AQA

AS
Computer Science

7516/1 - Paper 1

Mark scheme

June 2018

Version/Stage: 1.0 Final

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 Assessment Writer.

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.

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

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.

Pages 5 to 14 contain the generic mark scheme.

Pages 15 to 48 contain the 'Program Source Codes' specific to the programming languages for questions 03, 11, 12, 13 and 14;

pages 20 to 29 – VB.NET
pages 30 to 36 – PYTHON 2
pages 37 to 43 – PYTHON 3
pages 44 to 50 – PASCAL/Delphi
pages 51 to 61 – C#
pages 62 to 69 – JAVA

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 candidates' responses to the most appropriate level according to **its overall quality**, 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 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.

<table>
<tr><th colspan="2">Qu</th><th>Marks</th></tr>
<tr><td>01</td><td>1</td><td>All marks for AO1 (knowledge)

<table>
<tr><td>Breaking a problem into a number of sub-problems</td><td>F</td></tr>
<tr><td>Models are put into action to solve problems</td><td>H</td></tr>
<tr><td>Combining procedures into compound procedures</td><td>G</td></tr>
<tr><td>Details are removed until the problem is represented in a way that is possible to solve because the problem reduces to one that has already been solved</td><td>E</td></tr>
</table>

1 mark per correct label

Note: each label must only be used once (if given more than once, ignore all occurrences)
A. handwritten answers
A. lower case</td><td>4</td></tr>
</table>

Qu		Marks	
02	1	All marks for A02 (apply)	3

Number	Root	d	FactorFound	r	Output
5	1				
	2				
	3				
		2	FALSE	1	
		3		2	
		4			Prime

1 mark for correct columns `Root` **and** `d`
1 mark for correct column `r`
1 mark for correct columns `FactorFound` **and** `Output`

Max 2 marks if any incorrect values written in table

3

I. Annotation indicating 'no value'
I. Indication of repeating values
I. Quotes
I. Case & spelling for FactorFound & Output
A. T, F instead of True False

Qu		Marks	
02	2	All marks for A02 (apply)	3

Number	Root	d	FactorFound	r	Output
25	1				
	2				
	3				
	4				
	5	2	False	1	
		3		1	
		4		1	
		5		0	
			True		
		6			Not prime

1 mark for correct columns `Root` **and** `d`
1 mark for correct column `r`
1 mark for correct columns `FactorFound` **and** `Output`

Max 2 marks if any incorrect values written in table

03	1	**All marks for AO3 (programming)** **Mark as follows:** 1) Correct variable declarations for `Number`, `c`, `k;` **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) `WHILE` loop with syntax allowed by the programming language and one correct condition for termination of the loop; 3) Second correct condition for while loop; 4) Correct prompt `"Enter a positive whole number: "` and `Number` assigned value entered by user; 5) Correct syntax for the `IF` statements inside attempt at loop; **A**. `IF … ELSEIF;` 6) correct contents in `IF` statements; 7) `FOR` loop with syntax allowed by the programming language over correct range; 8) Correct assignment to c inside `FOR` loop; 9) Output statement giving correct output; **A.** accept without spaces **I.** Ignore minor differences in case and spelling **R**. real `Number` **Max 8** if code does not function correctly	9

03	2	**Mark is for AO3 (evaluate)**
		
		****** SCREEN CAPTURE ******
		Must match code from 03.1, including prompts on screen capture matching those in code.
		Code for 03.1 must be sensible.
		
		Screen capture showing:
		'-3' being entered and the message 'Not a positive number.' displayed
		'11' being entered and the message 'Number too large.' displayed
		'10' being entered and line of numbers displayed
		
		`Enter a positive whole number: -3`
		`Not a positive number.`
		`Enter a positive whole number: 11`
		`Number too large.`
		`Enter a positive whole number: 10`
		`  1  9  36  84  126  126  84  36  9  1`
		
		`>`
		
		A. Alternative layout :
		`Enter a positive whole number: -3`
		`Not a positive number.`
		`Enter a positive whole number: 11`
		`Number too large.`
		`Enter a positive whole number: 10`
		`1`
		`9`
		`36`
		`84`
		`126`
		`126`
		`84`
		`36`
		`9`
		`1`
		`>`
		
		A. input on new line
			1

04	1	**Mark is for AO1 (understanding)** `LetterEnd // ProgramEnd ;` **R.** if any additional code **R.** if spelt incorrectly **I.** case & spacing	1

<table>
<tr><td>04</td><td>2</td><td>Mark is for AO1 (understanding)

<code>GetMenuOption // Decode ;</code> A. <code>GetNextSymbol</code>

R. if any additional code
R. if spelt incorrectly
I. case & spacing</td><td>1</td></tr>
<tr><td>05</td><td></td><td>All marks for AO1 (understanding)

<table>
<tr><th>Identifier</th><th>Description</th></tr>
<tr><td><code>PlainTextLetter</code></td><td>uncoded letter, part of <code>PlainText</code></td></tr>
<tr><td><code>Signal</code></td><td>single unit of <code>Transmission</code> (= or <code>SPACE</code> or <code>EOL</code>)</td></tr>
<tr><td><code>FirstSignal</code></td><td>first character in <code>Transmission</code></td></tr>
<tr><td><code>Symbol</code></td><td>used to build <code>SymbolString</code> (. or –)</td></tr>
</table>

R. if any additional code
R. if spelt incorrectly
I. case & spacing

1 mark per correct identifier

Note: each identifier must only be used once (if given more than once, ignore all occurrences)</td><td>4</td></tr>
</table>

06		**Mark is for AO1 (understanding)** 1) makes a subroutine self-contained; 2) releases storage when subroutine terminates; 3) able to test/debug subroutine independently ; 4) easier to re-use subroutine in another program; 5) local variable values cannot be inadvertently/accidentally altered by a subroutine call from the subroutine; **Max 1**	1
07	1	**All marks for AO2 (analyse)** the variable is used as the index of /pointer to / iterator for / place value; … the current character/symbol/signal in the transmission string;	2
07	2	**All marks for AO2 (analyse)** 1) empty string returned from `StripLeadingSpaces` (and assigned to `Transmission)` // generate empty string in `StripLeadingSpaces`; 2) `StripLeadingSpaces` (calls ReportError) to display "No signal received" 3) empty string is returned to `ReceiveMorseCode` // empty string is returned from `GetTransmission`; 4) `LastChar` is set to -1; 5) so loop is not entered; 6) `MorseCodeString` / `PlainText` remain empty strings; **Max 5**	5
08		**All marks for AO2 (analyse)** 1) Any example string with only two consecutive symbols, for example "==" // any example string with more than 3 consecutive symbols, such as "====" ; ***Note:*** *"xxx" would not cause an error.* ***Note:*** *It does not matter which non-space symbol is used in transmission.* 2) The while loop counts the number of consecutive non-spaces; 3) If this number is not 0, 1 or 3, (it calls the `ReportError` subroutine); **I**. quotes	3

09	1	**Mark is for AO2 (analyse)** `GetNextLetter;` **R.** if any additional code **R.** if spelt incorrectly **I.** case & spacing	1
09	2	**Mark is for AO2 (analyse)** `GetNextSymbol;` **R.** if any additional code **R.** if spelt incorrectly **I.** case & spacing	1
10		**All marks AO2 (analyse)** **mark as follows:** 1) include digits 0 to 9 in `Letter` array; 2) include Morse codes in `MorseCode` array for digit characters; 3) extend `Dash` and `Dot` array; 4) at the corresponding positions some of the zeros will need to change (to include new pointers) // binary tree to include routes to digit characters; 5) `Decode` subroutine needs no changes; 6) `SendMorseCode` needs to test for digits; 7) Explain a method to look up Morse code for digits (eg linear search of `Letter` array); Unusual answers should be referred to the PE **A**. answers using dictionary for digits **A**. Letter and MorseCode arrays need changing; for **1 mark** if (1) and (2) not awarded **Max 6**	6

11	1	**1 mark for AO3 (design) and 3 marks for AO3 (programming)** **Mark as follows:** **AO3 (design) – 1 mark:** 1) Identifying that a selection statement (or equivalent method) is required to test that character is within range of uppercase letters or is a space // identifying that selection statement needs modifying (e.g.`if Char in Letter...);` **AO3 (programming) – 3 marks:** 2) Selection structure is created with correct logic so that if error detected it ensures error message is displayed only once & subroutine exits; 3) calls `ReportError` subroutine with suitable message if error in input string; 4) final value of `MorseCodeString` set to `EMPTYSTRING` (accept `''` or `SPACE`) if error in input string; **A**. accept if `MorseCodeString` set to `EMPTYSTRING` initially and not changed.	4

11	2	**Mark is for AO3 (evaluate)**
		
		****** SCREEN CAPTURE ******
		Must match code from 11.1, including prompts on screen capture matching those in code.
		Code for 11.1 must be sensible.
		
