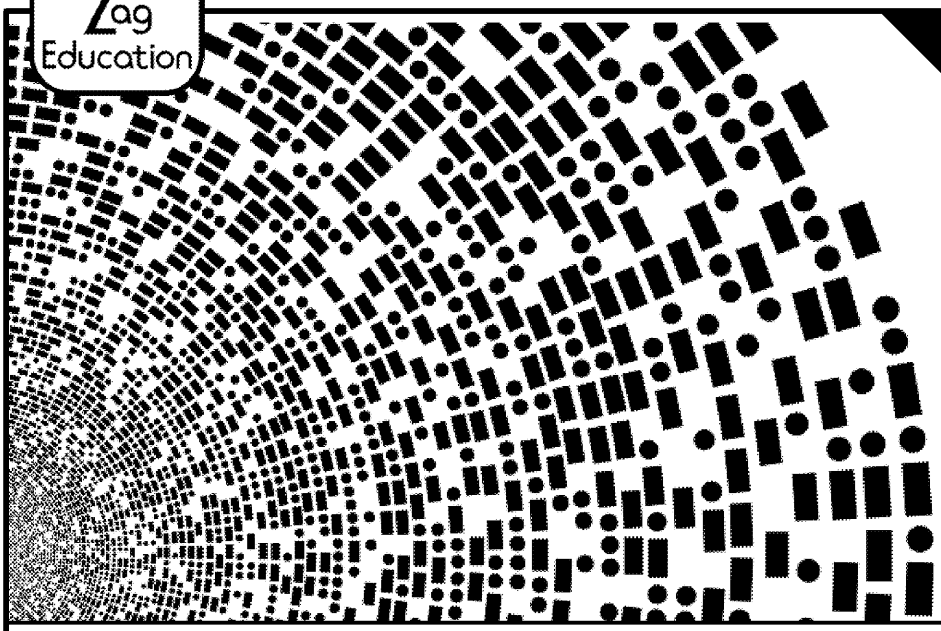
Zig Zag Education

Computer Science AS | AQA | 7516



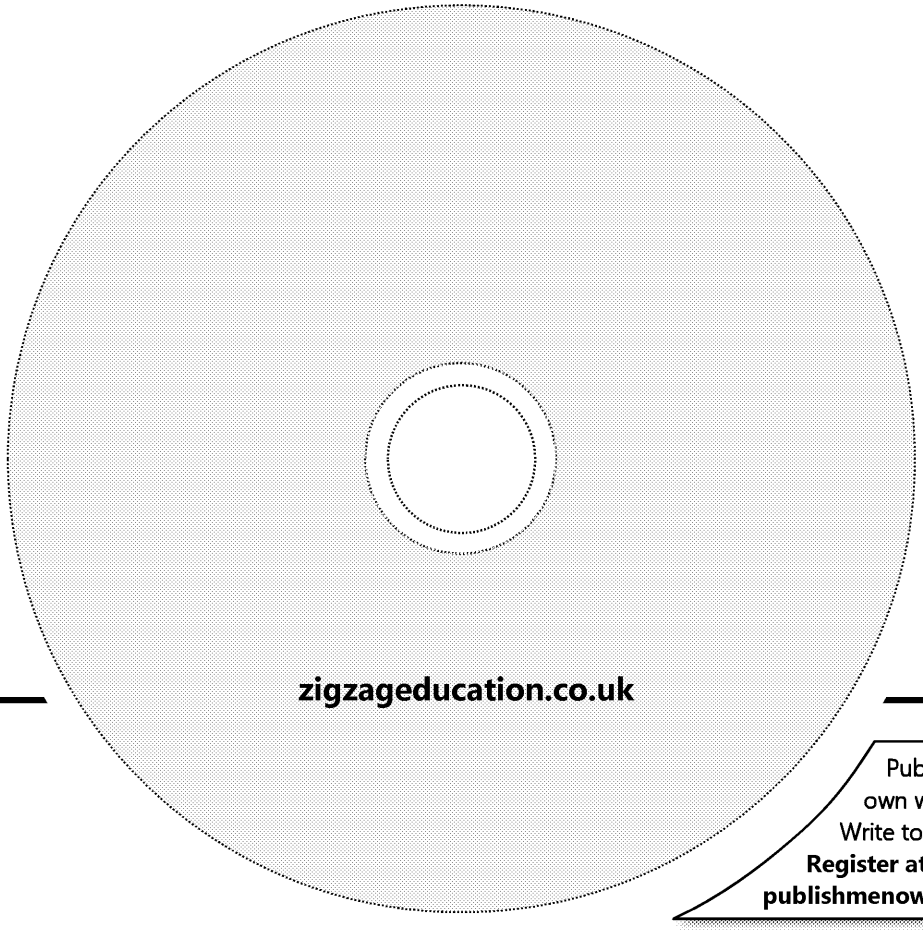
2015 specification
for the 2018 AS exam

PAPER 1 EXAM RESOURCE PACK 2018

MORSE CODE

VB.NET

for AS AQA Computer Science



CH10/
8529

POD
8529

zigzageducation.co.uk

Contents

Teacher's Introduction

This resource pack is designed to help you support your students taking the **AS Computer Science Paper 1** examination. It is based on the ***Morse Code*** preliminary material (VB.NET) – for examination June 2018.

On the CD, you will find the following files.

`MorseCode`	for student use – this folder contains all of the content, accessible via a HTML interface
`editable`	for teacher use – this folder contains ALL of the documents in editable (docx) formats
`Passwords.txt`	for teacher use – this file contains all of the passwords for the protected PDFs (also listed below)

* PRINTED COPIES OF ALL THE MATERIALS IN THIS DIGITAL RESOURCE PACK ARE INCLUDED FOR REFERENCE.

Installation: Copy the entire `MorseCode` folder onto a network location that is accessible for students, and provide them with a shortcut to the index.html file. All content can be accessed from this page.

Passwords: All of the PDFs in the 'Answers & Solutions' HTML page (`answers.html`) are password-protected, so that students can only access them with your permission. Each password is a four-digit code, as follows:

`Commentary.pdf`	`1005`
`DiagramComplete.pdf`	`6113`
`QuestionsMarkScheme.pdf`	`2887`
`TaskMarkScheme.pdf`	`4392`

Should you wish to give students access to ALL protected-PDFs, the master password for all files is: `zz2ghc4`

The resource pack consists of the following:

① **Pre-release Commentary**, consisting of two parts:

- A general walkthrough of the skeleton program, including a written description and flowcharts giving a visual demonstration of the game.
- A detailed, technical overview of the skeleton program, describing how all subroutines and the various code elements work.

Note: although this section is intended to give extra support to teachers and students, it should in no way be seen as a substitute to a student exploring the code for themselves. For this reason, this content has been placed on the 'Answers & Solutions' HTML page as a password-protected file, to allow you to control if/when students access it.

② **Structure Diagram Activity**

Partially completed structure diagram activity for students to complete while getting to grips with the skeleton program. Any missing subroutine names, return values, parameters and directional arrows must be added to the diagram. An A4 printed copy is provided in this pack for reference, however it is recommended that you print this in A3 size from the PDF. Solutions are provided on the *Answers & Solutions* page as a protected PDF.

③ **Written Questions**

Theory questions testing students' understanding of the *Morse Code* program. These questions require access to the skeleton code, but no modifications need to be made to the program. Write-on (with answer lines) and non-write-on version are available format. Solutions are provided on the *Answers & Solutions* page as a protected PDF.

④ **Programming Tasks**

Fifteen modification exercises put students' programming skills to the test. Solutions are provided on the *Answers & Solutions* page as a protected PDF. Note that these are example solutions and you must use your discretion to award marks accordingly where there are valid alternative solutions.

Free Updates

Register your email address to receive any future free minor updates made to this resource or other Computing resources your school has purchased, and details of any promotions for your subject.

** resulting from minor specification changes, suggestions from teachers and peer reviews, or occasional errors reported by customers*

zzed.uk/freeupdates

An **Electronic Answer Document (EAD)** is provided should you wish students to use it for ③ and/or ④ above.

This resource is intended to supplement your teaching only. **Please read full disclaimer (p. iv) before using it.**

-- --- .-. ... .

MORSE CODE

-.-. --- -.. .

Description of the Program

The program is a system that converts between plaintext and Morse code.

Plaintext is language printed alphabetically (A, B, C, etc.) whereas Morse code u
to represent each letter in the alphabet:

Plaintext	Morse code	Plaintext	Morse code
A	.-	J	.---
B	-...	K	-.-
C	-.-.	L	.-..
D	-..	M	--
E	.	N	-.
F	..-.	O	---
G	--.	P	.--.
H	....	Q	--.-
I	..	R	.-.

Each character is separated by a space, so the word HELLO is represented as foll

.... . .-.. .-.. ---

H	E	L	L
....	.	.-..	.-

Included w e pre-release material is a text file called 'message.txt'. The c

===⌂⌂⌂=⌂⌂⌂=⌂===⌂⌂⌂⌂⌂⌂===

Note: *The ⌂ symbols are not included in the text file, they have been included in thes to make them more visible for this explanation. The message.txt file consists of spac*

Overview

The program has two subroutines that handle conversion between plaintext and

ReceiveMorseCode

The subroutine `ReceiveMorseCode` reads Morse code from a text file and con
key subroutines used to perform this conversion is `Decode`. The subroutine `Se`
from the user at the keyboard and converts it to Morse code.

`ReceiveMorseCode` consists of three main stages:

1. Extract text from a file. The file contains only spaces and equals symbols. A single equals (=) makes a dot. Three in a row (===) make a dash.

