OXFORD AQA
INTERNATIONAL QUALIFICATIONS

INTERNATIONAL AS
COMPUTER SCIENCE
CS01

Paper 1 Programming

Mark scheme

Specimen

Version: 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 Examiner.

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

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

How to mark

Aims

When you are marking your allocation of scripts your main aims should be to:

- recognise and identify the achievements of students
- where relevant, place students in the appropriate mark band and in the appropriate part of that mark band (high, low, middle) for **each** assessment objective
- record your judgements with brief notes, annotations and comments that are relevant to the mark scheme and make it clear to other associates how you have arrived at the numerical mark awarded for each assessment objective
- ensure comparability of assessment for all students, regardless of question or examiner.

Approach

It is important to be **open minded** and **positive** when marking scripts.

The specification recognises the variety of experiences and knowledge that students will have. It encourages them to study computer science in a way that is relevant to them. The questions have been designed to give them opportunities to discuss what they have found out about computer science. It is important to assess the quality of **what the student offers**.

Do not mark scripts based on the answer **you** would have written. The mark schemes have been composed to assess **quality of response** and not to identify expected items of knowledge.

Assessment Objectives

This component requires students to:

AO1: Demonstrate knowledge and understand of the key concepts and principles of computer science.

AO2: Apply knowledge and understanding of key concepts and principles of computer science.

AO3: Analyse problems in computational terms in order to develop and test programmed solutions and demonstrate an understanding of programming concepts.

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 in some questions a specific error made by a student, if repeated, could result in the student failing to achieve multiple marks. The **DPT** label indicates that this mistake should result in a student not achieving only 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.

Question	Part	Marking guidance	Total marks
01	**1**	Item Code; **I.** missing space, minor misspelling	**1** **AO3 = 1**

Question	Part	Marking guidance	Total marks
01	**2**	Total**;** **I.** minor misspelling	**1** **AO3 = 1**

Question	Part	Marking guidance	Total marks
01	**3**	A loop // code is repeated;	**1** **AO3 = 1**

Question	Part	Marking guidance	Total marks
02		Problem definition; Requirements specification // list of objectives; Feedback about requirements specification from end user; Data model / ER diagram; Analysis data dictionary; Interviews; Questionnaires; Observations; Examination of documents; Research existing solutions; Acceptable limitations / constraints; **Max 2**	**2** **AO3 = 2**

<table>
<tr><th>Question</th><th>Part</th><th>Marking guidance</th><th>Total marks</th></tr>
<tr><td>03</td><td>1</td><td>
<table>
<tr><th>numstr</th><th>result</th><th>val</th><th>position</th></tr>
<tr><td>10111</td><td>0</td><td>1</td><td>4</td></tr>
<tr><td></td><td>1</td><td>2</td><td>3</td></tr>
<tr><td></td><td>3</td><td>4</td><td>2</td></tr>
<tr><td></td><td>7</td><td>8</td><td>1</td></tr>
<tr><td></td><td></td><td>16</td><td>0</td></tr>
<tr><td></td><td>23</td><td>32</td><td>-1</td></tr>
</table>

1 mark: <code>numstr</code> initialised to <code>10111</code>, <code>result</code> initialised to <code>0</code>, <code>val</code> initialised to <code>1</code>, <code>position</code> initialised to <code>4</code>
1 mark: <code>result</code> changes to <code>1</code>, <code>val</code> changes to <code>2</code>, <code>position</code> changes to <code>3</code>
1 mark: <code>result</code> changes to <code>3</code> then <code>7</code>, <code>val</code> changes to <code>4</code> then <code>8</code>, <code>position</code> changes to <code>2</code> then <code>1</code>
1 mark: <code>result</code> changes to <code>23</code>, <code>val</code> changes to <code>16</code> then <code>32</code>, <code>position</code> changes to <code>0</code> then <code>-1</code>

Max 3 if any incorrect values in table
</td><td>4

AO3 = 4</td></tr>
</table>

Question	Part	Marking guidance	Total marks
03	2	Converts a (binary) number to decimal / denary;	**1** **AO3 = 1**

Question	Part	Marking guidance	Total marks
04		**Mark against whichever method gives the highest mark.** **Method 1** 1. Check the queue is not already full; 2. Compare the value of the (rear) pointer with the maximum size of the array; 3. If equal then (rear) pointer becomes zero; **A.** index of the first position in the array instead of zero 4. Otherwise, add one to the (rear) pointer; 5. Insert new item in position indicated by (rear) pointer; **Method 2** 1. Check the queue is not already full;	**5** **AO3 = 5**