		Screen capture showing:
		'S' being entered followed by 'Help' and suitable message displayed
		
		`Main Menu`
		`=========`
		`R - Receive Morse code`
		`S - Send Morse code`
		`X - Exit program`
		
		`Enter your choice: S`
		`Enter your message (uppercase letters and spaces only):`
		`Help`
		`*  Invalid character entered  *`
		
		A. any suitable message, but must be within *s
			1

12	1	**1 mark for AO3 (design) and 6 marks for AO3 (programming)** **Mark as follows:** **AO3 (design) – 1 mark:** 1) Identifying that within an iterative statement a selection statement (or equivalent method) is required to test whether the Morse code is a dot, a dash or a space; **AO3 (programming) – 5 marks:** 2) Correct subroutine heading (`SendSignals`) and ending and correct parameter (`MorseCodeString`) ; 3) loop for each character in `MorseCodeString`; 4) start with empty string and keep adding a symbol string; 5) at least one conversion of dot, dash or space to the correct symbol string; 6) dot, dash and space converted to the correct symbol string; 7) output the signals correctly;	7

12	2	**Mark is for AO3 (evaluate)**
		
		****** SCREEN CAPTURE ******
		Must match code from 12.1, including prompts on screen capture matching those in code.
		Code for 12.1 must be sensible.
		
		Screen capture showing:
		`S` being entered followed by `MORSE X`
		and the string **=== === === === === = === = = = = = = === = = ===** ‘ being displayed after the Morse code.
		
		`Main Menu`
		`=========`
		`R - Receive Morse code`
		`S - Send Morse code`
		`X - Exit program`
		
		`Enter your choice: S`
		`Enter your message (uppercase letters and spaces only):`
		`MORSE X`
		`-- --- .-. ... .  -..-`
		`=== ===  === === ===  = === =  = = =  =  === = = ===`
			1

13	1	**2 marks for AO3 (design) and 4 marks for AO3 (programming)** **Note** that AO3 (design) marks 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 regardless of whether the solution works.	6

Level	Description	Mark Range
3	A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution. Code is written to ensure that all letters are output with their corresponding Morse code. The formatting of each line has been considered. A formal interface is used to pass the data structures' data into the subroutine. All of the appropriate design decisions have been taken.	5-6
2	There is evidence that a line of reasoning has been partially followed. The formatting of each line does not fully comply with requirements. There is evidence of some appropriate design work. There is Morse code output for each letter.	3-4
1	An attempt has been made to create `OutputAlphabetWithCode` and some appropriate programming statements have been written. There is insufficient 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 subroutine will have very little or none of the required functionality. It is unlikely that any of the key design elements of the task have been recognised.	1-2

Marking guidance:

Evidence of AO3 design – 2 points:

Evidence of design to look for in response:

1) identify the need for an iterative statement to act on each letter in turn
2) identify a method to output four letters per line

Evidence of AO3 programming – 7 points:
Evidence of programming to look for in response:

3) add option `A` to `DisplayMenu` subroutine
4) add test for new option and call `OutputAlphabetWithCode` with correct parameters
5) create new subroutine `OutputAlphabetWithCode` with correct parameters
6) loop from A to Z to output each letter and corresponding code separated from

		letter by one space (**A**. two spaces)	

13	2	**Mark is for AO3 (evaluate)**
		
		****** SCREEN CAPTURE ******
		Must match code from 13.1, including prompts on screen capture matching those in code.
		Code for 13.1 must be sensible.
		
		Screen capture showing:
		main menu with new option `A`
		'`A`' being entered and alphabet with Morse codes displayed
		
		`Main Menu`
		`=========`
		`R – Receive Morse code`
		`S – Send Morse code`
		`A – Output alphabet with Morse code`
		`X – Exit program`
		
		`Enter your choice: A`
		`A .-  B -...  C -.-.  D -..`
		`E .  F ..-.  G --.  H ....`
		`I ..  J .---  K -.-  L .-..`
		`M --  N -.  O ---  P .--.`
		`Q --.-  R .-.  S ...  T -`
		`U ..-  V ...-  W .--  X -..-`
		`Y -.--  Z --..`
		
		If not in columns as shown, do not award screen capture mark
			1

14	1	**3 marks for AO3 (design) and 6 marks for AO3 (programming)** **Note** that AO3 (design) marks 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 regardless of whether the solution works.	9

Level	Description	Mark Range
3	A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution. Code is written to ensure that each letter of the message is encrypted using the user-supplied keys. All of the appropriate design decisions have been taken.	7-9
2	There is evidence that a line of reasoning has been partially followed. The encryption of each character does not fully comply with requirements. There is evidence of some appropriate design work.	4-6
1	An attempt has been made to amend the subroutines. 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.	1-3

Marking guidance:

Evidence of AO3 design – 3 points:

Evidence of design to look for in response:

1) identifying the need to validate a key is an integer
2) identifying a method to encrypt each character with a key
3) identifying suitable method to alternate keys depending on character position in message

Evidence of AO3 programming – 6 points:

Evidence of programming to look for in response:

4) in `SendReceiveMessages` correctly store 3 integer keys entered by the user (in a list or separate variables)
5) amend call and subroutine header of `SendMorseCode` to include keys as parameter(s)
6) correctly encrypt first three characters of message
7) correctly encrypt all characters in message
8) ensure index is within range of array subscripts
9) code to encrypt character inserted in suitable place in `SendMorseCode`

14	2	**Mark is for AO3 (evaluate)**
		
		****** SCREEN CAPTURE ******
		Must match code from 14.1, including prompts on screen capture matching those in code.
		Code for 14.1 must be sensible.
		
		Screen capture showing:
		`17, 5 and -3` being entered followed by option `S` and then `TEA X` followed by the output `.--- .--- -.-- --.- -...`
		
		`Enter encryption key (integer): 17`
		`Enter encryption key (integer): 5`
		`Enter encryption key (integer): -3`
		
		`Main Menu`
		`=========`
		`R - Receive Morse code`
		`S - Send Morse code`
		`X - Exit program`
		
		`Enter your choice: S`
		`Enter your message (uppercase letters and spaces only):`
		`TEA X`
		`.--- .--- -.-- --.- -...`
			1

		Total	75

VB.NET

03	1	Sub Main()
		  Dim Number As Integer
		  Dim c As Integer
		  Number = 0
		  While Number < 1 Or Number > 10
		  Console.Write("Enter a positive whole number: ")
		  Number = Console.ReadLine
		  If Number > 10 Then
		  Console.WriteLine("Number too large")
		  Else
		  If Number < 1 Then
		  Console.WriteLine("Not a positive number.")
		  End If
		  End If
		  End While
		  c = 1
		  For k = 0 To Number - 1
		  Console.WriteLine(c)
		  c = (c * (Number - 1 - k)) \ (k + 1)
		  Next
		  Console.ReadLine()
		End Sub
			9

11	1	Sub SendMorseCode(ByVal MorseCode() As String)
		  Dim PlainText As String
		  Dim PlainTextLength As Integer
		  Dim MorseCodeString As String
		  Dim PlainTextLetter As Char
		  Dim CodedLetter As String
		  Dim Index As Integer
		  Console.Write("Enter your message (uppercase letters and spaces only): ")
		  PlainText = Console.ReadLine()
		  PlainTextLength = PlainText.Length()
		  MorseCodeString = EMPTYSTRING
		  For i = 0 To PlainTextLength - 1
		  PlainTextLetter = PlainText(i)
		  If PlainTextLetter = SPACE Then
		  Index = 0
		  Else**If PlainTextLetter >= "A" And PlainTextLetter <= "Z" Then**
		  Index = Asc(PlainTextLetter) - Asc("A") + 1
		  Else
		  ReportError("Invalid character entered")
		  Index = 0
		  MorseCodeString = EMPTYSTRING
		  Exit For
		  End If
		  CodedLetter = MorseCode(Index)
		  MorseCodeString = MorseCodeString + CodedLetter + SPACE
		  Next
		  Console.WriteLine(MorseCodeString)
		End Sub
		
		Alternative answer:
		Sub SendMorseCode(ByVal MorseCode() As String)
		  Dim PlainText As String
		  Dim PlainTextLength As Integer
		  Dim MorseCodeString As String
		  Dim PlainTextLetter As Char
		  Dim CodedLetter As String
		  Dim Index As Integer
		  Console.Write("Enter your message (uppercase letters and spaces only): ")
		  PlainText = Console.ReadLine()
		  Dim Valid As Boolean = True
		  For Each ch In PlainText
		  If ch <> SPACE Then
		  If ch < "A" Or ch > "Z" Then
		  Valid = False
		  MorseCodeString = EMPTYSTYRING
		  ReportError("Invalid character entered")
		  End If
			4

```
    End If
  Next
  If Valid Then
    PlainTextLength = PlainText.Length()
    MorseCodeString = EMPTYSTRING
    For i = 0 To PlainTextLength - 1
      PlainTextLetter = PlainText(i)
      If PlainTextLetter = SPACE Then
        Index = 0
      Else
        Index = Asc(PlainTextLetter) - Asc("A") + 1
      End If
      CodedLetter = MorseCode(Index)
      MorseCodeString = MorseCodeString + CodedLetter +
SPACE
    Next
  End If
  Console.WriteLine(MorseCodeString)
End Sub
```

