QLength
  OutputTillAndQueueStates(Tills, NoOfTills, BuyerQ, QLength)
  return  Tills, NoOfTills, BuyerQ, QLength, Stats

def OutputTillAndQueueStates(Tills, NoOfTills, BuyerQ, QLength):
  for i in range(0, NoOfTills + 1): # Q18 MP12
    sys.stdout.write('{0:>36}{1:>5}{2:>5}{3:>6}'.format(i, Tills[i][TIME_IDLE],
Tills[i][TIME_BUSY], Tills[i][TIME_SERVING]))
    print
  print "                                                    ** Start of queue **"
  for i in range(QLength):
    sys.stdout.write('{0:>57}{1:>7}{2:>6}'.format(BuyerQ[i].BuyerID, BuyerQ[i].WaitingTime,
BuyerQ[i].ItemsInBasket))
    print
  print "                                                    *** End of queue ***"
  print "-------------------------------------------------------------------------"
```

Pascal

04 **1** **8**

```
var
  Number, Number1, Number2, Count: integer;
begin
  write('Enter an integer: ');
  readln(Number1);  // MP1
  write('Enter another integer: ');
  readln(Number2);  // MP2
  if Number1 > Number2 then  // MP3
    Number := Number1 div Number2
  else
    Number := Number2 div Number1;  // MP4
  Count := 0;
  while Count <> Number do  // MP5
    begin
    inc(Count);
    if Count mod 10 = 0 then  // MP6
      write('X')
    else
      if Count mod 5 = 0 then  // MP7
        write('V')
      else
        write('/');  // MP8
    end;
  readln;
end.
```

16 **1** **7**

```
  //  indices for Stats data structure
  MAX_Q_LENGTH = 0;
  MAX_WAIT = 1;
  TOTAL_WAIT = 2;
  TOTAL_Q = 3;
  TOTAL_Q_OCCURRENCE = 4;
  TOTAL_NO_WAIT = 5;
  TOTAL_SHUNS = 6; // Q16 MP1

  procedure BuyerArrives(var Data: TData; var BuyerQ:TBuyerQ; var QLength:
integer; BuyerNumber: integer; var NoOfTills: integer; var Stats: TStats);
  begin
    if QLength < 5 then // Q16 MP2
      begin
        writeln('  B', BuyerNumber, '(', Data[BuyerNumber, ITEMS], ')');
        BuyerJoinsQ(Data, BuyerQ, QLength, BuyerNumber);
      end
    else
      begin
        inc(Stats[TOTAL_SHUNS]); // Q16 MP3
        writeln('Buyer', BuyerNumber, ' shunned the queue'); // Q16 MP4
        if NoOfTills < MAX_TILLS then  // Q16 MP5
          inc(NoOfTills);  // Q16 MP6
      end;
  end;

procedure OutputStats(var Stats: TStats; BuyerNumber: integer;
SimulationTime: integer);
var
  AverageWaitingTime, AverageQLength: real;
begin
  writeln('The simulation statistics are:');
  writeln('==============================');
```

```
  writeln('The maximum queue length was: ', Stats[MAX_Q_LENGTH], '
buyers');
  writeln('The maximum waiting time was: ', Stats[MAX_WAIT], ' time
units');
  writeln(BuyerNumber, ' buyers arrived during ', SimulationTime, ' time
units');
  AverageWaitingTime := Stats[TOTAL_WAIT] / BuyerNumber;
  writeln('The average waiting time was: ', AverageWaitingTime:2:1, ' time
units');
  if Stats[TOTAL_Q_OCCURRENCE] > 0 then
    begin
      AverageQLength := Stats[TOTAL_Q] / Stats[TOTAL_Q_OCCURRENCE];
      writeln('The average queue length was: ', AverageQLength:2:2, '
buyers');
    end;
  writeln(Stats[TOTAL_NO_WAIT], ' buyers did not need to queue');
  writeln (Stats[TOTAL_SHUNS], ' buyers turned away because the queue was
too long'); // Q16 MP7
end;
```

17 **1** **8**