   Here is an extract from the text file:

   ===⌂=⌂=⌂===

2. Convert the series of equals symbols to a series of dots and dashes. The sequence in the box above would become:

   - . . -

3. Convert the series of dots and dashes to plaintext, which is a letter between A and Z. The pattern in the box above would become:

   X

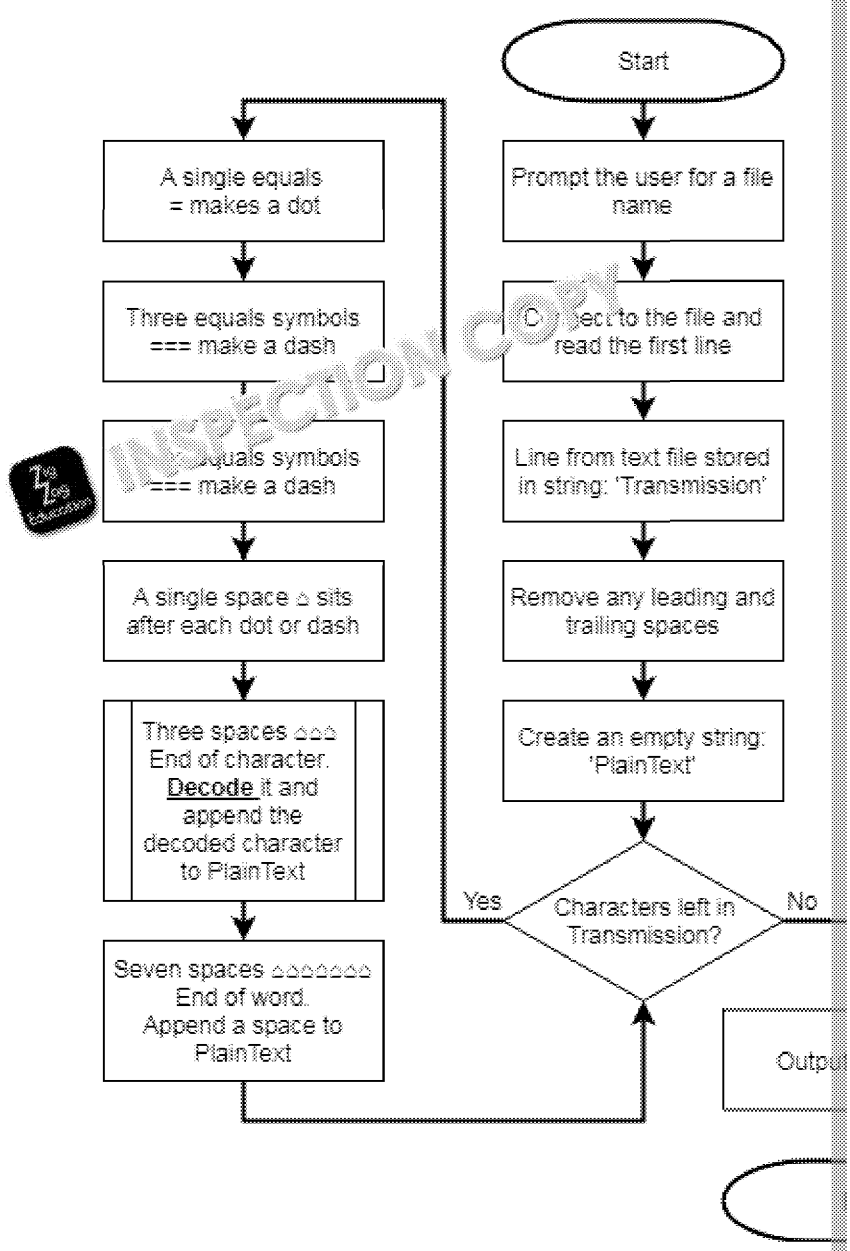
This
repe
enti
bee
pla
poi
in t

SendMorseCode

`SendMorseCode` is less involve
types uppercase plaintext at the co
Morse code. The Morse
displayed on the console. Any spaces in the
three space
or

Input	Output
COMPUTING	-.-. --- -- .--. ..-
AQA AS	.- --.- .- .- ...

ReceiveMorseCode Subroutine



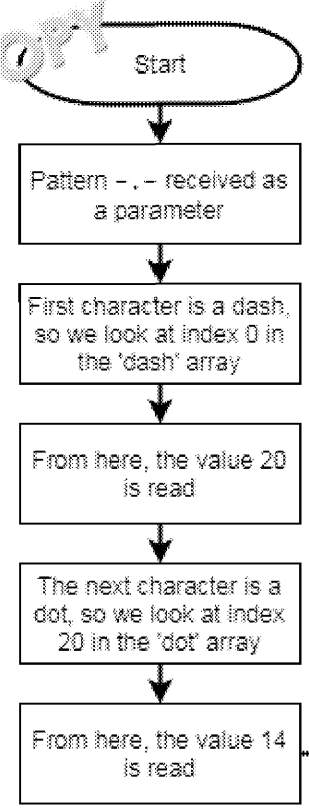
`ReceiveMorseCode` calls seven other subroutines, either directly or indirectly in the flowchart, as the flowchart exists only to provide a top-level understanding

Decode Subroutine

Element index in array:	Dot	Dash	Letter
0	5	20	⌂
1	18	23	A
2	0	0	B
3	0	0	C
4	2	24	D
5	9	1	E
6	0	[illegible]	F
7	[illegible]	17	G
8	0	0	H
9	19	21	I
10	0	0	J
11	3	25	K
12	0	0	L
13	7	15	M
14	4	11	N
15	0	0	O
16	0	0	P
17	0	0	Q
18	12	0	R
19	8	22	S
20	14	13	T
21	6	0	U
22	0	0	V
23	16	10	W
24	0	0	X
25	0	[illegible]	Y
26	[illegible]	0	Z

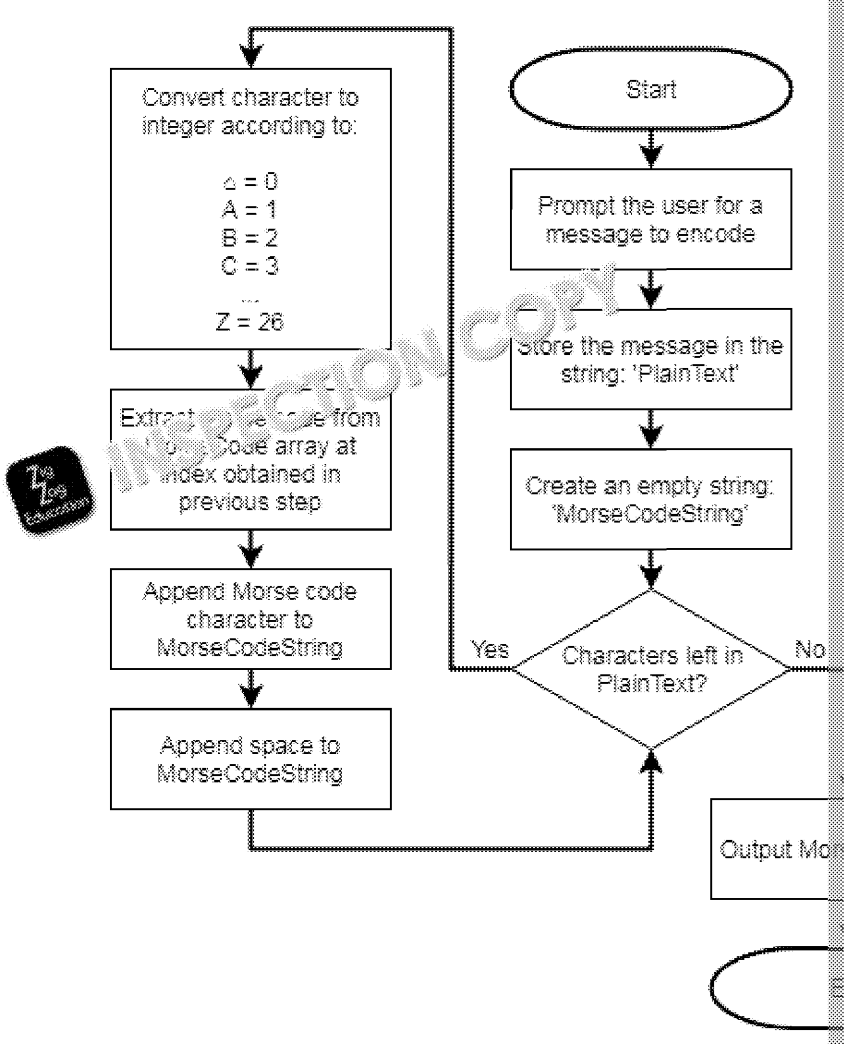
The subroutine `Decode` use
`Dot`, `Dash` and `Letter`, wh
throughout execution.

The flowchart below shows
the pattern `-.-` into the pla



If the first character is a das
looking at index 0 in the Da
is a dot (`.`), the starting poi

SendMorseCode Subroutine



Unlike `ReceiveMorseCode`, which calls several other subroutines, `SendMorse` calls no other subroutines. The user enters a message, which is validated to ens characters and spaces. The message is then translated, one character at-a-time, taken from an array called `MorseCode`.

The Text File (message.txt)

The contents of the text file are explained below:

===⌂⌂⌂	This is a dash (===), followed by three spa Three spaces signals the end of a characte The character that is made up of a single
=⌂⌂⌂	This is the se a character, which is a si
=⌂==⌂	This character is a dot followed by a dash A single space is used between them (inst the character is not finished yet. The Morse code comprising a dot followe
⌂⌂⌂⌂⌂⌂⌂	This is then followed by seven spaces, wh between two words.
===⌂=⌂=⌂===⌂	This is a character that is made up of a da by a dot, followed by a dash, which make

The whole message, therefore, is **TEA X**

Subroutine Calls, Parameters and Return Values

The numbers to the left do **not** indicate the order in which subroutines are called, as there are multiple possible orders.
Instead, these numbers relate to the numbers in the structure diagram.