12	1	Sub SendSignals(ByVal MorseCodeString As String)
		  Dim Transmission As String = EMPTYSTRING
		  Dim CodeStringLength = MorseCodeString.Length()
		  Dim Symbol As String
		  Dim SymbolString As String
		  For i = 0 To CodeStringLength - 1
		  Symbol = MorseCodeString(i)
		  If Symbol = "." Then
		  SymbolString = "= "
		  ElseIf Symbol = "-" Then
		  SymbolString = "=== "
		  ElseIf Symbol = Space Then
		  SymbolString = SPACE + SPACE
		  End If
		  Transmission += SymbolString
		  Next
		  Console.WriteLine(Transmission)
		End Sub
			7

13 1 6

```
Sub OutputAlphabetWithCode(ByVal Letter() As String, ByVal
MorseCode() As String)
  For Ptr = 1 To 26
    Console.Write(Letter(Ptr) + " ")
    Console.Write(MorseCode(Ptr).PadRight(6))
    If Ptr Mod 4 = 0 Then
      Console.WriteLine()
    End If
  Next
  Console.WriteLine()
End Sub

Sub DisplayMenu()
  Console.WriteLine()
  Console.WriteLine("Main Menu")
  Console.WriteLine("=========")
  Console.WriteLine("R - Receive Morse code")
  Console.WriteLine("S - Send Morse code")
  Console.WriteLine("A - Output alphabet with Morse code")
  Console.WriteLine("X - Exit Program")
  Console.WriteLine()
End Sub

Sub SendReceiveMessages()
  Dim Dash = {20, 23, 0, 0, 24, 1, 0, 17, 0, 21, 0, 25, 0,
15, 11, 0, 0, 0, 0, 22, 13, 0, 0, 10, 0, 0, 0}
  Dim Letter = {"SPACE", "A", "B", "C", "D", "E", "F",
"G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q",
"R", "S", "T", "U", "V", "W", "X", "Y", "Z"}
  Dim Dot = {5, 18, 0, 0, 2, 9, 0, 26, 0, 19, 0, 3, 0, 7,
4, 0, 0, 0, 12, 8, 14, 6, 0, 16, 0, 0, 0}
  Dim MorseCode = {"SPACE", ".-", "-...", "-.-.", "-..",
".", "..-.", "--.", "....", "..", ".---", "-.-", ".-..",
"--", "-.", "---", ".--.", "--.-", ".-.", "...", "-", "..-
", "...-", ".--", "-..-", "-.--", "--.."}
  Dim MenuOption As String
  Dim ProgramEnd As Boolean = False
  While Not ProgramEnd
    DisplayMenu()
    MenuOption = GetMenuOption()
    If MenuOption = "R" Then
      ReceiveMorseCode(Dash, Letter, Dot)
    ElseIf MenuOption = "S" Then
      SendMorseCode(MorseCode)
    ElseIf MenuOption = "A" Then
      OutputAlphabetWithCode(Letter, MorseCode)
    ElseIf MenuOption = "X" Then
      ProgramEnd = True
    End If
  End While
End Sub
```

14	1	Sub SendMorseCode(ByVal MorseCode() As String, **ByVal Keys() As Integer**)
		  Dim PlainText As String
		  Dim PlainTextLength As Integer
		  Dim MorseCodeString As String
		  Dim PlainTextLetter As Char
		  Dim CodedLetter As String
		  Dim Index As Integer
		  Console.Write("Enter your message (uppercase letters and spaces only): ")
		  PlainText = Console.ReadLine()
		  Dim Valid As Boolean = True
		  For Each ch In PlainText
		  If ch <> SPACE Then
		  If ch < "A" Or ch > "Z" Then
		  Valid = False
		  MorseCodeString = EMPTYSTRING
		  ReportError("Invalid character entered")
		  End If
		  End If
		  Next
		  If Valid Then
		  PlainTextLength = PlainText.Length()
		  MorseCodeString = EMPTYSTRING
		  For i = 0 To PlainTextLength - 1
		  PlainTextLetter = PlainText(i)
		  If PlainTextLetter = SPACE Then
		  Index = 0
		  Else
		  Index = Asc(PlainTextLetter) - Asc("A") + 1
		  End If
		  Index += Keys(i Mod 3)
		  While Index < 0
		  Index += 27
		  End While
		  While Index >= 27
		  Index -= 27
		  End While
		  CodedLetter = MorseCode(Index)
		  MorseCodeString = MorseCodeString + CodedLetter + SPACE
		  Next
		  End If
		  Console.WriteLine(MorseCodeString)
		  SendSignals(MorseCodeString)
		End Sub
		
		Sub SendReceiveMessages()
		  Dim Dash = {20, 23, 0, 0, 24, 1, 0, 17, 0, 21, 0, 25, 0, 15, 11, 0, 0, 0, 0, 22, 13, 0, 0, 10, 0, 0, 0}
		  Dim Letter = {"SPACE", "A", "B", "C", "D", "E", "F",
			9

```
"G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q",
"R", "S", "T", "U", "V", "W", "X", "Y", "Z"}
  Dim Dot = {5, 18, 0, 0, 2, 9, 0, 26, 0, 19, 0, 3, 0, 7,
4, 0, 0, 0, 12, 8, 14, 6, 0, 16, 0, 0, 0}
  Dim MorseCode = {"SPACE", ".-", "-...", "-.-.", "-..",
".", "..-.", "--.", "....", "..", ".---", "-.-", ".-..",
"--", "-.", "---", ".--.", "--.-", ".-.", "...", "-", "..-
", "...-", ".--", "-..-", "-.--", "--.."}
  Dim MenuOption As String
  Dim ProgramEnd As Boolean = False
  Dim Keys() As Integer = {0, 0, 0}
  Dim ValidDisplacement As Boolean
  Dim Displacement As Integer
  For i = 0 To 2
    ValidDisplacement = False
    While Not ValidDisplacement
      Try
        Console.Write("Enter encryption key (integer): ")
        Displacement = Console.ReadLine
        ValidDisplacement = True
      Catch ex As Exception
      End Try
      Keys(i) = Displacement
    End While
  Next
  While Not ProgramEnd
    DisplayMenu()
    MenuOption = GetMenuOption()
    If MenuOption = "R" Then
      ReceiveMorseCode(Dash, Letter, Dot)
    ElseIf MenuOption = "S" Then
      SendMorseCode(MorseCode, Keys)
    ElseIf MenuOption = "A" Then
      OutputAlphabetWithCode(Letter, MorseCode)
    ElseIf MenuOption = "X" Then
      ProgramEnd = True
    End If
  End While
End Sub
```

Alternative answer:

```
Sub SendMorseCode(ByVal MorseCode() As String, ByVal Key1
As Integer, ByVal Key2 As Integer, ByVal Key3 As Integer)
  Dim PlainText As String
  Dim PlainTextLength As Integer
  Dim MorseCodeString As String
  Dim PlainTextLetter As Char
  Dim CodedLetter As String
  Dim Index As Integer
  Dim Displacement As Integer
  Console.Write("Enter your message (uppercase letters and
spaces only): ")
  PlainText = Console.ReadLine()
  Dim Valid As Boolean = True
  For Each ch In PlainText
    If ch <> SPACE Then
      If ch < "A" Or ch > "Z" Then
        Valid = False
        MorseCodeString = EMPTYSTRING
        ReportError("Invalid character entered")
      End If
    End If
  Next
  If Valid Then
    PlainTextLength = PlainText.Length()
    MorseCodeString = EMPTYSTRING
    For i = 0 To PlainTextLength - 1
      PlainTextLetter = PlainText(i)
      If PlainTextLetter = SPACE Then
        Index = 0
      Else
        Index = Asc(PlainTextLetter) - Asc("A") + 1
      End If
      If i Mod 3 = 0 Then
        Displacement = Key1
      ElseIf i Mod 3 = 1 Then
        Displacement = Key2
      Else
        Displacement = Key3
      End If
      Index += Displacement
      While Index < 0
        Index += 27
      End While
      While Index >= 27
        Index -= 27
      End While
      CodedLetter = MorseCode(Index)
      MorseCodeString = MorseCodeString + CodedLetter +
SPACE
    Next
```

```
  End If
  Console.WriteLine(MorseCodeString)
  SendSignals(MorseCodeString)
End Sub

Sub SendReceiveMessages()
  Dim Dash = {20, 23, 0, 0, 24, 1, 0, 17, 0, 21, 0, 25, 0,
15, 11, 0, 0, 0, 0, 22, 13, 0, 0, 10, 0, 0, 0}
  Dim Letter = {"SPACE", "A", "B", "C", "D", "E", "F",
"G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q",
"R", "S", "T", "U", "V", "W", "X", "Y", "Z"}
  Dim Dot = {5, 18, 0, 0, 2, 9, 0, 26, 0, 19, 0, 3, 0, 7,
4, 0, 0, 0, 12, 8, 14, 6, 0, 16, 0, 0, 0}
  Dim MorseCode = {"SPACE", ".-", "-...", "-.-.", "-..",
".", "..-.", "--.", "....", "..", ".---", "-.-", ".-..",
"--", "-.", "---", ".--.", "--.-", ".-.", "...", "-", "..-
", "...-", ".--", "-..-", "-.--", "--.."}
  Dim MenuOption As String
  Dim ProgramEnd As Boolean = False
  Dim ValidDisplacement As Boolean
  Dim Key1 As Integer
  Dim Key2 As Integer
  Dim Key3 As Integer
  ValidDisplacement = False
  While Not ValidDisplacement
    Try
      Console.Write("Enter encryption key (integer): ")
      Key1 = Console.ReadLine
      ValidDisplacement = True
    Catch ex As Exception
    End Try
  End While
  ValidDisplacement = False
  While Not ValidDisplacement
    Try
      Console.Write("Enter encryption key (integer): ")
      Key2 = Console.ReadLine
      ValidDisplacement = True
    Catch ex As Exception
    End Try
  End While
  ValidDisplacement = False
  While Not ValidDisplacement
    Try
      Console.Write("Enter encryption key (integer): ")
      Key3 = Console.ReadLine
      ValidDisplacement = True
    Catch ex As Exception
    End Try
  End While
  While Not ProgramEnd
```

```
    DisplayMenu()
    MenuOption = GetMenuOption()
    If MenuOption = "R" Then
      ReceiveMorseCode(Dash, Letter, Dot)
    ElseIf MenuOption = "S" Then
      SendMorseCode(MorseCode, Key1, Key2, Key3)
    ElseIf MenuOption = "A" Then
      OutputAlphabetWithCode(Letter, MorseCode)
    ElseIf MenuOption = "X" Then
      ProgramEnd = True
    End If
  End While
End Sub
```