```
procedure ResetDataStructures(var Stats: TStats; var Tills: TTills; var
BuyerQ: TBuyerQ);
  var
    Count, i: integer;
  begin
    for i := 0 to 9 do
      begin
        Stats[i] := 0;
      end;
    for Count := 0 to MAX_TILLS do
      begin
        for i := 0 to 2 do
          begin
            Tills[Countl, i] := 0;
          end;
        Tills[Count, 3] := 7 – Count;    // Q17 MP1, MP3, MP4
      end;
    for i := 0 to MAX_Q_SIZE - 1 do
      begin
        BuyerQ[i].BuyerID := BLANK;
        BuyerQ[i].WaitingTime := 0;
        BuyerQ[i].ItemsInBasket := 0;
      end;
  end;

  procedure ChangeSettings(var SimulationTime: integer; var NoOfTills:
integer; var Tills: TTills);    // Q17 MP5 part
  var Answer: char;
    Count: integer;
  begin
    SimulationTime := 10;
    NoOfTills := 2;
    writeln('Settings set for this simulation:');
    writeln('================================');
    writeln('Simulation time: ', SimulationTime);
    writeln('Tills operating: ', NoOfTills);
    writeln('================================');
    writeln;
    write('Do you wish to change the settings?  Y/N: ');
```

```
    readln(Answer);
    if Answer = 'Y' then
      begin
        writeln('Maximum simulation time is ', MAX_TIME, ' time units');
        write('Simulation run time: ');
        readln(SimulationTime);
        while (SimulationTime > MAX_TIME) or (SimulationTime < 1) do
          begin
            writeln('Maximum simulation time is ', MAX_TIME, ' time
units');
            write('Simulation run time: ');
            readln(SimulationTime);
          end;
        writeln('Maximum number of tills is ', MAX_TILLS);
        write('Number of tills in use: ');
        readln(NoOfTills);
        while (NoOfTills > MAX_TILLS) or (NoOfTills < 1) do
          begin
            writeln('Maximum number of tills is ', MAX_TILLS);
            write('Number of tills in use: ');
            readln(NoOfTills);
          end;
        for Count := 1 to NoOfTills  do // Q17 MP6
          begin
            write('Enter the till speed (items processed per time unit
(default ', Tills[Count, 3], ')) for till ',  Count, ': '); // Q17 MP7
            readln(Tills[Count, 3]);  // Q17 MP8
          end;
      end;
  end;

  procedure CalculateServingTime(var Tills: TTills; ThisTill: integer;
NoOfItems: integer);
  var
    ServingTime: integer;
  begin
    ServingTime := (NoOfItems DIV Tills[ThisTill, 3]) + 1;  // Q17 MP2
    Tills[ThisTill][TIME_SERVING] := ServingTime;
    writeln(ThisTill:6, ServingTime:6);
  end;


type
    Q_Node = record
      BuyerID: string;
      WaitingTime: integer;
      ItemsInBasket: integer;
    end;
    TStats = array[0 .. 9] of integer;
    TTills = array[0 .. MAX_TILLS, 0 .. 3] of integer; // Q17 MP4
    TBuyerQ = array[0 .. MAX_Q_SIZE - 1] of Q_Node;
    TData = array[0 .. MAX_TIME, 0 .. 1] of integer


procedure QueueSimulator();
var
  BuyerNumber, QLength, SimulationTime, NoOfTills, TimeToNextArrival,
TimeUnit, ExtraTime: integer;
  Stats: TStats;
  Tills: TTills;
  BuyerQ: TBuyerQ;
```

```
  Data: TData;
begin
  BuyerNumber := 0;
  QLength := 0;
  ResetDataStructures(Stats, Tills, BuyerQ);
  ChangeSettings(SimulationTime, NoOfTills, Tills); // Q17 MP5 part
```

18 **1** **12**