Call	Parameters	Return
1 `Main` **calls** `SendReceiveMessages`	–	–
2 `SendReceiveMessages` **calls** `DisplayMenu`	–	–
3 `SendReceiveMessages` **calls** `GetMenuOption`	–	`String: MenuOption`

4 `SendReceiveMessages` **calls** `ReceiveMorseCode`	`Integer(): Dash`
	`String(): Letter`
	`Integer(): Dot`
		–

5 `SendReceiveMessages` **calls** `SendMorseCode`	`String(): MorseCode`	–
6 `ReceiveMorseCode` **calls** `GetTransmission`	–	`String: Transmission`
7 `ReceiveMorseCode` **calls** `GetNextLetter`	`Integer: i` `String: Transmission`	`String: SymbolString`

8 `ReceiveMorseCode` **calls** `Decode`	`String: CodedLetter`
	`Integer(): Dash`
	`String(): Letter`
	`Integer(): Dot`
		`String: Letter(Integer: Pointer)`
		
		This returns a string, but the string is always one character long, and is a character within the string `Letter`, at location `Pointer`.
		If `Letter` contains the string “Hello” , then `Letter(0)` = H, `Letter(1)` = E, etc.

Description of Subroutines

Each subroutine is described below.

Subroutine Name	Description	

`Decode`
 	
Receives a coded letter (i.e. a letter in Morse code, such as -..-), and returns the corresponding plain text letter ('X' in this case)
	Parameters: `String: CodedLetter`
	`Integer(): Dash`
	`String(): Letter`
	`Integer(): Dot`
	Returns: `String: Letter(Int: Pointer)`
	Called from: `ReceiveMorseCode`
	Calls: –
		1. Initialise an integer variable `CodedLetterLength` to be equal to the length of the parameter `CodedLetter`
		2. Initialise an integer variable `Pointer` to zero
		3. Set up a loop to iterate through each character in `CodedLetter`, using the variable `i`
		4. If `i` points to a space, this subroutine returns a space to `ReceiveMorseCode`
		5. If `i` points to a dash, `Pointer` is changed to navigate the Morse code binary tree (see Preliminary Material, page 4), one step to the left
		6. If i points to a dot, `Pointer` is changed to navigate the Morse code binary tree, one step to the right
		7. By the time `i` has looped through each dot/dash in the encoded character, the value of `Pointer` should point (in the `Letter` array) to the letter that corresponds to the Morse code character
		8. If a space is not returned to `ReceiveMorseCode` in step 4 (above), the letter identified in step 7 is returned as a string

`DisplayMenu` *Displays three options to the user – send Morse code, receive Morse code or end the program*	Parameters: – Returns: – Called from: `SendReceiveMessages` Calls: –	1. Output three menu options (R, S, X), one on each line

Subroutine Name	Description	
`GetNextLetter` *A Morse code transmission usually consists of multiple letters. This subroutine extracts the next letter from a transmission.*	Parameters: `Integer: i` `String: Transmission` Returns: `String: SymbolString` Called from: `ReceiveMorseCode` Calls: `GetNextSymbol`	1. Declare string variable `SymbolString` and initialise it to an empty string 2. Set up a loop to repeat until any **one** of these conditions is met: • A space is returned from a call to `GetNextSymbol` (meaning the Morse character being parsed has ended) • The `EOL` character (#) is reached (meaning the end of the entire message has been reached) • The two characters after the current character are both spaces (meaning the letter has ended) 3. Within the loop, a call is made to `GetNextSymbol`, which will return a space, a dash or a dot. A space (see first bullet point) terminates the loop 4. If the call to `GetNextSymbol` returns a dash or a dot, that dash or dot is appended to the string variable `SymbolString` 5. At the end of the word (see bullet points), `SymbolString` is returned to `ReceiveMorseCode`
`GetNextSymbol` *A Morse code letter can consist of multiple symbols (combinations of dots and dashes). There are also spaces, which are used to separate them. This subroutine determines whether the next symbol is a dot, a dash or a space.*	Parameters: `Integer: i` `String: Transmission` Returns: `String: Symbol` Called from: `GetNextLetter` Calls: `ReportError`	1. When the parameter `i` is initially passed to this subroutine, its value is zero; any changes to `i` are reflected in other subroutines, as it is passed by reference, not by value 2. Integer variable `SymbolLength` initialised to zero 3. `i` is used to point to characters within the string variable `Transmission` 4. If `i` points to the # character, '*End of transmission*' is written to the console, and an empty string is returned to `GetNextLetter` 5. Otherwise, `i` is incremented until it reaches either a space or the `EOF` character (#) within `Transmission`

Subroutine Name	Description	

`GetTransmission`
 	
This subroutine prompts the user for a filename, then reads the first line of the corresponding file, passing it back to `ReceiveMorseCode`
	Parameters: –
	Returns: `String: Transmission`
	Called from: `ReceiveMorseCode`
	Calls: `StripLeadingSpaces`
	`StripTrailingSpaces`
	`ReportError`
		1. Prompt the user for a file name
		2. Create a `StreamReader` connected to the specified file
		3. Read the first line of the file into the variable `Transmission`
		4. Pass the variable `Transmission` to the subroutine `StripLeadingSpaces`, from which it should be returned
		5. If the length of `Transmission` at this point is greater than zero, pass it to `StripTrailingSpaces`, from which it should be returned
		6. Append a # symbol (to mark end of line) to the variable `Transmission`
		7. If any errors occur between steps 2 and 6, call `ReportError` (passing '*No transmission found*' as a parameter) and set the variable `Transmission` to an empty string
		8. Return the variable `Transmission` to the subroutine `ReceiveMorseCode`

`Main` *This subroutine only exists to start the program (by calling `SendReceiveMessages`) and to pause the program before it terminates*	Parameters: – Returns: – Called from: – Calls: `SendReceiveMessages`	1. Call `SendReceiveMessages` 2. Prompt the user to press Enter before program execution ends

`ReceiveMorseCode`
 	
Calls other subroutines to manage the process of retrieving
	Parameters: `Integer(): Dash`
	`String(): Letter`
	`Integer(): Dot`
	Returns: –
		1. Set string variables `PlainText` and `MorseCodeString` to contain empty strings
		2. Set the string variable `Transmission` to contain the return value from a call to the subroutine `GetTransmission`

Subroutine Name	Description	

`ReportError`
 	
Writes an error to the console between two asterisks
	Parameters: `String: s`
	Returns: –
	Called from: `GetTransmission`
	`StripLeadingSpaces`
	`GetNextSymbol`
	Calls: –
		1. The error message arrives as a string parameter called `s`
		2. Parameter `s` is displayed between two asterisks

`SendMorseCode` *Accepts a plain text input from the user, translates it into Morse code and outputs the translation to the console*	Parameters: `String(): MorseCode` Returns: – Called from: `SendReceiveMessages` Calls: –	1. Prompt the user for a message to be encoded 2. Store the message in the variable `PlainText` 3. Store the length of the message in the variable `PlainTextLength` 4. Initialise variable `MorseCodeString` as an empty string 5. Set up a loop to iterate through each character in `PlainText` 6. If the character is a space, the integer variable `Index` is set to 0 7. Otherwise, `Index` is set to a number that represents that letter's position in the alphabet, e.g. if the letter is A, `Index` will be set to 1; if the letter is B, `Index` will be set to 2; etc. 8. The value of `Index` is used as an index in the `MorseCode` array that was passed in as a parameter. For example, if the letter being examined was A, the value of `Index` would be 1. Element 1 would then be retrieved from the `MorseCode` array. 9. The Morse code value retrieved from the array is appended to the variable `MorseCodeString`, followed by a space 10. Once steps 6–9 have been performed on each character in the variable `PlainText`, the `variableMorseCodeString` is output to the console

Subroutine Name	Description	

`SendReceiveMessages`
 	
This contains the main program loop, which repeatedly displays the menu, prompts the user for an input, then calls the appropriate subroutine in response. This loop ends when the user indicates a desire to end the program.
	Parameters: –
	Returns: –
	Called from: `Main`
	Calls: `DisplayMenu`
	`GetMenuOption`
	`ReceiveMorseCode`
	`SendMorseCode`
		1. Initialise `Dash` array (to contain integer pointers that relate to the Morse code binary tree)
		2. Initialise the `Letter` array (SPACE, 'A', B', 'C' … 'Z')
		3. Initialise the `Dot` array (to contain integer pointers that relate to the Morse code binary tree)
		4. Initialise the `MorseCode` array (to contain the Morse equivalents of letters, in the same order as the `Letter` array)
		5. Begin a loop that continues until the user indicates that they want to end the program
		5. Call `DisplayMenu` to display the menu
		6. Call `GetMenuOption` to get user input from menu
		7. Either call `ReceiveMorseCode`, call `SendMorseCode`, or terminate the loop, depending on user input

`StripLeadingSpaces` *Removes any spaces from the left of a string*	Parameters: `String: Transmission` Returns: `String: Transmission` Called from: `GetTransmission` Calls: `ReportError`	1. Store the length of the transmission in the integer variable `TransmissionLength` 2. Set up a loop that repeats while the first character of `Transmission` is a space, and while the length of `Transmission` is greater than zero 3. Within that loop, decrement the variable `TransmissionLength` and remove the first character of `Transmission` 4. If, after the loop, the length of `Transmission` is zero, call the subroutine `ReportError`, passing to it the string '*No signal received*' as a parameter
`StripTrailingSpaces`	Parameters: `String: Transmission`	1. Set the integer variable `LastChar` to point to the index of the last