Python 2

03	1	`Number = 0`
		`while Number < 1 or Number > 10:`
		`  Number = int(raw_input("Enter a positive whole number: "))`
		`  if Number > 10:`
		`  print "Number too large"`
		`  elif Number < 1:`
		`  print "Not a positive number"`
		`c = 1`
		`for k in range(Number):`
		`  print c`
		`  c = (c * (Number - 1 - k)) // (k + 1)`
			9

11	1	def SendMorseCode(MorseCode):
		  PlainText = raw_input("Enter your message (uppercase letters and spaces only): ")
		  PlainTextLength = len(PlainText)
		  MorseCodeString = EMPTYSTRING
		  for i in range(PlainTextLength):
		  PlainTextLetter = PlainText[i]
		  if PlainTextLetter == SPACE:
		  Index = 0
		  el**if PlainTextLetter >= 'A' and PlainTextLetter <= 'Z':**
		  Index = ord(PlainTextLetter) – ord('A') + 1
		  else:
		  ReportError("Invalid character entered")
		  Index = 0
		  MorseCodeString = EMPTYSTRING
		  break
		  CodedLetter = MorseCode[Index]
		  MorseCodeString = MorseCodeString + CodedLetter + SPACE
		  print MorseCodeString
		
		Alternative answer:
		def SendMorseCode(MorseCode):
		  PlainText = raw_input("Enter your message (uppercase letters and spaces only): ")
		  Valid = True
		  for Character in PlainText:
		  if Character != SPACE:
		  if Character < "A" or Character > "Z":
		  Valid = False
		  MorseCodeString = EMPTYSTRING
		  ReportError("Invalid character entered")
		  break
		  if Valid:
		  PlainTextLength = len(PlainText)
		  MorseCodeString = EMPTYSTRING
		  for i in range(PlainTextLength):
		  PlainTextLetter = PlainText[i]
		  if PlainTextLetter == SPACE:
		  Index = 0
		  else:
		  Index = ord(PlainTextLetter) - ord('A') + 1
		  CodedLetter = MorseCode[Index]
		  MorseCodeString = MorseCodeString + CodedLetter + SPACE
		  print MorseCodeString
			4

12	1	def SendSignals(MorseCodeString):
		  Transmission = EMPTYSTRING
		  CodeStringLength = len(MorseCodeString)
		  for i in range(CodeStringLength):
		  Symbol = MorseCodeString[i]
		  if Symbol == '.':
		  SymbolString = "= "
		  elif Symbol == '-':
		  SymbolString = "=== "
		  elif Symbol == SPACE:
		  SymbolString = SPACE + SPACE
		  Transmission = Transmission + SymbolString
		  print Transmission
			7

13	1	**def OutputAlphabetWithCode(Letter, MorseCode):**
		** for Ptr in range(1, 27):**
		** print Letter[Ptr], end=SPACE**
		** print '{0:<5}'.format(MorseCode[Ptr]), end=SPACE**
		** if Ptr % 4 == 0:**
		** print**
		** print**
		
		def DisplayMenu():
		  print
		  print "Main Menu"
		  print "========="
		  print "R - Receive Morse code"
		  print "S - Send Morse code"
		  print "A - Output alphabet with Morse code"
		  print "X - Exit program"
		  print
		
		def SendReceiveMessages():
		Dash =
		[20,23,0,0,24,1,0,17,0,21,0,25,0,15,11,0,0,0,0,22,13,0,0,10,0,0,0]
		  Dot =
		[5,18,0,0,2,9,0,26,0,19,0,3,0,7,4,0,0,0,12,8,14,6,0,16,0,0,0]
		  Letter =
		["SPACE",'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']
		
		  MorseCode = ["SPACE",'.-','-...','-.-.','-..','.','..-.','--.','....','..','.---','-.-','.-..','--','-.','---','.--.','--.-','.-.','...','-','..-','...-','.--','-..-','-.--','--..']
		  ProgramEnd = False
		  while not ProgramEnd:
		  DisplayMenu()
		  MenuOption = GetMenuOption()
		  if MenuOption == 'R':
		  ReceiveMorseCode(Dash, Letter, Dot)
		  elif MenuOption == 'S':
		  SendMorseCode(MorseCode)
		  elif MenuOption == 'A':
		  OutputAlphabetWithCode(Letter, MorseCode)
		  elif MenuOption == 'X':
		  ProgramEnd = True
			6

14	1	def SendMorseCode(MorseCode, **Keys**):
		  PlainText = raw_input("Enter your message (uppercase letters and spaces only): ")
		  PlainTextLength = len(PlainText)
		  MorseCodeString = EMPTYSTRING
		  for i in range(PlainTextLength):
		  PlainTextLetter = PlainText[i]
		  if PlainTextLetter == SPACE:
		  Index = 0
		  else:
		  Index = ord(PlainTextLetter) - ord('A') + 1
		  Index = Index + Keys[i % 3]
		  while Index < 0:
		  Index = Index + 27
		  while Index >= 27:
		  Index = Index - 27
		  CodedLetter = MorseCode[Index]
		  MorseCodeString = MorseCodeString + CodedLetter + SPACE
		  print(MorseCodeString)
		
		def SendReceiveMessages():
		Dash = [20,23,0,0,24,1,0,17,0,21,0,25,0,15,11,0,0,0,0,22,13,0,0,10,0,0,0]
		  Dot = [5,18,0,0,2,9,0,26,0,19,0,3,0,7,4,0,0,0,12,8,14,6,0,16,0,0,0]
		  Letter = ["SPACE",'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']
		
		  MorseCode = ["SPACE",'.-','-...','-.-.','-..','.','..-.','--.','....','..','.---','-.-','.-..','--','-.','---','.--.','--.-','.-.','...','-','..-','...-','.--','-..-','-.--','--..']
		  ProgramEnd = False
		
		  Keys = [0,0,0]
		  for i in range(3):
		  ValidDisplacement = False
		  while not ValidDisplacement:
		  try:
		  Displacement = int(raw_input("Enter encryption key (integer): "))
		  ValidDisplacement = True
		  except:
		  pass
		  Keys[i] = Displacement
		
		  while not ProgramEnd:
		  DisplayMenu()
		  MenuOption = GetMenuOption()
		  if MenuOption == 'R':
		  ReceiveMorseCode(Dash, Letter, Dot)
			9

```
    elif MenuOption == 'S':
      SendMorseCode(MorseCode, Keys)
    elif MenuOption == 'X':
      ProgramEnd = True
```