		2. Compare the value of the (rear) pointer with the maximum size of the array minus one; 3. If equal then (rear) pointer becomes one; **A.** index of the first position in the array instead of one 4. Otherwise, add one to the (rear) pointer; 5. Insert new item in position indicated by (rear) pointer; **Method 3** 1. Check the queue is not already full; 2. Add one to the (rear) pointer; 3. Compare the value of the (rear) pointer with the maximum size of the array; 4. If equal then (rear) pointer becomes zero; **A.** index of the first position in the array instead of zero 5. Insert new item in position indicated by (rear) pointer; **Method 4** 1. Check the queue is not already full; 2. Add one to the (rear) pointer; 3. Compare the value of the (rear) pointer with the maximum size of the array plus one; 4. If equal then (rear) pointer becomes one; **A.** index of the first position in the array instead of one 5. Insert new item in position indicated by (rear) pointer; **Method 5** 1. Check the queue is not already full; 2. Add one to the (rear) pointer; 3. Use modulus/modulo operator/function with new value of (rear) pointer; 4. Use modulus/modulo operator/function with maximum size of array; 5. Insert new item in position indicated by (rear) pointer; **Max 4** if any errors	

Question	Part	Marking guidance	Total marks
05	**1**	1. Suitable prompt asking user to enter the test score. 2. User input assigned to appropriate variable. 3. Use of selection structure. 4. One of the four levels (distinction, merit, pass, fail) output under the correct circumstances. 5. All four of the levels output under the correct circumstances.	**8** **AO3 = 8**

6. At least one correct condition to detect an invalid percentage and error message displayed. **I**. message does not match question paper

7. User required to re-enter score if entered percentage is invalid (must work for both below and above 0).

8. Error message displayed if user input is not a number or user required to re-enter score if not a number. **I**. message does not match question paper

Max 7 if code contains any errors

Exemplar Solutions

Python

```
valid = False
while valid == False:
    try:
        score = int(input("Enter percentage score: "))
        if score < 0 or score > 100:
            print("Invalid percentage")
        else:
            valid = True
    except:
        print("Not a number")
if score >= 80:
    print("Distinction")
elif score >= 60:
    print("Merit")
elif score >= 40:
    print("Pass")
else:
    print("Fail")
```

C#

```
int score;
bool valid;
do
{
    Console.Write("Enter percentage score: ");
    valid = int.TryParse(Console.ReadLine(), out score);
    if (!valid)
    {
        Console.WriteLine("Not a number");
    }
    else if (score < 0 || score > 100)
    {
        Console.WriteLine("Invalid percentage");
        valid = false;
    }
} while (!valid);
if (score >= 80) Console.WriteLine("Distinction");
else if (score >= 60) Console.WriteLine("Merit");
else if (score >= 40) Console.WriteLine("Pass");
else Console.WriteLine("Fail");
```

VB.Net

```
Dim Score As Integer
Do
    Console.Write("Enter percentage score: ")
Try
    Score = Convert.ToInt32(Console.ReadLine())
    If Score < 0 Or Score > 100 Then
        Console.WriteLine("Invalid percentage")
    End If
Catch
    Console.WriteLine("Not a number")
    Score = -1
End Try
Loop Until Score >= 0 And Score <= 100
If Score < 40 Then
    Console.WriteLine("Fail")
ElseIf Score < 60 Then
    Console.WriteLine("Pass")
ElseIf Score < 80 Then
    Console.WriteLine("Merit")
Else
    Console.WriteLine("Distinction")
End If
```

Question	Part	Marking guidance	Total marks

05	2	*Evidence must match code from **05.1**, including prompts matching those in code. Code for **05.1** must be sensible.*
		
		Test evidence shows:
		
		• `84` input and `Distinction` output
		• `60` input and `Merit` output
		• `120` input and message output to say percentage invalid or user forced to re-input score. **I.** if message does not match question paper as long as it matches ***05.1***
		• `Hippo` input and message output to say not a number or user forced to re-input score. **I.** if message does not match question paper as long as it matches ***05.1***
		
		Exemplar Test Results
		
		`Enter percentage score: 84`
		`Distinction`
		`>>>`
		`== RESTART: testscore.py ==`
		`Enter percentage score: 60`
		`Merit`
		`>>>`
		`== RESTART: testscore.py ==`
		`Enter percentage score: 120`
			1
			
			AO3 = 1

		Invalid percentage Enter percentage score: Hippo Not a number	

Question	Part	Marking guidance	Total marks

06	1	1. Suitable prompts asking user to enter forename and surname of student and these values assigned to appropriate variables. **I.** order these are input in
		
		2. Use of a method such as character indexing or substring to isolate at least one character.
		
		3. First two letters of forename isolated.
		
		4. First and last letters of surname isolated.
		
		5. A random number is generated.
		
		6. Random number is in correct range.
		
		7. Program displays an identity code and stores it in the variable `idcode`. **I.** minor misspellings of `idcode` **I.** if the identity code is correct or not
		