Description of Variables, Constants and Parameters

The following table contains variables ⓥ, constants ⓒ and parameters ⓟ

Name	Type	Description	Created in
CodedLetter ⓟ	String	Contains a single Morse code letter that is about to be decoded (passed by value)	Decode ReceiveMorseCode
CodedLetter ⓥ	String	Contains a single Morse code letter that is about to be decoded or has just been encoded	ReceiveMorseCode SendMorseCode
CodedLetterLength ⓥ	Integer	The number of Morse symbols in an encoded letter	Decode
Dash ⓟ	Integer array	Contains pointers to left branches of the binary tree seen on the Preliminary Material document, page 4 (passed by value)	Decode ReceiveMorseCode
Dash ⓥ	Integer array	Contains pointers to left branches of the binary tree seen on the Preliminary Material document, page 4	SendReceiveMessages
Dot ⓟ	Integer array	Contains pointers to right branches of the binary tree seen on the Preliminary Material document, page 4 (passed by value)	Decode ReceiveMorseCode
Dot ⓥ	Integer array	Contains pointers to right branches of the binary tree seen on the Preliminary Material document, page 4	SendReceiveMessages
EMPTYSTRING ⓒ	String	Constant to store an empty string: “”	(global)
EOL ⓒ	Char	Constant to store # symbol, which marks the end of a line	(global)
FileName ⓥ	String	Name (which can include path) of a text file to be read	GetTransmission

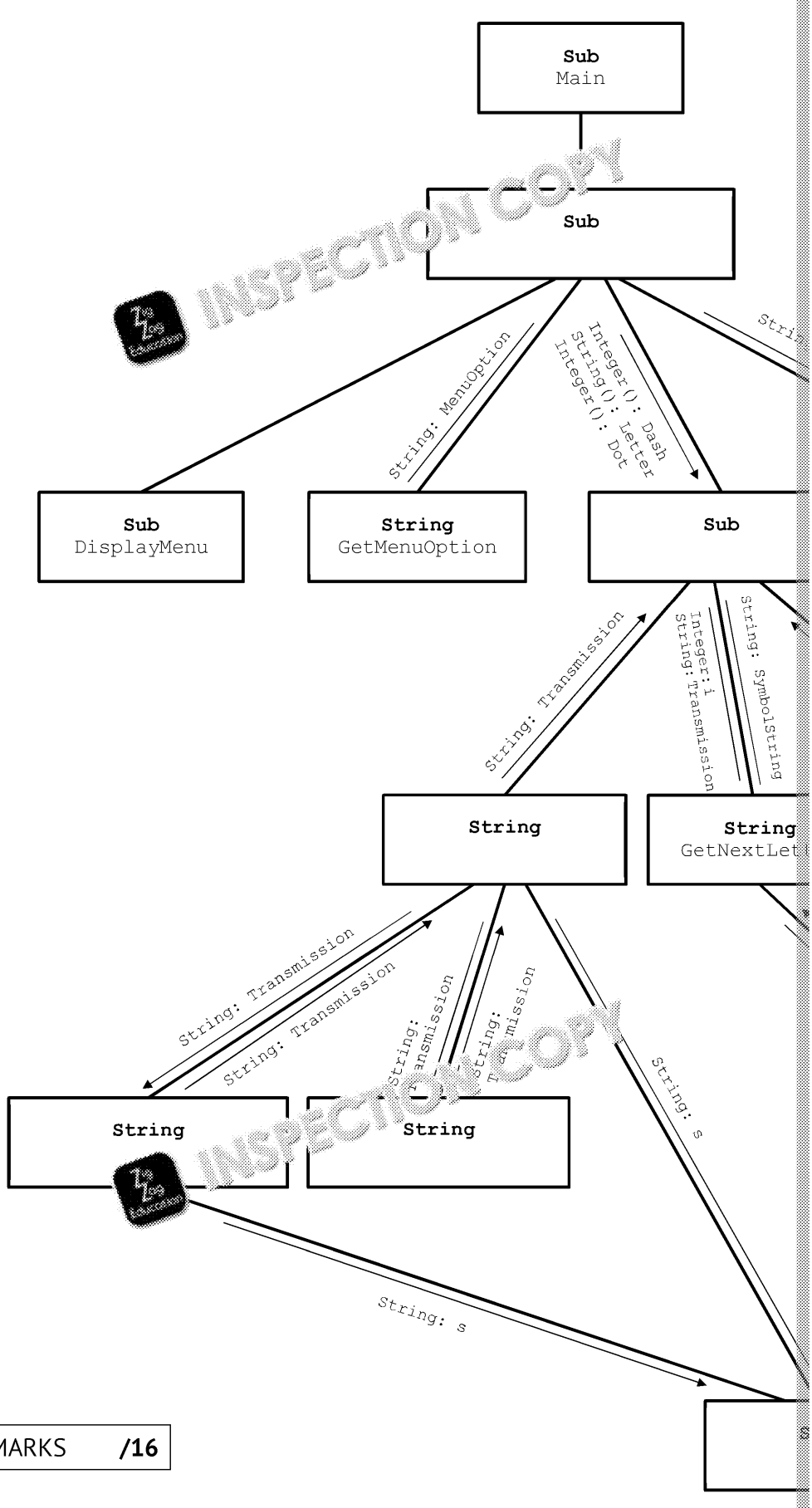
Name	Type	Description	Created in
Letter ⓟ	String array	Contains a space in the first element, followed by the upper-case alphabet, with each letter in its own element (passed by value)	Decode ReceiveMorseCode
Letter ⓥ	String array	Contains a space in the first element, followed by the upper-case alphabet, with each letter in its own element	SendReceiveMessages
LetterEnd ⓥ	Boolean	Set to true if the end of a Morse code letter has been reached while it is being parsed character by character	GetNextLetter
MenuOption ⓥ	String	Contains the user's response when presented with the program's main menu	GetMenuOption SendReceiveMessages
MorseCode ⓟ	String array	Contains a space in the first element, followed by Morse code equivalents for each letter, with one such letter per element (passed by value)	SendMorseCode
MorseCode ⓥ	String array	Contains a space in the first element, followed by Morse code equivalents for each letter, with one such letter per element	SendReceiveMessages
MorseCodeString ⓥ	String	An entire Morse code message, which can contain any number of Morse code characters	ReceiveMorseCode
MorseCodeString ⓥ	String	Contains a Morse code message, constructed character by character	SendMorseCode
PlainText ⓥ	String	Contains a message that has been (or is about to be) decoded from its Morse code equivalent	ReceiveMorseCode SendMorseCode
PlainTextLength ⓥ	Integer	The number of characters to be converted to Morse code	SendMorseCode
PlainTextLetter ⓥ	Char	Contains a single, upper-case letter that has been decoded, i.e. is no longer Morse code	ReceiveMorseCode
PlainTextLetter ⓥ	Char	Contains each letter in turn, as it is about to be converted to Morse code	SendMorseCode

Name	Type	Description	Created in
SPACE ⓒ	Char	Constant to store a single space character	(global)
Symbol ⓥ	Char	Contains a dot, dash or space within a Morse code letter	GetNextSymbol Decode
Symbol ⓥ	String	Contains the value returned from GetNextSymbol (a single dot, dash or space) that forms part of a Morse code letter	GetNextLetter
SymbolLength ⓥ	Integer	Stores the number of characters in a single Morse code letter	GetNextSymbol
SymbolString ⓥ	String	Built up, one dot or dash at a time, into a Morse code letter	GetNextLetter
Transmission ⓟ	String	Stores a sequence of Morse code letters (passed by value)	StripLeadingSpaces StripTrailingSpaces GetNextSymbol GetNextLetter
Transmission ⓥ	String	Stores a sequence of Morse code letters	GetTransmission ReceiveMorseCode
TransmissionLength ⓥ	Integer	Stores the length of the Transmission variable	StripLeadingSpaces

Structure Diagram (Activity)

The following structure diagram is incomplete, and you will need to make the following changes, as requir

- Adding a subroutine's name, e.g. **`ReceiveMorseCode`**
- Adding or completing one or more parameters, e.g. **`Integer(): Dash`**
- Adding a return value, e.g. **`String: Symbol`**
- Completing the arrow by drawing its head – parameters in this diagram are passed downwards; retu



MARKS **/16**

Programming Questions

These questions refer to the preliminary material and require you to load the skel program, but do not require any additional programming.

1. State the name of an identifier for:

   a) A string constant **[1]**

   .....

   b) A subroutine with two parameters **[1]**

   .....

   c) A parameter that is passed by reference **[1]**

   .....

   d) A Boolean variable **[1]**

   .....

   e) A parameter that is an array **[1]**

   .....

   f) An integer array **[1]**

   .....

   g) A library function called from within the `GetMenuOption` subroutine **[1**

   .....

   h) A parameter passed **out** of the `GetNextLetter` subroutine **[1]**

   .....

2. State the purpose of each of the following lines in the `GetTransmission`

```
Dim Filename As String
...
Filename = Console.ReadLine()
Dim Reader As New StreamReader(Filename)
Transmission = Reader.ReadLine
```

.....

.....

.....

.....

3. Describe the purpose of the `While` loop within the `SendReceiveMessag`

.....

.....

.....

4. Describe the nature and purpose of the `Dash` data structure in `SendRecei`

.....

.....

.....

.....

.....

5. Look at the subroutine `StripLeadingSpaces`. Des e the purpose and
`FirstSignal`. **[2]**

.....

.....

.....

6. Describe each of the following lines of code, taken from the `StripTraili`

```
Dim LastChar As Integer = Transmission.Length() - 1
While Transmission(LastChar) = SPACE
      Transmission = Transmission.Remove(LastChar)
      LastChar -= 1
End While
Return Transmission
```

.....

.....

.....

.....

.....

.....

7. Describe the function of the following line from th n `MorseCode` subro

```
Index = Asc(PlainTextLe  - Asc("A") + 1
```

.....

.....

.....

.....

.....

8. Describe the purpose of the `Catch` block in the `GetTransmission` subrou

State one situation in which the code in the `Catch` block would be executed

.....