Alternative answer:

```
def SendMorseCode(MorseCode, Key1, Key2, Key3):
  PlainText = input("Enter your message (uppercase letters and
spaces only): ")
  PlainTextLength = len(PlainText)
  MorseCodeString = EMPTYSTRING
  for i in range(PlainTextLength):
    PlainTextLetter = PlainText[i]
    if PlainTextLetter == SPACE:
      Index = 0
    else:
      Index = ord(PlainTextLetter) - ord('A') + 1
    if i % 3 == 0:
      Displacement = Key1
    elif i % 3 == 1:
      Displacement = Key2
    else:
      Displacement = Key3
    Index = Index + Displacement
    while Index < 0:
      Index = Index + 27
    while Index >= 27:
      Index = Index - 27
    CodedLetter = MorseCode[Index]
    MorseCodeString = MorseCodeString + CodedLetter + SPACE
  print(MorseCodeString)

def SendReceiveMessages():
Dash =
[20,23,0,0,24,1,0,17,0,21,0,25,0,15,11,0,0,0,0,22,13,0,0,10,0,0,0]
  Dot =
[5,18,0,0,2,9,0,26,0,19,0,3,0,7,4,0,0,0,12,8,14,6,0,16,0,0,0]
  Letter =
["SPACE",'A','B','C','D','E','F','G','H','I','J','K','L','M','N','
O','P','Q','R','S','T','U','V','W','X','Y','Z']

  MorseCode = ["SPACE",'.-','-...','-.-.','-..','.','..-.','--
.','....','..','.---','-.-','.-..','--','-.','---','.--.','--.-
','.-.','...','-','..-','...-','.--','-..-','-.--','--..']
  ProgramEnd = False

  ValidDisplacement = False
  while not ValidDisplacement:
    try:
      Key1 = int(input("Enter encryption key (integer): "))
      ValidDisplacement = True
```

```
    except:
      pass
ValidDisplacement = False
while not ValidDisplacement:
  try:
    Key2 = int(input("Enter encryption key (integer): "))
    ValidDisplacement = True
  except:
    pass
ValidDisplacement = False
while not ValidDisplacement:
  try:
    Key3 = int(input("Enter encryption key (integer): "))
    ValidDisplacement = True
  except:
    pass

while not ProgramEnd:
  DisplayMenu()
  MenuOption = GetMenuOption()
  if MenuOption == 'R':
    ReceiveMorseCode(Dash, Letter, Dot)
  elif MenuOption == 'S':
    SendMorseCode(MorseCode, Key1, Key2, Key3)
  elif MenuOption == 'X':
    ProgramEnd = True
```

Python 3

03	**1**	`Number = 0`
		`while Number < 1 or Number > 10:`
		`  Number = int(input("Enter a positive whole number: "))`
		`  if Number > 10:`
		`  print("Number too large")`
		`  elif Number < 1:`
		`  print("Not a positive number")`
		`c = 1`
		`for k in range(Number):`
		`  print(c)`
		`  c = (c * (Number - 1 - k)) // (k + 1)`
			9

11 **1**

```
def SendMorseCode(MorseCode):
  PlainText = input("Enter your message (uppercase letters and
spaces only): ")
  PlainTextLength = len(PlainText)
  MorseCodeString = EMPTYSTRING
  for i in range(PlainTextLength):
    PlainTextLetter = PlainText[i]
    if PlainTextLetter == SPACE:
      Index = 0
    elif PlainTextLetter >= 'A' and PlainTextLetter <= 'Z':
      Index = ord(PlainTextLetter) - ord('A') + 1
    else:
      ReportError("Invalid character entered")
      Index = 0
      MorseCodeString = EMPTYSTRING
      break
    CodedLetter = MorseCode[Index]
    MorseCodeString = MorseCodeString + CodedLetter + SPACE
  print(MorseCodeString)
```

Alternative answer:

```
def SendMorseCode(MorseCode):
  PlainText = input("Enter your message (uppercase letters and
spaces only): ")
  Valid = True
  for Character in PlainText:
    if Character != SPACE:
      if Character < "A" or Character > "Z":
        Valid = False
        MorseCodeString = EMPTYSTRING
        ReportError("Invalid character entered")
        break
  if Valid:
    PlainTextLength = len(PlainText)
    MorseCodeString = EMPTYSTRING
    for i in range(PlainTextLength):
      PlainTextLetter = PlainText[i]
      if PlainTextLetter == SPACE:
        Index = 0
      else:
        Index = ord(PlainTextLetter) - ord('A') + 1
      CodedLetter = MorseCode[Index]
      MorseCodeString = MorseCodeString + CodedLetter + SPACE
  print(MorseCodeString)
```

4

12	**1**	`def SendSignals(MorseCodeString):`
		`  Transmission = EMPTYSTRING`
		`  CodeStringLength = len(MorseCodeString)`
		`  for i in range(CodeStringLength):`
		`  Symbol = MorseCodeString[i]`
		`  if Symbol == '.':`
		`  SymbolString = "= "`
		`  elif Symbol == '-':`
		`  SymbolString = "=== "`
		`  elif Symbol == SPACE:`
		`  SymbolString = SPACE + SPACE`
		`  Transmission = Transmission + SymbolString`
		`  print(Transmission)`
			7

13 1

```
def OutputAlphabetWithCode(Letter, MorseCode):
  for Ptr in range(1, 27):
    print(Letter[Ptr], end=SPACE)
    print('{0:<5}'.format(MorseCode[Ptr]), end=SPACE)
    if Ptr % 4 == 0:
      print()
  print()

def DisplayMenu():
  print()
  print("Main Menu")
  print("=========")
  print("R - Receive Morse code")
  print("S - Send Morse code")
  print("A - Output alphabet with Morse code")
  print("X - Exit program")
  print()

def SendReceiveMessages():
Dash = [20,23,0,0,24,1,0,17,0,21,0,25,0,15,11,0,0,0,0,22,13,0,0,10,0,0,0]
  Dot = [5,18,0,0,2,9,0,26,0,19,0,3,0,7,4,0,0,0,12,8,14,6,0,16,0,0,0]
  Letter =
["SPACE",'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q

  MorseCode = ["SPACE",'.-','-...','-.-.','-..','.','..-.','--.','....','..
',','.--.','--.-','.-.','...','-','..-','...-','.--','-..-','-.--','--..']
  ProgramEnd = False
  while not ProgramEnd:
    DisplayMenu()
    MenuOption = GetMenuOption()
    if MenuOption == 'R':
      ReceiveMorseCode(Dash, Letter, Dot)
    elif MenuOption == 'S':
      SendMorseCode(MorseCode)
    elif MenuOption == 'A':
      OutputAlphabetWithCode(Letter, MorseCode)
    elif MenuOption == 'X':
      ProgramEnd = True
```

14 **1**

```
def SendMorseCode(MorseCode, Keys):
  PlainText = input("Enter your message (uppercase letters and spaces only)
  PlainTextLength = len(PlainText)
  MorseCodeString = EMPTYSTRING
  for i in range(PlainTextLength):
    PlainTextLetter = PlainText[i]
    if PlainTextLetter == SPACE:
      Index = 0
    else:
      Index = ord(PlainTextLetter) - ord('A') + 1
    Index = Index + Keys[i % 3]
    while Index < 0:
      Index = Index + 27
    while Index >= 27:
      Index = Index - 27
    CodedLetter = MorseCode[Index]
    MorseCodeString = MorseCodeString + CodedLetter + SPACE
  print(MorseCodeString)

def SendReceiveMessages():
Dash = [20,23,0,0,24,1,0,17,0,21,0,25,0,15,11,0,0,0,0,22,13,0,0,10,0,0,0]
  Dot = [5,18,0,0,2,9,0,26,0,19,0,3,0,7,4,0,0,0,12,8,14,6,0,16,0,0,0]
  Letter =
["SPACE",'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q

  MorseCode = ["SPACE",'.-','-...','-.-.','-..','.','..-.','--.','....','..
',','.--.','--.-','.-.','...','-','..-','...-','.--','-..-','-.--','--..']
  ProgramEnd = False

  Keys = [0,0,0]
  for i in range(3):
    ValidDisplacement = False
    while not ValidDisplacement:
      try:
        Displacement = int(input("Enter encryption key (integer): "))
        ValidDisplacement = True
      except:
        pass
    Keys[i] = Displacement

  while not ProgramEnd:
    DisplayMenu()
    MenuOption = GetMenuOption()
    if MenuOption == 'R':
      ReceiveMorseCode(Dash, Letter, Dot)
    elif MenuOption == 'S':
      SendMorseCode(MorseCode, Keys)
    elif MenuOption == 'X':
      ProgramEnd = True
```

Alternative answer:

```
def SendMorseCode(MorseCode, Key1, Key2, Key3):
  PlainText = input("Enter your message (uppercase letters and spaces only)
  PlainTextLength = len(PlainText)
  MorseCodeString = EMPTYSTRING
  for i in range(PlainTextLength):
    PlainTextLetter = PlainText[i]
    if PlainTextLetter == SPACE:
      Index = 0
    else:
      Index = ord(PlainTextLetter) - ord('A') + 1
    if i % 3 == 0:
      Displacement = Key1
    elif i % 3 == 1:
      Displacement = Key2
    else:
      Displacement = Key3
    Index = Index + Displacement
    while Index < 0:
      Index = Index + 27
    while Index >= 27:
      Index = Index - 27
    CodedLetter = MorseCode[Index]
    MorseCodeString = MorseCodeString + CodedLetter + SPACE
  print(MorseCodeString)

def SendReceiveMessages():
Dash = [20,23,0,0,24,1,0,17,0,21,0,25,0,15,11,0,0,0,0,22,13,0,0,10,0,0,0]
  Dot = [5,18,0,0,2,9,0,26,0,19,0,3,0,7,4,0,0,0,12,8,14,6,0,16,0,0,0]
  Letter =
["SPACE",'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q

  MorseCode = ["SPACE",'.-','-...','-.-.','-..','.','..-.','--.','....','..
','.--.','--.-','.-.','...','-','..-','...-','.--','-..-','-.--','--..']
  ProgramEnd = False

  ValidDisplacement = False
  while not ValidDisplacement:
    try:
      Key1 = int(input("Enter encryption key (integer): "))
      ValidDisplacement = True
    except:
      pass
  ValidDisplacement = False
  while not ValidDisplacement:
    try:
      Key2 = int(input("Enter encryption key (integer): "))
      ValidDisplacement = True
    except:
      pass
  ValidDisplacement = False
```

```
while not ValidDisplacement:
  try:
    Key3 = int(input("Enter encryption key (integer): "))
    ValidDisplacement = True
  except:
    pass

while not ProgramEnd:
  DisplayMenu()
  MenuOption = GetMenuOption()
  if MenuOption == 'R':
    ReceiveMorseCode(Dash, Letter, Dot)
  elif MenuOption == 'S':
    SendMorseCode(MorseCode, Key1, Key2, Key3)
  elif MenuOption == 'X':
    ProgramEnd = True
```