		8. Program always either displays or stores a valid identity code for the entered forename and surname.
		
		DPT. mixing up forename and surname
		
		Max 7 if code contains any errors
		
		<u>Exemplar Solutions</u>
		
		Python
		
		`import random`
		`forename = input("Enter forename: ")`
		`surname = input("Enter surname: ")`
		`idcode = forename[0] + forename[1] +`
		`surname[0] + surname[(len(surname) - 1)]`
		`number = random.randint(100,999)`
		`idcode = idcode + str(number)`
		`print(idcode)`
		
		C#
		
		`Console.Write("Enter forename: ");`
		`string forename = Console.ReadLine();`
			8
			
			AO3 = 8

		C#
		`Console.Write("Enter surname: ");`
		`string surname = Console.ReadLine();`
		`string idcode = forename.Substring(0, 2) + surname[0] + surname[surname.Length - 1];`
		`Random stream = new Random();`
		`idcode += stream.Next(100, 1000).ToString();`
		`Console.WriteLine(idcode);`
		
		VB.Net
		
		`Console.Write("Enter forename: ")`
		`Dim Forename As String = Console.ReadLine()`
		`Console.Write("Enter surname: ")`
		`Dim Surname As String = Console.ReadLine()`
		`Dim IDCode As String = Forename.Substring(0, 2) + Surname(0) + Surname(Surname.Length - 1)`
		`Dim Stream As Random = New Random()`
		`Dim Number As Integer = Stream.Next(100, 1000)`
		`IDCode = IDCode + Number.ToString`
		`Console.WriteLine(IDCode)`	

Question	Part	Marking guidance	Total marks

06	**2**	*Evidence must match code from **06.1**, including prompts matching those in code. Code for **06.1** must be sensible.*
		
		Test evidence shows:
		
		• `DYLAN` and `WINSER` input
		• A seven-character identity code output that starts with `DYWR` and ends with a three-digit number.
		
		Exemplar Test Results
		
		`Enter forename: DYLAN`
		`Enter surname: WINSER`
		`DYWR440`
			1
			
			AO3 = 1

Question	Part	Marking guidance	Total marks
07	**1**	1. Suitable data structure created to store English or Spanish words. 2. Suitable data structure(s) created to store both English and Spanish words and correct values stored in them. **I**. minor spelling errors	

07	**1**	3. Suitable prompt asking user to input an English word and this word assigned to an appropriate variable.
		
		4. Corresponding Spanish word found and displayed, if in list.
		
		5. Message displayed to indicate word not in list, if it is not. **I**. message does not match question paper.
		
		The following mark points should only be awarded if the binary search method is used:
		
		6. Use of variables to indicate both start and end of region currently being searched. **I.** if incorrect values stored
		
		7. Start and end of region variables assigned correct initial values.
		
		8. Middle of region correctly calculated for odd and even length regions.
		
		9. Comparison made between value at calculated middle of list and English word entered. **I.** if middle incorrectly calculated
		
		10. Variable storing start of search region updated to value of `middle + 1` or `middle` if English word being searched for is after the middle of the region. **I.** if middle incorrectly calculated
		
		11. Variable storing end of search region updated to value of `middle - 1` or `middle` if English word being searched for is before the middle of the region. **I.** if middle incorrectly calculated
		
		Max 10 if code contains any errors
		
		<u>Exemplar Solutions</u>
		
		Python
		
		`dictionary = [ ["apple", "manzana"],`
		`["cat", "gato"],`
		`["food", "alimento"],`
		`["moon", "luna"],`
		`["paint", "pintar"],`
		`["school", "escuela"],`
		`["water", "agua"]]`
		`searchword = input("English word: ")`
		`start = 0`
		`end = len(dictionary) - 1`
		`finished = False`
		`while finished == False:`
		`middle = (start + end) // 2`
		`if dictionary[middle][0] == searchword:`
		`print("The Spanish word is",`
		`dictionary[middle][1])`
		`finished = True`
			11
			
			AO3 = 11

```
    elif dictionary[middle][0] > searchword:
        end = middle - 1
    else:
        start = middle + 1
    if end < start:
        print("Word not known")
        finished = True
```

C#

```
string[,] dictionary = { { "apple", "manzana" },
    { "cat", "gato" },
    { "food", "alimento" },
    { "moon", "luna" },
    { "paint", "pintar" },
    { "school", "escuela" },
    { "water", "agua" } };
Console.Write("English word: ");
string searchWord = Console.ReadLine();
int start = 0;
int end = dictionary.Length - 1;
bool finished = false;
while (finished == false)
{
    int middle = (start + end) / 2;
    if (dictionary[middle, 0] == searchWord)
    {
        Console.WriteLine("The Spanish word is " +
dictionary[middle, 1]);
        finished = true;
    }
    else if (string.Compare(dictionary[middle, 0], searchWord)
> 0)
    {
        end = middle - 1;
    }
    else
    {
        start = middle + 1;
    }
    if (end < start)
    {
        Console.WriteLine("Word not known");
        finished = true;
    }
}
```