.....

.....

9. The skeleton program begins with a number of constants.

State two benefits of the program being written in this way. **[2]**

.....

.....

.....

10. The `StripLeadingSpaces` subroutine uses the `Substring` operation.

Describe the purpose of the `Substring` operation and explain how it is used

.....

.....

.....

.....

.....

11. Describe each of the circumstances that would lead to the subroutine `Repo`

.....

.....

.....

.....

.....

12. Describe fully the operation of the `Decode` subroutine if the value of `Coded`

.....

.....

.....

.....

.....

.....

.....

MORSE CODE: Programming

The following tasks require you to open the skeleton program and m

Task 1

This task refers to `GetTransmission`.

Currently, the user must include the suffix '.txt' in order to load a transmission fil
the file is not found.

Modify the code so that if '.txt' is not entered by the user as part of the file name,
If the user enters a file name that includes '.txt', the program should add nothing

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `GetTransmission`
- One screen capture showing the result of selecting option R from the ma
  the file name 'message'
- One screen capture showing the result of selecting option R from the ma
  the file name 'message.txt'

Task 2

This task refers to `SendReceiveMessages`.

At present, the main menu loops repeatedly until the user enters a valid option. I
event of an invalid entry, the error message 'Please select an option from the me
is presented the menu again.

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `SendReceiveMessages`
- One screen capture showing the result of selecting option Q from the ma
- One screen capture showing the result of selecting option S from the ma

Task 3

This task refers to `SendReceiveMessages`.

Currently, the main menu only accepts uppercase letters. Modify the code so that
the same way as uppercase letters. For example, a lowercase 'r' should have the s

All menu options should be selectable using their current uppercase option as w

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `SendReceiveMessages`
- One screen capture showing the result of selecting option r from the ma
- One screen capture showing the result of selecting option R from the ma

Task 4

This task refers to `SendMorseCode`.

When a plaintext message is input for encoding, it must consist of uppercase lette
code to allow users to input messages using lowercase letters, as well as upperca

If the user enters a purely uppercase message, it should be processed as normal.
a combination of uppercase and lowercase, they should be presented with the fo

```
Lowercase letters detected - convert to uppercase?
```

If they enter 'n', they should be re-prompted to enter the message. Otherwise, a
message should be converted to their uppercase equivalents.

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `SendMorseCode`
- One screen capture showing the result of selecting option S from the ma the message 'Basic', followed by 'y'
- One screen capture showing the result of selecting option S from the ma the message 'Basic', followed by 'n'
- One screen capture showing the result of selecting option S from the ma the message 'BASIC'

Task 5

This task refers to `DisplayMenu` and `SendReceiveMessages`, as well as a ne

Add the following option to the main menu (in any position):

```
D - Display Morse alphabet
```

If the user selects 'D' while `GetMenuOptions` is executing, a new subroutine ca
called. `SendReceiveMessages` should pass the arrays `Letter` and `MorseCo`

`ShowAlphabet` should display each plaintext character, followed by a space, fo
equivalent. Each plaintext letter, together with its Morse code equivalent, shoul
line will be:

```
A .-
```

You should not display the equivalent of a space. After the Morse code alphabet
should be output, followed by the main menu.

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `DisplayMenu`
- Your amended SOURCE CODE PROGRAM for `SendReceiveMessages`
- The SOURCE CODE for a new subroutine, in full, called `ShowAlphabet`
- One screen capture showing the result of selecting option D from the ma

Task 6

This task refers to `GetTransmission` as well as a new subroutine, `GetManua`

At the moment, a transmission signal must enter the program from a text file. A
series of equals symbols and spaces. Modify `GetTransmission` so that the fir
to the user with two options:

```
F - Read the transmission from a file
M - Enter the transmission manually
```

If the user enters 'F', execution carries on as before, with the program prompting
user enters 'M', a new subroutine `GetManualTransmission` should be called
the user for a signal by displaying the following text:

```
Enter transmission signal:
```

When the user enters a signal, it should be passed back as a string to `GetTra`
the same way as a file-read string.

No validation is required in `GetManualTransmission`, but the new submenu
should loop until the user has entered either 'F' or 'M'.

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `GetTransmission`
- The SOURCE CODE for a new subroutine, in full, called `GetManualTra`
- One screen capture showing the result of the following set of actions:
  - Select R from the main menu
  - Select M from the submenu
  - Enter the following, substituting each ⌂ with a space, and press

    =⌂===⌂⌂⌂=⌂=⌂=

Task 7

This task refers to `StripLeadingSpaces`.

Currently, any spaces at the start of a transmission are removed by `StripLead`
also removes spaces that were intended to be included within the transmission.

Modify the code so that transmissions that begin with seven spaces are not t
begin with fewer than seven spaces should be trimmed as normal.

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `StripLeadingSpaces`

Task 8

This task refers to `GetTransmission`.

At present, the message 'No transmission found' is displayed for files that are not
cause other exceptions.

Modify the code so that a file not being found calls `ReportError` with the strin
other exception should result in a call to `ReportError` with the string 'File erro

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `GetTransmission`
- One screen capture showing the result of selecting option R from the ma
  as the file name

Task 9

This task refers to `GetTransmission`.

Currently, `GetTransmission` returns a string read from a text file without vali
should contain dots represented by a single equals symbol, and dashes represent
symbols. Any other number of consecutive equals symbols would cause an inval

After calls to `StripLeadingSpaces` and `StripTrailingSpaces`, and the c
character, add code that checks for the following within the `Transmission` stri

- Four equals symbols in a row
- Two equals symbols followed by a space
- Two equals followed by the 'end-of-line' character

If any of these occur, a call should be made to `ReportError`, passing the string
`GetTransmission` should then return an empty string.

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `GetTransmission`

Task 10

This task refers to `SendMorseCode`.

Currently, the program converts user-input plaintext into Morse code and display

Modify the code so that each new Morse code message generated in `SendMorse`
transmission format (e.g. =⌂===⌂⌂⌂=⌂=⌂=) and displayed in this format on the c

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `SendMorseCode`
- One screen capture showing the result of selecting 'S' on the main menu,

Task 11

This task refers to `Decode`.

An individual dot or dash is known as a *symbol*. All possible combinations of one
assigned to letters. All combinations of four symbols are assigned with the exce

```
----
---.
.-.-
..--
```

Modify `Decode` so that if any of these four seque ce or any sequence longer th
Decode in the `CodedLetter` paramete, the asterisk * is returned to `ReceiveM`
that an error has occurred.

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `Decode`

Task 12

This task refers to `DisplayMenu` and `SendReceiveMessages`, as well as a ne
`ConvertMorseCode`.

Currently, there is no option for the message to be entered in Morse code at the k
`DisplayMenu` and `SendReceiveMessages` to add the following new menu o

```
C - Convert Morse code
```

This new menu option will need to call a new subroutine `ConvertMorseCode`
arrays `MorseCode` and `Letter`. The new subroutine should ask the user to ent
and print out the decoded message.

Any errors that arise as a result of users entering invalid characters, combinations
alphabet, or incorrect numbers of spaces should be handled with a call to `Report`

```
Data entry error
```

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `DisplayMenu`
- Your amended SOURCE CODE PROGRAM for `SendReceiveMessages`
- The SOURCE CODE for a new subroutine, in full, called `ConvertMorse`
- On screen capture showing the result of the following set of actions:
  - Select 'C' from the main menu
  - Enter the following Morse code, substituting each ⌂ with a space

```
....⌂..⌂⌂⌂-⌂....⌂.⌂.-.⌂.
```

Task 13

This task refers to `GetTransmission`.

Currently, the program reads the first line of the text file and stores it in the vari the code so that the user is asked 'Which line number' after they have specified a only the second line should be stored in transmission.

If the user requests a line that does not exist, or if they enter a value other than a display 'No transmission found' as normal.

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `GetTransmission`
- One screen capture showing the result of the following set of actions:
  - Select 'R from the main menu
  - Enter 'message.txt' at the first prompt
  - Enter '2' at the second prompt
- One screen capture showing the result of the following set of actions:
  - Select 'R' from the main menu
  - Enter 'message.txt' at the first prompt
  - Enter '1' at the second prompt

Task 14

This task refers to `SendMorseCode`.

Morse code has been used in the past by the army and navy in times of war. As w Morse code, they would also probably be encrypted to make them less intelligibl

Modify `SendMorseCode` to implement a Caesar cipher with a shift value of 5. T with ⌂ used here to represent a space. The top row represents plaintext and the text.

⌂	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W

For example, if the character L were to be sent, the Morse code for the character

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `SendMorseCode`
- One screen capture showing the result of selecting option S from the ma mes BY VWN'

Task 15

This task refers to `SendReceiveMessages` and `SendMorseCode`

Currently, the program works only with letters. In this task, you will modify the a
`MorseCode` to incorporate numerals using the Morse code values shown in the t

You should append the values in the order shown below to the `Letter` and `Mo`
need to modify the `Dot` and `Dash` arrays, so that the binary tree structure functio
letters.