Pascal

03	1	`var`
		`  Number, c, k : Integer;`
		`begin`
		`  Number := 0;`
		`  while (Number < 1) or (Number > 10) do`
		`  begin`
		`  write('Enter a positive whole number: ');`
		`  readln(Number);`
		`  if Number > 10 then`
		`  writeln('Number too large')`
		`  else`
		`  if Number < 1 then`
		`  writeln('Not a positive number.')`
		`  end;`
		`  c := 1;`
		`  for k := 0 to (Number - 1) do`
		`  begin`
		`  writeln(c);`
		`  c := (c * (Number - 1 - k)) div (k + 1);`
		`  end;`
		`  readln;`
		`end.`
			9

11	1	Procedure SendMorseCode(MorseCode : Array of String);
		var
		  PlainText, MorseCodeString, CodedLetter : String;
		  PlainTextLength, i, Index : Integer;
		  PlainTextLetter : Char;
		begin
		  write('Enter your message (uppercase letters and spaces only): ');
		  readln(PlainText);
		  PlainTextLength := length(PlainText);
		  MorseCodeString := EMPTYSTRING;
		  for i := 1 to PlainTextLength do
		  begin
		  PlainTextLetter := PlainText[i];
		  if PlainTextLetter = SPACE then
		  Index := 0
		  else
		  if (PlainTextLetter >= 'A') and (PlainTextLetter <= 'Z') then
		  Index := ord(PlainTextLetter) - ord('A') + 1
		  else
		  begin
		  ReportError('Invalid character entered');
		  Index := 0;
		  MorseCodeString := EMPTYSTRING;
		  break;
		  end;
		  CodedLetter := MorseCode[Index];
		  MorseCodeString := MorseCodeString + CodedLetter + SPACE;
		  end;
		  writeln(MorseCodeString);
		end;
			4

12	1	**Procedure SendSignals(MorseCodeString : String);**
		var
		** Transmission, SymbolString : String;**
		** CodeStringLength, i : Integer;**
		** Symbol : Char;**
		begin
		** Transmission := EMPTYSTRING;**
		** CodeStringLength := length(MorseCodeString);**
		** for i := 1 to CodeStringLength do**
		** begin**
		** Symbol := MorseCodeString[i];**
		** if Symbol = '.' then**
		** SymbolString := '= '**
		** else**
		** if Symbol = '-' then**
		** SymbolString := '=== '**
		** else**
		** if Symbol = SPACE then**
		** SymbolString := SPACE + SPACE;**
		** Transmission := Transmission + SymbolString;**
		** end;**
		** writeln(Transmission);**
		end;
			7

13	1	**Procedure OutputAlphabetWithCode(Letter : Array of Char;**
		MorseCode : Array of String);
		var
		** Index : Integer;**
		begin
		** for Index := 1 to 26 do**
		** begin**
		** write(Letter[Index], SPACE);**
		** write(Format('%0:-6s',[MorseCode[Index]]));**
		** if index mod 4 = 0 then**
		** writeln;**
		** end;**
		** writeln;**
		end;
		Procedure DisplayMenu();
		begin
		  writeln;
		  writeln('Main Menu');
		  writeln('=========');
		  writeln('R - Receive Morse code');
		  writeln('S = Send Morse code');
		  writeln('A - Output alphabet with Morse code');
		  writeln('X - Exit program');
		  writeln;
		end;
			6

14	1	Procedure SendMorseCode(MorseCode : TStringArray; **Keys : Array of Integer**);
		var
		  PlainText, MorseCodeString, CodedLetter : String;
		  PlainTextLength, i, Index, **Displacement** : Integer;
		  PlainTextLetter : Char;
		begin
		  write('Enter your message (uppercase letters and spaces only): ');
		  readln(PlainText);
		  PlainTextLength := length(PlainText);
		  MorseCodeString := EMPTYSTRING;
		  for i := 1 to PlainTextLength do
		  begin
		  PlainTextLetter := PlainText[i];
		  if PlainTextLetter = SPACE then
		  Index := 0
		  else
		
		Index := ord(PlainTextLetter) - ord('A') + 1
		  Displacement := Keys[(i-1) mod 3];
		  Index := Index + Displacement;	

```
      while Index <0 do
        Index := Index + 27;
      while Index >= 27 do
        Index := Index - 27;
      CodedLetter := MorseCode[Index];
      MorseCodeString := MorseCodeString + CodedLetter + SPACE;
    end;
  writeln(MorseCodeString);
end;


Procedure SendReceiveMessages();
var
  Dash: array[0..26] of Integer =
(20,23,0,0,24,1,0,17,0,21,0,25,0,15,11,0,0,0,0,22,13,0,0,10,0,0,
0);
  Dot : array[0..26] of Integer =
(5,18,0,0,2,9,0,26,0,19,0,3,0,7,4,0,0,0,12,8,14,6,0,16,0,0,0);
  Letter : array[0..26] of Char = ('
','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P
','Q','R','S','T','U','V','W','X','Y','Z');
  MorseCode : array[0..26] of String = (' ','.-','-...','-.-
.','-..','.','..-.','--.','....','..','.---','-.-','.-..','--
','-.','---','.--.','--.-','.-.','...','-','..-','...-','.--','-
..-','-.--','--..');
  ProgramEnd : Boolean;
  MenuOption, Input : String;
  Keys : array [0..2] of Integer = (0,0,0);
  i, Error : Integer;
begin
    ProgramEnd := False;
  for i := 0 to 2 do
    begin
      write('Enter encryption key (integer): ');
      readln(Input);
      val(Input,Keys[i],Error);
      while Error <> 0 do
        begin
          write('Error - invalid input - please re-enter: ');
          readln(Input);
          val(Input,Keys[i],Error);
        end;
    end;

  while not(ProgramEnd) do
    begin
      DisplayMenu();
      MenuOption := GetMenuOption();
      if MenuOption = 'R' then
        ReceiveMorseCode(Dash, Letter, Dot)
      else if MenuOption = 'S' then
        SendMorseCode(MorseCode, Keys)
```

```
      else if MenuOption = 'X' then
        ProgramEnd := True;
    end;
end;
```

C#

03	1	**static void Main(string[] args)** **{** **int Number = 0;** **while (Number < 1 || Number > 10)** **{** **Console.WriteLine("Enter a positive whole number: ");** **Number = Convert.ToInt32(Console.ReadLine());** **if (Number > 10)** **{** **Console.WriteLine("Number too large.");** **}** **else** **{** **if (Number < 1)** **{** **Console.WriteLine("Not a positive number.");** **}** **}** **}** **int c = 1;** **for (int k = 0; k < Number ; k++)** **{** **Console.WriteLine(c);** **c = (c * (Number - 1 - k)) / (k + 1);** **}** **Console.ReadLine();** **}**	9

11	1	private static void SendMorseCode(string[] MorseCode)
		{
		  Console.Write("Enter your message (uppercase letters and spaces only): ");
		  string PlainText = Console.ReadLine();
		  int PlainTextLength = PlainText.Length;
		  string MorseCodeString = EMPTYSTRING;
		  char PlainTextLetter = SPACE;
		  int Index = 0;
		  for (int i = 0; i < PlainTextLength; i++)
		  {
		  PlainTextLetter = PlainText[i];
		  if (PlainTextLetter == SPACE)
		  {
		  Index = 0;
		  }
		  else **if (PlainTextLetter >= 'A' && PlainTextLetter <= 'Z')**
		  {
		  Index = (int)PlainTextLetter - (int)'A' + 1;
		  }
		  else
		  {
		  ReportError("Invalid character entered");
		  Index = 0;
		  MorseCodeString = EMPTYSTRING;
		  break;
		  }
		  string CodedLetter = MorseCode[Index];
		  MorseCodeString = MorseCodeString + CodedLetter + SPACE;
		  }
		  Console.WriteLine(MorseCodeString);
		}
		Alternative answer
		  private static void SendMorseCode(string[] MorseCode)
		  {
		  bool Valid = true;
		  Console.Write("Enter your message (uppercase letters and spaces only): ");
		  string PlainText = Console.ReadLine();
		  int PlainTextLength = PlainText.Length;
		  string MorseCodeString = EMPTYSTRING;
		  char PlainTextLetter = ' ';
		  int Index = 0;
		  foreach (char Character in PlainText)
		  {
		  if (Character != SPACE)
		  {
		  if (Character < 'A' || Character > 'Z')
		  {
		  Valid = false;
		  MorseCodeString = EMPTYSTRING;
			4

```
            ReportError("Invalid character entered");
            break;
          }
        }
      }
      if (Valid)
      {
        for (int i = 0; i < PlainTextLength; i++)
        {
          PlainTextLetter = PlainText[i];
          if (PlainTextLetter == SPACE)
          {
            Index = 0;
          }
          else
          {
            Index = (int)PlainTextLetter - (int)'A' + 1;
          }
          string CodedLetter = MorseCode[Index];
          MorseCodeString = MorseCodeString + CodedLetter +
SPACE;
        }
      }
      Console.WriteLine(MorseCodeString);
    }
```