```
procedure ServeBuyerExpress (var Tills: TTills; var BuyerQ: TBuyerQ; var
QLength: integer; var Stats: TStats);
var
  Found, EndOfQ: boolean;
  NextInQ, WaitingTime, ItemsInBasket, Count: integer;
  ThisBuyerID: string;
  begin
  // find a buyer with fewer than 10 items
  Found := False;
  EndOfQ := False;
  NextInQ := 0;
  while not Found and not EndOfQ do // Q18 MP2  // Q18 MP5
    if BuyerQ[NextInQ].ItemsInBasket < 10 then
      begin
        Found := True;
      end
    else
      begin
        inc(NextInQ);
        if NextInQ = QLength then
          begin
            EndOfQ := True;
          end;
      end;
  if Found then
    begin
      // move buyer out  // MP6
      ThisBuyerID := BuyerQ[NextInQ].BuyerID;
      WaitingTime := BuyerQ[NextInQ].WaitingTime;
      ItemsInBasket := BuyerQ[NextInQ].ItemsInBasket;
      // close gap in queue
      for Count := NextInQ to QLength do // Q18 MP3 // Q18 MP7
        begin
          BuyerQ[Count].BuyerID := BuyerQ[Count + 1].BuyerID;
          BuyerQ[Count].WaitingTime := BuyerQ[Count + 1].WaitingTime;
          BuyerQ[Count].ItemsInBasket := BuyerQ[Count + 1].ItemsInBasket;
        end;
      // blank last element
      BuyerQ[QLength].BuyerID := BLANK;
      BuyerQ[QLength].WaitingTime := 0;
      BuyerQ[QLength].ItemsInBasket := 0;
      dec(QLength);
      write(ThisBuyerID: 17); // Q18 MP8
      // update stats   // Q18 MP9
      UpdateStats(Stats, WaitingTime);
      // serve buyer at till 0   // Q18 MP10
      CalculateServingTime(Tills, 0, ItemsInBasket);
    end;
end;

  procedure OutputTillAndQueueStates(var Tills: TTills; NoOfTills:
integer; var BuyerQ: TBuyerQ; QLength: integer);
  var
    i: integer;
  begin
```

```
    for i  := 0 to NoOfTills do   //  Q18 MP12
      writeln(i:36, Tills[i, TIME_IDLE]:5, Tills[i, TIME_BUSY]:5, Tills[i,
TIME_SERVING]:6);
    writeln('                                                          ** Start
of queue **');
    for i := 0 to QLength - 1 do
      writeln(BuyerQ[i].BuyerID:57, BuyerQ[i].WaitingTime:7,
BuyerQ[i].ItemsInBasket:6);
    writeln('                                                          *** End
of queue ***');
    writeln('------------------------------------------------------------
-----------');
  end;

  procedure Serving(var Tills: TTills; var NoOfTills: integer; var BuyerQ:
TBuyerQ; var QLength: integer; var Stats: TStats);
  var
    TillFree: integer;
    BuyerID: string;
    WaitingTime, ItemsInBasket: integer;
  begin
    if QLength > 0 then   // Q18 MP1
      if Tills[0, TIME_SERVING] = 0 then // Q18 MP11
        ServeBuyerExpress (Tills, BuyerQ, QLength, Stats); // Q18 MP4
    TillFree := FindFreeTill(Tills, NoOfTills);
    while (TillFree <> -1) and (QLength > 0) do
      begin
        ServeBuyer(BuyerQ, QLength, BuyerID, WaitingTime, ItemsInBasket);
        UpdateStats(Stats, WaitingTime);
        CalculateServingTime(Tills, TillFree, ItemsInBasket);
        TillFree := FindFreeTill(Tills, NoOfTills);
      end;
    IncrementTimeWaiting(BuyerQ, QLength);
    UpdateTills(Tills, NoOfTills);
    if QLength > 0 then
      begin
        inc(Stats[TOTAL_Q_OCCURRENCE]);
        Stats[TOTAL_Q] := Stats[TOTAL_Q] + QLength;
      end;
    if QLength > Stats[MAX_Q_LENGTH] then
      begin
         Stats[MAX_Q_LENGTH] := QLength;
      end;
    OutputTillAndQueueStates(Tills, NoOfTills, BuyerQ, QLength);
  end;
```

C#

04 **1** **8**

```
Console.Write("Enter an integer: ");
int Number1 = Convert.ToInt32(Console.ReadLine()); // MP1
Console.Write("Enter another integer: ");
int Number2 = Convert.ToInt32(Console.ReadLine()); // MP2
int Number;
if (Number1 > Number2) // MP3
{
    Number = Number1 / Number2;
}
else
{
    Number = Number2 / Number1; // MP4
}
int Count = 0;
while (Count != Number) // MP5
{
    Count++;
    if (Count % 10 == 0) // MP6
    {
        Console.Write("X");
    }
    else
    {
        if (Count % 5 == 0) // MP7
        {
            Console.Write("V");
        }
        else
        {
            Console.Write("/"); // MP8
        }
    }
}
Console.ReadLine();
```

16 **1** **7**