You will need to ensure that `SendMorseCode` assigns values of `Index` appropr
in the array, i.e. the value for 0 (zero) will be stored in the 27th element of the `Mo`
`Index` value of 27 should be applied. For 1 [illegible] `Index` value of 28 should

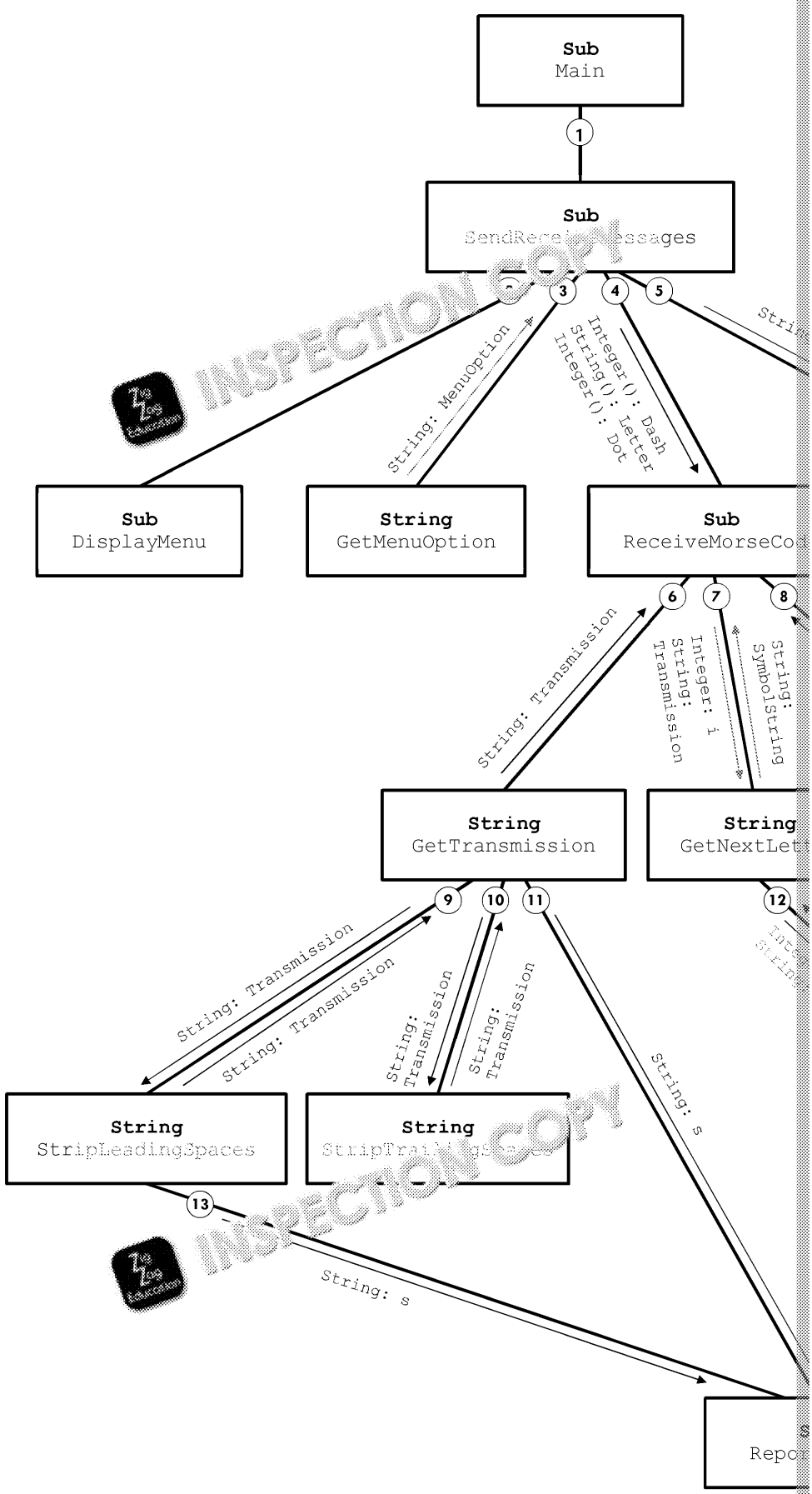
0	-----
1	[illegible]
2	[illegible]
3	...--
4	....-
5	.....
6	-....
7	--...
8	---..
9	----.

Evidence you need to provide:

- Your amended SOURCE CODE PROGRAM for `SendReceiveMessages`
- Your amended SOURCE CODE PROGRAM for `SendMorseCode`
- One screen capture showing the result of selecting option S from the ma
  the message to be encoded

Structure Diagram (Complete)

Subroutines are called downwards, i.e. `Main` calls `SendReceiveMessages`, not the other way ar
Arrows pointing downwards indicate parameters; arrows pointing upwards indicate return values.



Programming Questions (Solutions)

Q	Answer/Guidance
1a	`EMPTYSTRING`
1b	`GetNextSymbol // GetNextLetter`
1c	`i`
1d	`LetterEnd // ProgramEnd`
1e	`Dash // Letter // Dot // MorseCode`
1f	`Dash // Dot`
1g	`Length // Write // ReadLine`
1h	`i // Transmission`
2	1 mark for each of the following: • String variable declared • Initialised to user/console input • Specified file accessed/opened/passed to `StreamReader` object • Different/Transmission variable set to first line of the file
3	1 mark for each of the following: • (Repeatedly) prompt the user / accept user input • Until X is entered / loop terminates at X
4	1 mark for each of the following **(max 3)**: • Integer array • Contains pointers • Indicates which element to move to next... • ... if the next Morse signal is a dash
5	1 mark for each of the following: • Initially set to the first character in `Transmission` • As spaces are removed, it points to the new first character
6	1 mark for each of the following: • Integer is declared • Set to the index of the last character in `Transmission` • Loop repeats while `LastChar` / last character is a space • If the last character is a space, remove it (from `Transmission`) • Decrement `LastChar` / integer variable • Return `Transmission`, with all spaces removed from end/right
7	1 mark for each of the following (max 3): • Gets ASCII value of `PlainTextCharacter` • Gets ASCII value of A / Gets value 65 • Subtracts ASCII value of A / 65 from ASCII value of `PlainTextCharacter` • `PlainTextLetter` is A, `Index` is 1 (for example)
8	1 mark for each of the following: • `Catch` block executed if `Try` block fails to execute correctly / generates error • File name mistyped // file not found // error reading file // error/exception in `StripLeadingSpaces` // error/exception in `StripTrailingSpaces` // `Transmission`/`EOL` not being a valid string

Q	Answer/Guidance
9	1 mark for each of the following **(max 2)**: • Constants won't be accidentally changed • By being at the start of the code, the code is easier to read/unders *this is for the benefit of the human, not the computer*) • No need to remember (precise) values // constant names more me code is more readable
10	1 mark for each of the following **(max 3)**: • A substring is part of a string • Is used here to trim the first character/space from a string • The substring is all characters besides the [illegible] one • Is called repeatedly if multiple spaces [illegible]
11	1 mark for `StripLeadingS[illegible]stance`: • If the length [illegible]`smission` is zero 1 ma[illegible]`Ge[illegible]ransmission` instance: • [illegible]here is a file error (accept any error relating to code in the `Try` block executes // if the `Try` block fails / generates an error/except 3 marks for `GetNextSymbol` instance: • If the symbol is not a dot... • ... not a dash / minus sign... • ... not a space
12	1 mark for each of the following: • `CodedLetterLength` variable set to 4 • `For` loop to run four times • `Symbol` initially set to – (dash) • `Pointer` set to 20 • `Symbol` then set to dot (on next iteration) • `Pointer` set to 14 • `Pointer` set to 4 (on next iteration) • `Symbol` then set to dash (on next iteration) • `Pointer` set to 24 • X retrieved from `Letter` array / X returned (*only credit this mark i* *parse the arrays*)
TOTAL MARKS	

MORSE CODE: Programming

Suggested Solutions and Mark Scheme

The following are recommended solutions, and not an exhaustive list of all possible so
guidance should be used as a guide only. Discretion should be used in awarding credit w

Task 1

1 mark IF statement after `FileName` has been initialised

1 mark .txt suffix added only if missing

```
Filename = Console.ReadLine()
If Filename.IndexOf(".txt") = -1 Then
    Filename = Filename & ".txt"
End If
```

1 mark Morse code translated to 'TEA X' when suffix is not included

```
Main Menu
=========
R - Receive Morse code
S - Send Morse code
X - Exit program

Enter your choice: R
Enter file name: message
 - . . -    - . . -
TEA X
```

1 mark Morse code translated to 'TEA X' when suffix is included

```
Main Menu
=========
R - Receive Morse code
S - Send Morse code
X - Exit program

Enter your choice: R
Enter file name: message.txt
 - . . -    - . . -
TEA X
```

Task 2

1 mark Clause (probably Else) triggered by any input other than R, S or X

1 mark Correct message displayed in clause (**R**. any alternative wording, **I**. spelli

```
If MenuOption = "R" Then
    ReceiveMorseCode(Dash, Letter, Dot)
ElseIf MenuOption = "S" Then
    SendMorseCode(MorseCode)
ElseIf MenuOption = "X" Then
    ProgramEnd = True
Else
    Console.WriteLine("Please select an option from t
End If
```

1 mark Cap causing error followed by main menu being re-displayed

```
Main Menu
=========
R - Receive Morse code
S - Send Morse code
X - Exit program

Enter your choice: Q
Please select an option from the menu

Main Menu
=========
R - Receive Morse code
S - Send Morse code
X - Exit program

Enter your choice: _
```

1 mark Capital R followed by prompt for file

```
Main Menu
=========
R - Receive Morse code
S - Send Morse code
X - Exit program

Enter your choice: R
Enter file name: _
```

Task 3

1 mark Uppercase options still function correctly

1 mark Lowercase r, s and x have the same effect as R, S and X respectively

1 mark No other inputs trigger a menu option

```
MenuOption = GetMenuOption()
MenuOption = MenuOption.ToUpper
If MenuOption = "R" Then
    ReceiveMorseCode(Dash, Letter, Text)
ElseIf MenuOption = "S" Then
    SendMorseCode(MorseCode)
ElseIf MenuOption = "X" Then
    ProgramDone = True
End If
```

Full marks can be awarded for any functioning solution; an alternative is to use OR o uppercase letter in each If/ElseIf statement.