12	1	`private static void SendSignals(string MorseCodeString)`
		`{`
		`  string Transmission = EMPTYSTRING;`
		`  char Symbol;`
		`  string SymbolString = "";`
		`  int CodeStringLength = MorseCodeString.Length;`
		`  for (int i = 0; i < CodeStringLength; i++)`
		`  {`
		`  Symbol = MorseCodeString[i];`
		`  if (Symbol == '.')`
		`  {`
		`  SymbolString = "= ";`
		`  }`
		`  else if (Symbol == '-')`
		`  {`
		`  SymbolString = "=== ";`
		`  }`
		`  if (Symbol == SPACE)`
		`  {`
		`  SymbolString = SPACE.ToString() + SPACE.ToString();`
		`  }`
		`  Transmission = Transmission + SymbolString;`
		`  }`
		`  Console.WriteLine(Transmission);`
		`}`
			7

13 1 6

```
private static void OutputAlphabetWithCode(char[] Letter,
string[] MorseCode)
{
  for (int Ptr = 1; Ptr < 27; Ptr++)
  {
    Console.Write(Letter[Ptr]);
    Console.Write(SPACE);
    Console.Write("{0,-6}",MorseCode[Ptr]);
    if (Ptr % 4 == 0)
    {
      Console.WriteLine();
    }
  }
}
private static void SendReceiveMessages()
{
  int[] Dash = new int[] { 20, 23, 0, 0, 24, 1, 0, 17, 0, 21, 0,
25, 0, 15, 11, 0, 0, 0, 0, 22, 13, 0, 0, 10, 0, 0, 0 };
  int[] Dot = new int[] { 5, 18, 0, 0, 2, 9, 0, 26, 0, 19, 0, 3,
0, 7, 4, 0, 0, 0, 12, 8, 14, 6, 0, 16, 0, 0, 0 };
  char[] Letter = new char[] { "SPACE", 'A', 'B', 'C', 'D', 'E',
'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R',
'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z' };
  string[] MorseCode = new string[] { " ", ".-", "-...", "-.-.",
"-..", ".", "..-.", "--.", "....", "..", ".---", "-.-", ".-..",
"--", "-.", "---", ".--.", "--.-", ".-.", "...", "-", "..-",
"...-", ".--", "-..-", "-.--", "--.." };
  bool ProgramEnd = false;
  string MenuOption = EMPTYSTRING;
  while (!ProgramEnd)
  {
    DisplayMenu();
    GetMenuOption(ref MenuOption);
    if (MenuOption == "R")
    {
      ReceiveMorseCode(Dash, Letter, Dot);
    }
    else if (MenuOption == "S")
    {
      SendMorseCode(MorseCode);
    }
    else if (MenuOption == "A")
    {
      OutputAlphabetWithCode(Letter, MorseCode);
    }
    else if (MenuOption == "X")
    {
      ProgramEnd = true;
    }
  }
}
```

```
private static void DisplayMenu()
{
  Console.WriteLine();
  Console.WriteLine("Main Menu");
  Console.WriteLine("=========");
  Console.WriteLine("R - Receive Morse code");
  Console.WriteLine("S - Send Morse code");
  Console.WriteLine("A - Output alphabet with Morse code");
  Console.WriteLine("X - Exit program");
  Console.WriteLine();
}
```

14	1	private static void SendMorseCode(string[] MorseCode, **int[] Keys**)
		{
		  Console.Write("Enter your message (uppercase letters and spaces only): ");
		  string PlainText = Console.ReadLine();
		  int PlainTextLength = PlainText.Length;
		  string MorseCodeString = EMPTYSTRING;
		  char PlainTextLetter = ' ';
		  int Index = 0;
		  for (int i = 0; i < PlainTextLength; i++)
		  {
		  PlainTextLetter = PlainText[i];
		  if (PlainTextLetter == SPACE)
		  {
		  Index = 0;
		  }
		  else
		  {
		  Index = (int)PlainTextLetter - (int)'A' + 1;
		  }
		  Index = Index + Keys[i % 3];
		  while (Index < 0)
		  {
		  Index = Index + 27;
		  }
		  while (Index >=27)
		  {
		  Index = Index - 27;
		  }
		  string CodedLetter = MorseCode[Index];
		  MorseCodeString = MorseCodeString + CodedLetter + SPACE;
		  }
		  Console.WriteLine(MorseCodeString);
		}
		private static void SendReceiveMessages()
		{
		  int[] Dash = new int[] { 20, 23, 0, 0, 24, 1, 0, 17, 0, 21, 0, 25, 0, 15, 11, 0, 0, 0, 0, 22, 13, 0, 0, 10, 0, 0, 0 };
		  int[] Dot = new int[] { 5, 18, 0, 0, 2, 9, 0, 26, 0, 19, 0, 3, 0, 7, 4, 0, 0, 0, 12, 8, 14, 6, 0, 16, 0, 0, 0 };
		  char[] Letter = new char[] { "SPACE", 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z' };
		  string[] MorseCode = new string[] { " ", ".-", "-...", "-.-.", "-..", ".", "..-.", "--.", "....", "..", ".---", "-.-", ".-..", "--", "-.", "---", ".--.", "--.-", ".-.", "...", "-", "..-", "...-", ".--", "-..-", "-.--", "--.." };
		  bool ProgramEnd = false;
		  string MenuOption = EMPTYSTRING;
		  int[] Keys = new int[3];
			9

```
  int Displacement = 0;
  for (int i = 0; i < 3; i++)
  {
    bool ValidDisplacement = false;
    while (!ValidDisplacement)
    {
      Console.Write("Enter encryption key (integer): ");
      try
      {
        Displacement = Convert.ToInt32(Console.ReadLine());
        ValidDisplacement = true;
      }
      catch (Exception)
      {
      }
    }
    Keys[i] = Displacement;
  }
  while (!ProgramEnd)
  {
    DisplayMenu();
    GetMenuOption(ref MenuOption);
    if (MenuOption == "R")
    {
      ReceiveMorseCode(Dash, Letter, Dot);
    }
    else if (MenuOption == "S")
    {
      SendMorseCode(MorseCode, Keys);
    }
    else if (MenuOption == "X")
    {
      ProgramEnd = true;
    }
  }
}
```

Alternative Answer:

```
private static void SendMorseCode(string[] MorseCode, int Key1,
int Key2, int Key3)
{
  Console.Write("Enter your message (uppercase letters and
spaces only): ");
  string PlainText = Console.ReadLine();
  int PlainTextLength = PlainText.Length;
  string MorseCodeString = EMPTYSTRING;
  char PlainTextLetter = ' ';
  int Index = 0;
  int Displacement = 0;
  for (int i = 0; i < PlainTextLength; i++)
```

```
  {
    PlainTextLetter = PlainText[i];
    if (PlainTextLetter == SPACE)
    {
      Index = 0;
    }
    else
    {
      Index = (int)PlainTextLetter - (int)'A' + 1;
    }
    if (i % 3 == 0)
    {
      Displacement = Key1;
    }
    else if (i % 3 == 1)
    {
      Displacement = Key2;
    }
    else
    {
      Displacement = Key3;
    }
    Index = Index + Displacement;
    while (Index < 0)
    {
      Index = Index + 27;
    }
    while (Index >= 27)
    {
      Index = Index - 27;
    }
    string CodedLetter = MorseCode[Index];
    MorseCodeString = MorseCodeString + CodedLetter + SPACE;
  }
  Console.WriteLine(MorseCodeString);
}

private static void SendReceiveMessages()
{
  int[] Dash = new int[] { 20, 23, 0, 0, 24, 1, 0, 17, 0, 21, 0,
25, 0, 15, 11, 0, 0, 0, 0, 22, 13, 0, 0, 10, 0, 0, 0 };
  int[] Dot = new int[] { 5, 18, 0, 0, 2, 9, 0, 26, 0, 19, 0, 3,
0, 7, 4, 0, 0, 0, 12, 8, 14, 6, 0, 16, 0, 0, 0 };
  char[] Letter = new char[] { "SPACE", 'A', 'B', 'C', 'D', 'E',
'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R',
'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z' };
  string[] MorseCode = new string[] { " ", ".-", "-...", "-.-.",
"-..", ".", "..-.", "--.", "....", "..", ".---", "-.-", ".-..",
"--", "-.", "---", ".--.", "--.-", ".-.", "...", "-", "..-",
"...-", ".--", "-..-", "-.--", "--.." };
  bool ProgramEnd = false;
```

```
string MenuOption = EMPTYSTRING;
int Key1 = 0, Key2 = 0, Key3 = 0;
bool ValidDisplacement = false;
while (!ValidDisplacement)
{
  Console.Write("Enter encryption key (integer): ");
  try
  {
    Key1 = Convert.ToInt32(Console.ReadLine());
    ValidDisplacement = true;
  }
  catch (Exception)
  {
  }
}
ValidDisplacement = false;
while (!ValidDisplacement)
{
  Console.Write("Enter encryption key (integer): ");
  try
  {
    Key2 = Convert.ToInt32(Console.ReadLine());
    ValidDisplacement = true;
  }
  catch (Exception)
  {
  }
}
ValidDisplacement = false;
while (!ValidDisplacement)
{
  Console.Write("Enter encryption key (integer): ");
  try
  {
    Key3 = Convert.ToInt32(Console.ReadLine());
    ValidDisplacement = true;
  }
  catch (Exception)
  {
  }
}
while (!ProgramEnd)
{
  DisplayMenu();
  GetMenuOption(ref MenuOption);
  if (MenuOption == "R")
  {
    ReceiveMorseCode(Dash, Letter, Dot);
  }
  else if (MenuOption == "S")
  {
```

```
        SendMorseCode(MorseCode, Key1, Key2, Key3);
      }
      else if (MenuOption == "X")
      {
        ProgramEnd = true;
      }
    }
}
```