```
// indices for Stats data structure
    const int MAX_Q_LENGTH = 0;
    const int MAX_WAIT = 1;
    const int TOTAL_WAIT = 2;
    const int TOTAL_Q = 3;
    const int TOTAL_Q_OCCURRENCE = 4;
    const int TOTAL_NO_WAIT = 5;
    const int TOTAL_SHUNS = 6; // Q16 MP1

    public static void BuyerArrives(int[,] Data, Q_Node[] BuyerQ, ref int
QLength, int BuyerNumber, ref int NoOfTills, int[] Stats)
    {
      if (QLength < 5) // Q16 MP2
      {
        Console.WriteLine($"  B{BuyerNumber}({Data[BuyerNumber,
ITEMS]})");
        BuyerJoinsQ(Data, BuyerQ, ref QLength, BuyerNumber);
      }
      else
      {
        Stats[TOTAL_SHUNS]++; // Q16 MP3
```

```
        Console.WriteLine($"  B{BuyerNumber} has shunned the queue.");
//Q16 MP4
        if (NoOfTills < MAX_TILLS) // Q16 MP5
        {
          NoOfTills++; // Q16 MP6
        }
      }
    }

    public static void OutputStats(int[] Stats, int BuyerNumber, int
SimulationTime)
    {
      double AverageWaitingTime, AverageQLength;
      Console.WriteLine("The simulation statistics are:");
      Console.WriteLine("============================");
      Console.WriteLine($"The maximum queue length was:
{Stats[MAX_Q_LENGTH]} buyers");
      Console.WriteLine($"The maximum waiting time was: {Stats[MAX_WAIT]}
time units");
      Console.WriteLine($"{BuyerNumber} buyers arrived during
{SimulationTime} time units");
      AverageWaitingTime = Math.Round((double)Stats[TOTAL_WAIT] /
BuyerNumber, 1);
      Console.WriteLine($"The average waiting time was:
{AverageWaitingTime} time units");
      if (Stats[TOTAL_Q_OCCURRENCE] > 0)
      {
        AverageQLength = Math.Round((double)Stats[TOTAL_Q] /
Stats[TOTAL_Q_OCCURRENCE], 2);
        Console.WriteLine($"The average queue length was: {AverageQLength}
buyers");
      }
      Console.WriteLine($"{Stats[TOTAL_NO_WAIT]} buyers did not need to
queue");
      Console.WriteLine($"{Stats[TOTAL_SHUNS]} buyers shunned the queue");
//Q16 MP7
    }
```

17 **1** **8**

```
    public static void ResetDataStructures(int[] Stats, int[,] Tills, Q_Node[]
BuyerQ)
    {
      for (int i = 0; i <= 9; i++)
      {
        Stats[i] = 0;
      }
      for (int Count = 0; Count <= MAX_TILLS; Count++)
      {
        for (int i= 0; i <= 2; i++)
        {
          Tills[Count, i] = 0;
        }
        Tills[Count, 3] = 7 - Count; // Q17 MP1, MP3, MP4
      }
      for (int i = 0; i < MAX_Q_SIZE; i++)
      {
        BuyerQ[i].BuyerID = BLANK;
        BuyerQ[i].WaitingTime = 0;
        BuyerQ[i].ItemsInBasket = 0;
      }
    }
```