1 mark Lowercase r followed by prompt for file

```
Main Menu
=========
R - Receive Morse code
S - Send Morse code
X - Exit program

Enter your choice: r
Enter file name: _
```

1 mark Uppercase R followed by prompt for file

```
Main Menu
=========
R - Receive Morse code
S - Send Morse code
X - Exit program

Enter your choice: R
Enter file name: _
```

Task 4

1 mark Creation of a loop that begins before prompt for message

1 mark Termination condition possible (**A**. if termination condition never actually

```
Dim validEntry As Boolean = False
Dim responseToYN As String

While validEntry = False

    Console.Write("Enter your message (uppercase letters a
    PlainText = Console.ReadLine()
```

1 mark Every character in the input string is checked

1 mark Any lowercase characters would be detected

1 mark Integer incremented, Boolean set or complex **If** statement to denote dete

```
Dim lowerCaseCout As Integer = 0
For x = 1 To PlainText.Length
    If Char.IsLower(PlainText(x - 1)) Then
        lowerCaseCout += 1
    End If
Next
```

1 mark Check for any detected lowercase character (**A**. if part of same complex **If**

1 mark Correct prompt displayed (**R**. any alternative wording, **I**. spelling/capitalis
lowercase characters have been entered

1 mark Response to prompt processed via variable or complex If statement

1 mark If user enters 'y' (**I**. case), convert `PlainText` to uppercase

1 mark Loop terminated if `PlainText` converted to uppercase

1 mark Loop terminated if there were no lowercase characters

```
    If lowerCaseCout > 0 Then
        Console.Write("Lowercase letters detected - convert
        responseToYN = Console.ReadLine
        If Not (responseToYN = "n") Then
            PlainText = PlainText.ToUpper
            validEntry = True
        End If
    Else
        validEntry = True
    End If
End While
```

1 mark Input sequence of 'S', 'Basic', 'y' and Morse code output

```
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):
Lowercase letters detected - convert to uppercase? (y/n)
-... .- ... .. -.-.
```

1 mark Input sequence of 'S', 'Basic', 'n' and re-prompt for message

```
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):
Lowercase letters detected - convert to uppercase? (y/n)
Enter your message (uppercase letters and spaces only):
```

1 mark Input sequence of 'S', 'BASIC' and Morse code output

```
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):
-... .- ... .. -.-.
```

Task 5

1 mark New menu option added in `DisplayMenu` (**R**. any alternative wording,

```
Sub DisplayMenu()
    Console.WriteLine()
    Console.WriteLine("Main Menu")
    Console.WriteLine("=========")
    Console.WriteLine("R - Receive Morse code")
    Console.WriteLine("S - Send Morse code")
    Console.WriteLine("D - Display Morse alphabet")
    Console.WriteLine("X - Exit program")
    Console.WriteLine()
End Sub
```

1 mark Appropriate sel[illegible]ucture used in `SendReceiveMessages`

1 mark Cal[illegible]`splayAlphabet` in correct place

1 mark Letter and `MorseCode` arrays passed (**R**. if any other structures or variabl

```
If MenuOption = "R" Then
    ReceiveMorseCode(Dash, Letter, Dot)
ElseIf MenuOption = "S" Then
    SendMorseCode(MorseCode)
ElseIf MenuOption = "D" Then
    ShowAlphabet(Letter, MorseCode)
ElseIf MenuOption = "X" Then
    ProgramEnd = True
End If
```

1 mark Sub `DisplayAlphabet` declared

1 mark Two string arrays declared as parameters

1 mark An attempt to loop through each element of the arrays, even if unsuccess

1 mark All plaintext letters correctly displayed

1 mark All Morse code letters correctly displayed

1 mark Space not displayed

```
Sub ShowAlphabet(ByVal let[illegible] As String, ByVal MorseCode[illegible]

    For x [illegible] 1 [illegible]ter.Length - 1
        [illegible]ole.WriteLine(letter(x) & " " & MorseCode(x))
    Next

End Sub
```

1 mark Input of D and correct display of plaintext and Morse alphabets (**R**. if each
own line or if space is displayed)

```
Enter your choice: D
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  --..

Main Menu
=========
R - Receive Morse code
S - Send Morse code
D - Display Morse alphabet
X - Exit program

Enter your choice: _
```

Task 6

1 mark Declaration of `GetManualTransmission` as a string function

1 mark Prompting user for transmission signal (**R**. any alternative wording, **I**. spel

1 mark Returning the user input

```
Function GetManualTransmission() As String

    Console.WriteLine("Enter transmission signal")
    Return Console.ReadLine

End Function
```

1 mark Loop that continues until F or M [illegible]d

1 mark Menu options correctly [illegible] (**R**. any alternative wording, **I**. spelling/

1 mark Storage of r[illegible]n string variable

```
Dim sele[illegible] As String = ""

While Not (selection = "F" Or selection = "M")

    Console.WriteLine("F - read the transmission from a file")
    Console.WriteLine("M - enter the transmission manually")
    selection = Console.ReadLine

End While
```

1 mark User input of M results in no prompt for file name or attempt to access a

1 mark User input of M results in a call to `GetManualTransmission`

1 mark User input of F results in prompt for file, followed by access to that file

1 mark Input string processed identically whether manually entered or from file

```
Try
    If selection = "F" Then
        Console.Write("Enter file name: ")
        Filename = Console.ReadLine()
        Dim Reader As New StreamReader(Filename)
        Transmission = Reader.ReadLine
        Reader.Close()
    Else
        Transmission = GetManualTransmission()
    End If
    Transmission = StripLeadingSpaces(Transmission)
    If Transmission.Length[illegible] Then
        Transmission = [illegible]TrailingSpaces(Transmission)
        Transmi[illegible] Transmission + EOL
    End [illegible]
Catch
```

1 mark Input of 'R', 'M' and =⌂===⌂⌂⌂=⌂=⌂= results in plaintext AS

```
Enter your choice: R
F - read the transmission from a file
M - enter the transmission manually
M
Enter transmission signal
= ===    = = =
 .-  ...
AS
```

Task 7

1 mark Checking for seven consecutive spaces only at start of transmission

1 mark If there are seven consecutive spaces at the start, transmission is not trim

1 mark If there are not seven consecutive spaces at the start, transmission is trim

```
If TransmissionLength > 0 Then
    If Not (Transmission.IndexOf("       ") = 0) Th
        FirstSignal = Transmission(0)
        While FirstSignal = SPACE And Transmission
            TransmissionLength = 
            Transmission = Transmission.Substring(
            If TransmissionLength > 0 Then
                FirstSignal = Transmission(0)
            End If
        End While
    End If
End If
```

Task 8

1 mark Any attempt to detect two exceptions separately, even if unsuccessful

1 mark Try block contains code to attempt to access but not read specified file (
be either before the Try block or within it)

1 mark Exceptions caught call `ReportError` with string 'File not found' (**R**. an
spelling/capitalisation errors)

1 mark Exception would not be triggered by a call to `ReadLine`

```
Dim Transmission As String = EMPTYSTRING
Dim fileFound As Boolean = False

Try
    Filename = Console.ReadLine()
    reader = New StreamReader(Filename)
    fileFound = True
Catch ex As Exception
    ReportError("File not found")
End Try
```

1 mark Attempt to read file is only made if file exists

1 mark Second Try block, which must contain calls to `ReadLine`, `StripLeadi`
`StripTrailingSpaces`

1 mark Catch block calls `ReportError` with string 'File error' (**R**. any alternativ
spelling/capitalisation errors)

1 mark No path through the code allows `Transmission` to remain uninitialise

```
If fileFound Then
    Try
        Transmission = reader.ReadLine
        reader.Close()
        Transmission = StripLeadingSpaces(Transmission)
        If Transmission.Length() > 0 Then
            Transmission = StripTrailingSpaces(Transmission)
            Transmission = Transmission + EOL
        End If
    Catch ex As Exception
        ReportError("File error")
        Transmission = EMPTYSTRING
    End Try

End If
```

1 mark Input of 'R' followed by 'mmm' resulting in 'File not found' message

```
Main Menu
=========
R - Receive Morse code
S - Send Morse code
X - Exit program

Enter your choice: R
Enter file name: mmm
*     File not found     *
```

Task 9

1 mark checking for presence of four equals

1 mark checking for presence of two equals followed by a space

1 mark checking for presence of two equals followed by EOL (**A**. "#")

1 mark 'Invalid file format' passed to `ReportError` in each case (**R**. any alterna
spelling/capitalisation errors)

1 mark 'Invalid file format' would only be displayed, at most, once per call to Ge

1 mark Empty string returned in each case (**A.** "")

N.B. full credit can be given if all th[illegible] strings are checked for in a single line

```
If Transmission.Length() > 0 Then
    Transmission = StripTrailingSpaces(Transmission)
    Transmission = Transmission + EOL

    If Transmission.IndexOf("====") > -1 Then
        ReportError("Invalid file format")
        Return EMPTYSTRING
    End If

    If Transmission.IndexOf("== ") > -1 Then
        ReportError("Invalid file format")
        Return EMPTYSTRING
    End If

    If Transmission.IndexOf("==" & EOL) > -1 Then
        ReportError("Invalid file format")
        Return EMPTYSTRING
    End If

End If
```

Task 10

1 mark Variable to store transmission-format output

1 mark Loop to examine each character in `MorseCodeString`

1 mark Check whether character is a dot

1 mark If it is a dot, add a single equals to the transmission string (**A**. with or wit

1 mark Check whether the character is a dash

1 mark If it is a dot, add three equals to the transmission string (**A**. with or witho

1 mark Check whether the character is a space

1 mark If it is a space, either two or three spaces added to the transmission scree

1 mark Output should be correct for any valid combination of Morse code. One e may use substring (or similar) to handle spaces between words.

```
Dim transmission As String = EMPTYSTRING

For x = 1 To MorseCodeString.Length

    If MorseCodeString(x - 1) = "." Then
        transmission = transmission & "= "
    ElseIf MorseCodeString(x - 1) = "-" Then
        transmission = transmission & "=== "
    ElseIf MorseCodeString(x - 1) = " " Then
        transmission = transmission & "  "
    End If

Next

Console.WriteLine(transmission)
```