Java

03	1	`int number = 0;`
		`while (number < 1 || number > 10)`
		`{`
		`  Console.writeLine(("Enter a positive whole number: "));`
		`  number = Integer.parseInt(Console.readLine());`
		`  if (number > 10)`
		`  {`
		`  Console.writeLine("Number too large");`
		`  }`
		`  else if (number < 1)`
		`  {`
		`  Console.writeLine("Not a positive number");`
		`  }`
		`}`
		`int c = 1;`
		`for (int k = 0; k < number; k++)`
		`{`
		`  Console.write(c + " ");`
		`  c = (c * (number - 1 - k) / (k + 1));`
		`}`
			9

11	1	`  void sendMorseCode(String[] morseCode)`
		`  {`
		`    Console.write("Enter your message (uppercase letters and spaces only): ");`
		`    String plainText = Console.readLine();`
		`    int plainTextLength = plainText.length();`
		`    String morseCodeString = EMPTYSTRING;`
		`    int index;`
		`    for (int i = 0; i < plainTextLength; i++)`
		`    {`
		`      char plainTextLetter = plainText.charAt(i);`
		`      if (plainTextLetter == SPACE)`
		`      {`
		`        index = 0;`
		`      }`
		`      else`
		`      {`
		`        index = (int)plainTextletter - (int)'A' + 1;`
		`      }`
		`      **if (index >= 0 && index <= 26)**`
		`      **{**`
		`        String codedLetter = morseCode[index];`
		`        morseCodeString = morseCodeString + codedLetter + SPACE;`
		`      **}**`
		`      **else**`
		`      **{**`
		`        **morseCodeString = EMPTYSTRING;**`
		`        **break;**`
		`      **}**`
		`    }`
		`    **if(morseCodeString != EMPTYSTRING)**`
		`    **{**`
		`      Console.writeLine(morseCodeString);`
		`    **}**`
		`    **else**`
		`    **{**`
		`      **reportError("Invalid character entered");**`
		`    **}**`
		`  }`
			4

12	1	**void sendSignals(String morseCodeString)** **{** **int morseCodeLength = morseCodeString.length();** **String signalString = EMPTYSTRING;** **for(int i = 0; i < morseCodeLength; i++)** **{** **if(morseCodeString.charAt(i) == '.')** **{** **signalString = signalString + "=" + SPACE;** **}** **else if(morseCodeString.charAt(i) == '-')** **{** **signalString = signalString + "===" + SPACE;** **}** **else if(morseCodeString.charAt(i) == SPACE)** **{** **signalString = signalString + SPACE + SPACE;** **}** **}** **Console.writeLine(signalString);** **}**	7

13	1	**void outputAlphabetWithCode(char[] letter, String[] morseCode)**
		{
		int counter = 0;
		for(int i = 1; i < 27; i++)
		{
		if(counter%4==0)
		{
		Console.writeLine();
		}
		Console.write(letter[i] + " ");
		Console.write(String.format("%-6s", morseCode[i]));
		counter++;
		}
		}
		void displayMenu()
		{
		Console.writeLine();
		Console.writeLine("Main Menu");
		Console.writeLine("=========");
		Console.writeLine("A - Output the alphabet with code");
		Console.writeLine("R - Receive Morse code");
		Console.writeLine("S - Send Morse code");
		Console.writeLine("X - Exit program");
		Console.writeLine();
		}
		void sendReceiveMessages()
		{
		int[] dash = { 20, 23, 0, 0, 24, 1, 0, 17, 0, 21, 0, 25, 0, 15, 11, 0, 0, 0, 0, 22, 13, 0, 0, 10, 0, 0, 0 };
		int[] dot = { 5, 18, 0, 0, 2, 9, 0, 26, 0, 19, 0, 3, 0, 7, 4, 0, 0, 0, 12, 8, 14, 6, 0, 16, 0, 0, 0 };
		char[] letter = { "SPACE", 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z' };
		String[] morseCode = { "SPACE + """, ".-", "-...", "-.-.", "-..", ".", "..-.", "--.", "....", "..", ".---", "-.-", ".-..", "--", "-.", "---", ".--.", "--.-", ".-.", "...", "-", "..-", "...-", ".--", "-..-", "-.--", "--.." };
		boolean programEnd = false;
		while (!programEnd)
		{
		displayMenu();
		char menuOption = getMenuOption();
		if (menuOption == 'R')
		{
		receiveMorseCode(dash, letter, dot);
		}
		else if (menuOption == 'S')
		{
		sendMorseCode(morseCode);
		}
			6

```
    else if (menuOption == 'A')
    {
      outputAlphabetWithCode(letter, morseCode);
    }
    else if (menuOption == 'X')
    {
      programEnd = true;
    }
  }
}
```

14 1

```
void sendReceiveMessages()
{
  int[] keys = new int[3];
  for (int i = 1; i < 4; i++)
  {
    boolean validNumber = false;
    while(!validNumber)
    {
      try
      {
        Console.write("Enter key number " + i + ": ");
        keys[i-1] = Integer.parseInt(Console.readLine());
        validNumber = true;
      }
      catch (Exception e)
      {
        reportError("Only enter integers");
      }
    }
  }
  int[] dash = { 20, 23, 0, 0, 24, 1, 0, 17, 0, 21, 0, 25, 0,
15, 11, 0, 0, 0, 0, 22, 13, 0, 0, 10, 0, 0, 0 };
  int[] dot = { 5, 18, 0, 0, 2, 9, 0, 26, 0, 19, 0, 3, 0, 7, 4,
0, 0, 0, 12, 8, 14, 6, 0, 16, 0, 0, 0 };
  char[] letter = { "SPACE", 'A', 'B', 'C', 'D', 'E', 'F', 'G',
'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
'U', 'V', 'W', 'X', 'Y', 'Z' };
  String[] morseCode = { "SPACE + """, ".-", "-...", "-.-.", "-
..", ".", "..-.", "--.", "....", "..", ".---", "-.-", ".-..", "-
-", "-.", "---", ".--.", "--.-", ".-.", "...", "-", "..-", "...-
", ".--", "-..-", "-.--", "--.." };
  boolean programEnd = false;

  while (!programEnd)
  {
    displayMenu();
    char menuOption = getMenuOption();
    if (menuOption == 'R')
    {
      receiveMorseCode(dash, letter, dot);
    }
    else if (menuOption == 'S')
    {
      sendMorseCode(morseCode, keys);
    }
    else if (menuOption == 'A')
    {
      outputAlphabetWithCode(letter, morseCode);
    }
    else if (menuOption == 'X')
    {
```

9

```
      programEnd = true;
    }
  }
}
void sendMorseCode(String[] morseCode, int [] keys)
{
  Console.writeLine("Enter your message (uppercase letters and
spaces only): ");
  String plainText = Console.readLine();
  int plainTextLength = plainText.length();
  String morseCodeString = EMPTYSTRING;
  int index;
  int keyNumber = 0;
  for (int i = 0; i < plainTextLength; i++)
  {
    char plainTextLetter = plainText.charAt(i);
    if (plainTextLetter == SPACE)
    {
      index = 0;
    }
    else
    {
      index = (int)plainTextLetter - (int)'A' + 1;
    }
    index += keys[keyNumber];
    keyNumber++;
    if(keyNumber == 3)
    {
      keyNumber = 0;
    }
    if(index > 26)
    {
      index = index - 27;
    }
    else if(index < 0)
    {
      index = index + 27;
    }
    if (index >= 0 && index <= 26)
    {
      String codedLetter = morseCode[index];
      morseCodeString = morseCodeString + codedLetter + SPACE;
    }
    else
    {
      morseCodeString = EMPTYSTRING;
      break;
    }
  }

  if(morseCodeString != EMPTYSTRING)
```

```
  {
    Console.writeLine(morseCodeString);
  }
  else
  {
    reportError("Enter only space or uppercase letters");
  }
  sendSignals(morseCodeString);
}
```