```
    public static void ChangeSettings(ref int SimulationTime, ref int NoOfTills,
int[,] Tills) // Q17 MP5 part
    {
      SimulationTime = 10;
      NoOfTills = 2;
      Console.WriteLine("Settings set for this simulation:");
      Console.WriteLine("================================");
      Console.WriteLine($"Simulation time: {SimulationTime}");
      Console.WriteLine($"Tills operating: {NoOfTills}");
      Console.WriteLine("================================");
      Console.WriteLine();
      Console.Write("Do you wish to change the settings?  Y/N: ");
      string Answer = Console.ReadLine();
      if (Answer == "Y")
      {
        Console.WriteLine($"Maximum simulation time is {MAX_TIME} time units");
        Console.Write("Simulation run time: ");
        SimulationTime = Convert.ToInt32(Console.ReadLine());
        while (SimulationTime > MAX_TIME || SimulationTime < 1)
        {
          Console.WriteLine($"Maximum simulation time is {MAX_TIME} time units");
          Console.Write("Simulation run time: ");
          SimulationTime = Convert.ToInt32(Console.ReadLine());
        }
        Console.WriteLine($"Maximum number of tills is {MAX_TILLS}");
        Console.Write("Number of tills in use: ");
        NoOfTills = Convert.ToInt32(Console.ReadLine());
        while (NoOfTills > MAX_TILLS || NoOfTills < 1)
        {
          Console.WriteLine($"Maximum number of tills is {MAX_TILLS}");
          Console.Write("Number of tills in use: ");
          NoOfTills = Convert.ToInt32(Console.ReadLine());
        }
        for (int count = 1; count <= NoOfTills; count++) // Q17 MP6
        {
          Console.WriteLine($"Enter operator speed for till {count}.");
          Console.Write($"Default value is {Tills[count, 3]} :"); // Q17 MP7
          Tills[count, 3] = Convert.ToInt32(Console.ReadLine());  // Q17 MP8
        }
      }
    }

    public static void CalculateServingTime(int[,] Tills, int ThisTill, int
NoOfItems)
    {
      int ServingTime = (NoOfItems / Tills[ThisTill, 3]) + 1;  // Q17 MP2
      Tills[ThisTill, TIME_SERVING] = ServingTime;
      Console.WriteLine($"{ThisTill,6}{ServingTime,6}");
    }


    public static void QueueSimulator()
    {
      int BuyerNumber = 0;
      int QLength = 0;
      int[] Stats = new int[10];
      int[,] Tills = new int[MAX_TILLS + 1, 4]; // Q17 MP4;
      int[,] Data = new int[MAX_TIME + 1, 2];
```

```
        Q_Node[] BuyerQ = new Q_Node[MAX_Q_SIZE];
        int SimulationTime = 0;
        int NoOfTills = 0;
        int TimeToNextArrival;
        int ExtraTime;
        int TimeUnit;
        ResetDataStructures(Stats, Till s, BuyerQ);
        ChangeSettings(ref SimulationTime, ref NoOfTills, Tills); // Q17 MP5 part
```

18 **1**

```
    public static void ServeBuyerExpress(Q_Node[] BuyerQ, ref int QLength, int[]
Stats, int[,] Tills)
    {
      string BuyerID;
      int WaitingTime, Items;
      int pos = 0;
      while (pos < QLength - 1 & BuyerQ[pos].ItemsInBasket > 9)  // Q18 MP2 MP5
        pos += 1;
      if (BuyerQ[pos].ItemsInBasket < 10)
      {
        BuyerID = BuyerQ[pos].BuyerID;   // Q18 MP6
        WaitingTime = BuyerQ[pos].WaitingTime;
        Items = BuyerQ[pos].ItemsInBasket;
        for (int Count = pos; Count <= QLength; Count++)  // Q18 MP3  MP7
        {
          BuyerQ[Count].BuyerID = BuyerQ[Count + 1].BuyerID;
          BuyerQ[Count].WaitingTime = BuyerQ[Count + 1].WaitingTime;
          BuyerQ[Count].ItemsInBasket = BuyerQ[Count + 1].ItemsInBasket;
        }
        BuyerQ[QLength].BuyerID = BLANK;
        BuyerQ[QLength].WaitingTime = 0;
        BuyerQ[QLength].ItemsInBasket = 0;
        QLength -= 1;
        Console.Write($"{BuyerID,17}");   // Q18 MP8
        UpdateStats(Stats, WaitingTime);  // Q18 MP9
        CalculateServingTime(Till s, 0, Items);   // Q18 MP10
      }
    }

public static void OutputTillAndQueueStates(int[,] Tills, int NoOfTills, Q_Node[]
BuyerQ, int QLength)
{
  for (int i = 0; i <= NoOfTills; i++)  // Q18 MP12
  {
    Console.WriteLine($"{i,36}{Tills[i, TIME_IDLE],5}{Tills[i,
TIME_BUSY],5}{Tills[i, TIME_SERVING],6}");
  }
  Console.WriteLine("                                                    ** Start
of queue **");
  for (int i = 0; i < QLength; i++)
  {