1 mark Following input of 'S', then 'AQA AS', output includes correct transmissio

```
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): A
.- --.- .-   .-
= ===   === =   = ===       = ===   = = =
```

Task 11

1 mark Check for length of `CodedLetter` being greater than four

1 mark Check for each invalid symbol given in question

1 mark Return * if invalid code detected (**A**. if errors made for previous marks)

1 mark No invalid value of `CodedLetter` would be checked using code already material, e.g. only a valid `CodedLetter` value would reach the 'For i = 0

```
If CodedLetter.Length > 4 Or CodedLetter = "----" Or CodedL
        CodedLetter = ".-.-" Or CodedLe  "..--" Then
    Return "*"
End If

For i = 0 To C  tterLength - 1
```

Task 12

1 mark New menu item added at the correct point in the menu (**R**. any alternativ
spelling/capitalisation errors)

```
Sub DisplayMenu()
    Console.WriteLine()
    Console.WriteLine("Main Menu")
    Console.WriteLine("=========")
    Console.WriteLine("R - Receive Morse code")
    Console.WriteLine("S - Send Morse code")
    Console.WriteLine("C - Convert Morse code")
    Console.WriteLine("X - Exit program")
    Console.WriteLine()
End Sub
```

1 mark Declaration of subroutine with two array parameters

1 mark Variable to [illegible] whether a Morse character is declared (**A**. if any attempt
var[illegible] determine whether an entered Morse character exists in the

1 mark String variable to store the plaintext output

1 mark String variable to store Morse code input

1 mark String variable to store each Morse character in turn (mark can be awarde
parsing individual letters, such as an array or list)

1 mark Any attempt to differentiate between a single space (between letters) an

1 mark Prompt the user to enter Morse code with any suitable method

1 mark Store the user response in a variable

```
Sub ConvertMorseCode(ByVal MorseCode() As String, ByVal Letter() As

    Dim validCharacter As Boolean
    Dim PlainText As String = ""
    Dim MorseMessage As String
    Dim MorseCharacter As String = ""
    Dim consecutiveSpaces As Integer = 0

    Console.Write("Enter Morse code message")
    MorseMessage = Console.ReadLine
```

1 mark Loop to examine each character in turn

1 mark Code checks each position for a dot

1 mark Code checks each position for a [illegible]

```
For x = 0 To MorseMe[illegible]gth - 1

    If M[illegible]ssage(x) = "." Then
        M[illegible]aracter = MorseCharacter & "."
        consecutiveSpaces = 0
    ElseIf MorseMessage(x) = "-" Then
        MorseCharacter = MorseCharacter & "-"
        consecutiveSpaces = 0
    End If
```

1 mark Code checks each position for a space

1 mark Code checks whether it's looking at the last character in the input string

1 mark Either of the above causes the program to process the Morse code letter

1 mark Any search method that would successfully find the Morse character in t

1 mark Corresponding value in `Letter` array appended to plaintext for output

1 mark Call to `ReportError` with 'Data Entry Error' (**R**. different wordings, **I**. sp

```
If (MorseMessage(x) = " " And consecutiveSpaces = 0) Or _
    (x = MorseMessage.Length - 1 And Not (MorseMessage(x) = " ")) Th

    validCharacter = False

    For y = 0 To MorseCode.Length - 1
        If MorseCharacter = MorseCode(y) Then
            PlainText = PlainText & Lette (y
            validCharacter = True
            MorseCharacter
        End If
    Next

    If va aracter = False Then
        ReportError("Data Entry Error")
        Exit Sub
    End If
```

1 mark Presence of three consecutive spaces adds one space to plaintext

1 mark `Plaintext` output at end of subroutine

1 mark Output would always be correct for valid Morse code entry, including any

```
        consecutiveSpaces += 1

    ElseIf MorseMessage(x) = " " And consecutiveSpaces = 2 Then
        PlainText = PlainText & " "
        consecutiveSpaces = 0
    ElseIf MorseMessage(x) = " " Then
        consecutiveSpaces += 1
    End If

Next

Console.WriteLine(PlainText)
```

1 mark Call to `ConvertMorseCode` from `SendReceiveMessages`, including Mo

```
If MenuOption = "R" Then
    ReceiveMorseCode(Dash, Letter, Do
ElseIf MenuOption = "S" Then
    SendMorseCode(Morse
ElseIf MenuOption = " Then
    Conve se(MorseCode, Letter)
```

1 mark Output from Morse code to read 'HI THERE'

```
Main Menu
=========
R - Receive Morse code
S - Send Morse code
C - Convert Morse code
X - Exit program

Enter your choice: C
Enter Morse code message.... ..   - .... . .-. .
HI THERE
```

Task 13

1 mark Variable to store the indicated line number

```
Function GetTransmission() As String
    Dim Filename As String
    Dim Transmission As String
    Dim lineNumber As Integer
```

1 mark Prompt to enter line number after file name has been entered

1 mark User input in response to prompt stored in appropriate variable (the 'Try' point in this part of the code)

```
Console.Write("Enter file name: ")
Try
    Filename = Console.ReadLine()
    Console.Write("Enter line number: ")
    lineNumber = Console.ReadLine
```

1 mark Use of any valid technique that could potentially reach beyond the first li

1 mark Technique would be effective, assuming integer has been input and that

```
Dim Reader As New StreamReader(Filename)
For x = 1 To lineNumber
    Transmission = Reader.ReadLine
Next
```

1 mark Input of 'R', 'message.txt' and '2' outputs 'No transmission found'

```
Main Menu
=========
R - Receive Morse code
S - Send Morse code
X - Exit program

Enter your choice: R
Enter file name: message.txt
Enter line number: 2
*     No transmission found     *
```

1 mark Input of 'R', 'message.txt' and '1' outputs 'TEA X'

```
Main Menu
=========
R - Receive Morse code
S - Send Morse code
X - Exit program

Enter your choice: R
Enter file name: message.txt
Enter line number: 1
 _ . . _    _ . . _
TEA X
```

Task 14

1 mark Space stored with an index of 5

1 mark Other character stored with an index five higher, i.e. `Asc("A") + 6` ins

1 mark Any attempt to 'wrap around' the characters V-Z, even if unsuccessful

1 mark V, W, X, Y, Z indexes altered to those of SPACE, A, B, C, D (0, 1, 2, 3, 4) res

```
If PlainTextLetter = SPACE Then
    Index = 5
Else
    Index = Asc(PlainTextLetter) - Asc("A") + 6
End If

If Index = 27 Then Index = 0
If Index = 28 Then Index = 1
If Index = 29 Then Index = 2
If Index = 30 Then Index = 3
If Index = 31 Then Index = 4
```

1 mark After cipher, output should be Morse equivalent of GCE AS, as below

```
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):
--. -.-. .   .- ...
```

Task 15

1 mark Adding numerals 0-9 as strings to the `Letter` array

1 mark Adding Morse equivalents to the `MorseCode` array

7 marks One for each correct value that is highlighted below in the `Dash` array

6 marks One for each correct value that is highlighted below in the `Dot` array

```
Dim Dash = {20, 23, 0, 0, 24, 1, 0, 17, 31, 21, 28, 25, 0, 15, 11, 27,
10, 0, 0, 0, 27, 0, 29, 0, 0, 0, 0, 0, 0, 0}
```

```
Dim Letter = {" ", "A", "B", "C", "D", "E", "F", "G", "H", "I", "J",
"P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z", "0", "1", "2",
"8", "9"}
```

```
Dim Dot = {5, 18, 33, 2, 0, 0, 26, 32, 19, 0, 3, 0, 7, 4, 35, 0, 0,
34, 36, 0, [illegible], 0, 0, 35, 0}
```

```
Dim MorseCode = {" ", ".-", "-...", "-.-.", "-..", ".", "..-.", "--."
"-.-", ".-..", "--", "-.", "---", ".--.", "--.-", ".-.", "...", "-",
"-..-", "-.--", "--..", "-----", ".----", "..---", "...--", "....-",
"---..", "----."}
```

1 mark Assigning Index values of 27-36 for numerals 0-9, even if unsuccessful

1 mark Successfully assigning all Index values

1 mark Not overwriting those values using the existing ASCII code, i.e. use of a

```
If IsNumeric(PlainTextLetter) Then
    Index = Integer.Parse(PlainTextLetter) + 27
ElseIf PlainTextLetter = SPACE Then
    Index = 0
Else
    Index = Asc(PlainTextLetter) - Asc("A") + 1
End If
```

1 mark Correct output for encoding a string of 123

```
Main Menu
=========
R - Receive Morse code
S - Send Morse code
X - Exit program

Enter your choice:
Enter your mes[illegible]percase letters and spaces only)
.---- [illegible]
```

Name	

ZigZag Education supporting

AS AQA Computer Science Paper 1

Summer 2018

MORSE CODE

Electronic Answer Document (EAD)

Instructions

- Enter your name in the box at the top of this page
- Answer **all** questions by entering your answers into this document
- Remember to **save** this document regularly
- Save and print this document and any additional pages

- Answer **all** questions
- The marks available for each question are shown in brackets

- You will need:
  - access to a computer
  - access to a printer
  - access to appropriate software
  - electronic copies of the required skeleton code
  - EAD (Electronic Answer Document)

Total marks:

Written Questions

Answer all questions.
Remember to save this document regularly.

Q		Answer
1	(a)	
	(b)	
	(c)	
	(d)	
	(e)	
	(f)	
	(g)	
	(h)	
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		

Programming Tasks

Answer all questions.
Remember to save this document regularly.

Q	Answer
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	