Console.WriteLine($"{BuyerQ[i].BuyerID,57}{BuyerQ[i].WaitingTime,7}{BuyerQ[i].Ite
msInBasket,6}");
  }
  Console.WriteLine("                                                    *** End
of queue ***");
  Console.WriteLine("-----------------------------------------------------------
------------");
```

12

```
}

    public static void Serving(int[,] Tills, ref int NoOfTills, Q_Node[] BuyerQ,
ref int QLength, int[] Stats)
    {
      int TillFree;
      string BuyerID = "";
      int WaitingTime = 0;
      int ItemsInBasket = 0;
      if (QLength > 0) // Q18 MP1
      {
        if (Tills[0, TIME_SERVING] == 0)  // Q18 MP11
        {
          ServeBuyerExpress(BuyerQ, ref QLength, Stats, Tills);  // Q18 MP4
        }
      }
      TillFree = FindFreeTill(Tills, NoOfTills);
      while (TillFree != -1 && QLength > 0)
      {
        ServeBuyer(BuyerQ, ref QLength, ref BuyerID, ref WaitingTime, ref
ItemsInBasket);
        UpdateStats(Stats, WaitingTime);
        CalculateServingTime(Tills, TillFree, ItemsInBasket);
        TillFree = FindFreeTill(Tills, NoOfTills);
      }
      IncrementTimeWaiting(BuyerQ, QLength);
      UpdateTills(Tills, NoOfTills);
      if (QLength > 0)
      {
        Stats[TOTAL_Q_OCCURRENCE] += 1;
        Stats[TOTAL_Q] += QLength;
      }
      if (QLength > Stats[MAX_Q_LENGTH])
      {
        Stats[MAX_Q_LENGTH] = QLength;
      }
      OutputTillAndQueueStates(Tills, NoOfTills, BuyerQ, QLength);
    }
```

Java

04 **1**

```
Console.write("Enter an integer: ");
int number1 = Integer.parseInt(Console.readLine());  // MP1
Console.write("Enter another integer: ");
int number2 = Integer.parseInt(Console.readLine());// MP2
int number;
if (number1 > number2) { // MP3
    number = number1 / number2;
} else {
    number = number2 / number1;  // MP4
}
int count = 0;
while (count != number) {  // MP5
    count++;
    if (count % 10 == 0) {  // MP6
      Console.write("X");
    } else {
      if (count % 5 == 0) {  // MP7
        Console.write("V");
      } else {
        Console.write("/");  // MP8
      }
    }
}
```

8

16 **1**

```
// indices for Stats data structure
final int MAX_Q_LENGTH = 0;
final int MAX_WAIT = 1;
final int TOTAL_WAIT = 2;
final int TOTAL_Q = 3;
final int TOTAL_Q_OCCURRENCE = 4;
final int TOTAL_NO_WAIT = 5;
final int TOTAL_SHUN = 6; // Q16 MP1

int[] buyerArrives(int[][] data, Q_Node[] buyerQ, int qLength, int
buyerNumber, int noOfTills,  int[] stats) {
    if (qLength < 5) {  // QP16 MP2
        Console.writeLine(String.format("  B%d(%d)", buyerNumber,
data[buyerNumber][ITEMS]));
        qLength = buyerJoinsQ(data, buyerQ, qLength, buyerNumber);
    } else {
        if (noOfTills < MAX_TILLS) { // QP16 MP5
            noOfTills++;             // QP16 MP6
        }
        stats[TOTAL_SHUN]++; // Q16 MP3
        Console.writeLine(String.format("  B%d shuns the queue",
buyerNumber)); // QP16 MP4
    }
    return new int[] { qLength, noOfTills };
}

void outputstats(int[] stats, int buyerNumber, int simulationTime) {
    Console.writeLine("The simulation statistics are:");
    Console.writeLine("==============================");
    Console.writeLine(String.format("The maximum queue length was: %d
buyers", stats[MAX_Q_LENGTH]));
    Console.writeLine(String.format("The maximum waiting time was: %d
time units", stats[MAX_WAIT]));
    Console.writeLine(String.format("%d buyers arrived during %d time
units", buyerNumber, simulationTime));
```

7

		`    double averagewaitingTime = (double)Math.round((double)stats[TOTAL_WAIT] / buyerNumber *10)/10;`
		`    Console.writeLine(String.format("The average waiting time was: %.1f time units", averagewaitingTime));`
		`    if (stats[TOTAL_Q_OCCURRENCE] > 0) {`
		`        double averageqLength = (double)Math.round((double)stats[TOTAL_Q] / stats[TOTAL_Q_OCCURRENCE] * 100)/100;`
		`        Console.writeLine(String.format("The average queue length was: %.2f buyers", averageqLength));`
		`    }`
		`    Console.writeLine(String.format("%d buyers did not need to queue", stats[TOTAL_NO_WAIT]));`
		`    **Console.writeLine(String.format("%d buyers shunned the queue", stats[TOTAL_SHUN])); // QP16 MP7**`
		`}`	

17	**1**	`public void resetDataStructures(int[] stats, int[][] tills, Q_Node[] buyerQ) {`
		`    for (int i = 0; i <=9; i++) {`
		`        stats[i] = 0;`
		`    }`
		`    for (int count = 0; count <= MAX_TILLS; count++) {`
		`        for (int i = 0; i <= 2; i++) {`
		`            tills[count][i] = 0;`
		`        }`
		`        **tills[count][3] = 7 - count; // Q17 MP1 MP3**`
		`    }`
		`    for (int i = 0; i < MAX_Q_SIZE; i++) {`
		`        buyerQ[i] = new Q_Node();`
		`    }`
		`}`
		
		`**int[] changeSettings(int[][] tills) { // Q17 MP5 part**`
		`    int simulationTime = 10;`
		`    int noOfTills = 2;`
		`    Console.writeLine("Settings set for this simulation:");`
		`    Console.writeLine("================================");`
		`    Console.writeLine(String.format("Simulation time: %d", simulationTime));`
		`    Console.writeLine(String.format("Tills operating: %d", noOfTills));`
		`    Console.writeLine("================================");`
		`    Console.writeLine();`
		`    Console.write("Do you wish to change the settings?  Y/N: ");`
		`    String answer = Console.readLine();`
		`    if (answer.equals("Y")) {`
		`        Console.writeLine(String.format("Maximum simulation time is %d time units", MAX_TIME));`
		`        Console.write("Simulation run time: ");`
		`        simulationTime = Integer.parseInt(Console.readLine());`
		`        while (simulationTime > MAX_TIME || simulationTime < 1) {`
		`            Console.writeLine(String.format("Maximum simulation time is %d time units", MAX_TIME));`
		`            Console.write("Simulation run time: ");`
		`            simulationTime = Integer.parseInt(Console.readLine());`
		`        }`
		`        Console.writeLine(String.format("Maximum number of tills is %d", MAX_TILLS));`
		`        Console.write("Number of tills in use: ");`
		`        noOfTills = Integer.parseInt(Console.readLine());`
		`        while (noOfTills > MAX_TILLS || noOfTills < 1) {`
		`            Console.writeLine(String.format("Maximum number of tills is %d", MAX_TILLS));`
			8

```
            Console.write("Number of tills in use: ");
            noOfTills = Integer.parseInt(Console.readLine());
        }
        for (int i = 1; i <= noOfTills; i++) { // QP17 MP6
            Console.writeLine(String.format("Enter the till speed for
till %d, the current till speed is %d:", i, tills[i][3])); // Q17 MP7
            tills[i][3] = Integer.parseInt(Console.readLine()); // Q17
MP8
        }
    }
    return new int[] { simulationTime, noOfTills };
}

        void calculateServingTime(int[][] tills, int thisTill, int
noOfItems) {
            int servingTime = (noOfItems / tills[thisTill][3]) + 1; //
Q17 MP2
            tills[thisTill][TIME_SERVING] = servingTime;
            Console.writeLine(String.format("%6d%6d", thisTill,
servingTime));
        }

        public QueueSimulator() {
            int buyerNumber = 0, qLength = 0, simulationTime, noOfTills,
extraTime;
            int[] stats = new int[10];
            int[][] tills = new int[MAX_TILLS+1][4]; // Q17 MP4
            int[][] data = new int[MAX_TIME+1][2];
            Q_Node[] buyerQ = new Q_Node[MAX_Q_SIZE];
            resetDataStructures(stats, tills, buyerQ);
            int[] settings = changeSettings(tills); // Q17 MP5 part
…
```

18 **1**

```
void outputTillAndQueueStates(int[][] tills, int noOfTills, Q_Node[]
buyerQ, int qLength) {
    for (int i = 0; i <= noOfTills; i++) { // Q18 MP12
        Console.writeLine(String.format("%36d%5d%5d%6d", i,
tills[i][TIME_IDLE], tills[i][TIME_BUSY], tills[i][TIME_SERVING]));
    }
    Console.writeLine("
** Start of queue **");
    for (int i = 0; i < qLength; i++) {
        Console.writeLine(String.format("%57s%7d%6d", buyerQ[i].buyerID,
buyerQ[i].waitingTime, buyerQ[i].itemsInBasket));
    }
    Console.writeLine("
*** End of queue ***");
    Console.writeLine("----------------------------------------------------
----------------------");
}

int serveBuyerExpress(int[][] tills, Q_Node[] buyerQ, int qLength, int[]
stats)
{
    boolean foundUnder10Items = false;
    int buyerNumber = -1;
    while (!foundUnder10Items && buyerNumber < qLength - 1) { // Q18 MP2,
MP5
        buyerNumber += 1;
        if (buyerQ[buyerNumber].itemsInBasket < 10) {
            foundUnder10Items = true;
        }
```

12

```
    }
    if (foundUnder10Items) { // Q18 MP6
        String thisBuyerID = buyerQ[buyerNumber].buyerID;
        int thisBuyerWaitingTime = buyerQ[buyerNumber].waitingTime;
        int thisBuyerItems = buyerQ[buyerNumber].itemsInBasket;
        for (int count = buyerNumber; count < qLength; count++) { // Q18
MP3, MP7
            buyerQ[count].buyerID = buyerQ[count + 1].buyerID;
            buyerQ[count].waitingTime = buyerQ[count + 1].waitingTime;
            buyerQ[count].itemsInBasket = buyerQ[count +
1].itemsInBasket;
        }
        buyerQ[qLength].buyerID = BLANK;
        buyerQ[qLength].waitingTime = 0;
        buyerQ[qLength].itemsInBasket = 0;
        qLength -= 1;
        Console.write(String.format("%17s", thisBuyerID)); // Q18 MP8
        updateStats(stats, thisBuyerWaitingTime); //MP9
        calculateServingTime(tills, 0, thisBuyerItems); // Q18 MP10
    }
    return qLength;
}

int[] serving(int[][] tills, int noOfTills, Q_Node[] buyerQ, int qLength,
int[] stats) {
    int tillFree, waitingTime, itemsInBasket;
    if (qLength > 0) {  // Q18 MP1
        if (tills[0][TIME_SERVING] == 0) { // Q18 MP 11
            qLength = serveBuyerExpress(tills, buyerQ, qLength, stats);
// Q18 MP4
        }
    }
    tillFree = findFreeTill(tills, noOfTills);
    while (tillFree != -1 && qLength > 0) {
        String[] buyerInfo = serveBuyer(buyerQ, qLength);
        qLength = Integer.parseInt(buyerInfo[0]);
        String buyerID = buyerInfo[1];
        waitingTime = Integer.parseInt(buyerInfo[2]);
        itemsInBasket = Integer.parseInt(buyerInfo[3]);
        updateStats(stats, waitingTime);
        calculateServingTime(tills, tillFree, itemsInBasket);
        tillFree = findFreeTill(tills, noOfTills);
    }
    incrementTimeWaiting(buyerQ, qLength);
    updateTills(tills, noOfTills);
    if (qLength > 0) {
        stats[TOTAL_Q_OCCURRENCE] += 1;
        stats[TOTAL_Q] += qLength;
    }
    if (qLength > stats[MAX_Q_LENGTH]) {
        stats[MAX_Q_LENGTH] = qLength;
    }
    outputTillAndQueueStates(tills, noOfTills, buyerQ, qLength);
    return new int[] { noOfTills, qLength };
}
```