VB.Net

```
Structure WordPair
    Dim English As String
    Dim Spanish As String
End Structure

Sub Main()
    Dim Dictionary() As WordPair = New WordPair(6) {}
```

```
    Dictionary(0).English = "apple"
    Dictionary(0).Spanish = "manzana"
    Dictionary(1).English = "cat"
    Dictionary(1).Spanish = "gato"
    Dictionary(2).English = "food"
    Dictionary(2).Spanish = "alimento"
    Dictionary(3).English = "moon"
    Dictionary(3).Spanish = "luna"
    Dictionary(4).English = "paint"
    Dictionary(4).Spanish = "pintar"
    Dictionary(5).English = "school"
    Dictionary(5).Spanish = "escuela"
    Dictionary(6).English = "water"
    Dictionary(6).Spanish = "aqua"
    Console.Write("English word: ")
    Dim SearchWord As String = Console.ReadLine()
    Dim ListStart As Integer = 0
    Dim ListEnd As Integer = Dictionary.Length - 1
    Dim Middle As Integer
    Dim Found As Boolean = False
    Do
        Middle = (ListStart + ListEnd) \ 2
        If Dictionary(Middle).English > SearchWord Then
            ListEnd = Middle - 1
        ElseIf Dictionary(Middle).English < SearchWord Then
            ListStart = Middle + 1
        Else
            Found = True
        End If
    Loop Until ListEnd < ListStart Or Found
    If Found Then
        Console.WriteLine("The Spanish word is " &
Dictionary(Middle).Spanish)
    Else
         Console.WriteLine("Word not known")
    End If
End Sub
```

Question	Part	Marking guidance	Total marks
07	2	*Evidence must match code from **07.1**, including prompts matching those in code. Code for **07.1** must be sensible.* Test evidence shows: • `moon` input and `luna` output • `cat` input and `gato` output • `paint` input and `pintar` output • `ball` input and message indicating word not known output. **I.** if message does not match question paper as long as it matches ***07.1***	**1** **AO3 = 1**

		Exemplar Test Results
		
		`English word: moon`
		`The spanish word is luna`
		`>>>`
		`= RESTART: binarysearch.py`
		`English word: cat`
		`The spanish word is gato`
		`>>>`
		`= RESTART: binarysearch.py`
		`English word: paint`
		`The spanish word is pintar`
		`>>>`
		`= RESTART: binarysearch.py`
		`English word: ball`
		`Word not found`	

Question	Part	Marking guidance	Total marks
08	**1**	1. Suitable prompts asking user to enter the number of numbers in the list followed by user input being assigned to appropriate variable. **R.** if inside or after iterative structure 2. Use of loop that repeats a number of times determined by the first number entered by the user. 3. Correct number of numbers obtained from the user. 4. Use of appropriate data structure(s) to store frequencies. 5. Adds one to correct frequency count for first number input. 6. Data structure stores correct frequencies of all numbers input. 7. Selection structure, inside iterative structure, that correctly compares calculated frequency (**I.** incorrect frequency) of a number with the lowest frequency found so far. 8. Use of loop and selection structure or function such as `count` to attempt to identify if there is more than one least-frequent number. 9. Selection structure that either outputs a calculated number (**I.** incorrectly calculated) or a message saying "`More than one least-frequent number`". **A.** any suitable message 10. Program outputs the correct least-frequent number. **I.** if done when there is more than one least-frequent number 11. Program outputs the correct least-frequent number count. **I.** if done when there is more than one least-frequent number	**12** **AO3 = 12**

12. Program displays message to indicate that there is more than one least-frequent number if and only if this is the case.

Max 11 if code contains any errors

Exemplar Solutions

Python

```
list_len = int(input("How many items in list? "))
num_count = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
for i in range(list_len):
    num = int(input("Type a number: "))
    num_count[num] = num_count[num] + 1

least_count = max(num_count)
for i in range(10):
    if num_count[i] <= least_count and num_count[i] > 0:
        least_count = num_count[i]
        least_pos = i

num_leasts = num_count.count(least_count)
if num_leasts == 1:
    print("Least:", least_pos)
    print("Count:", least_count)
else:
    print("More than one least-frequent number")
```

C#

```
Console.Write("How many items in list? ");
int listLen = int.Parse(Console.ReadLine());
int[] numCount = new int[10];
for (int i = 0; i < listLen; i++)
{
    Console.Write("Type a number: ");
    int num = int.Parse(Console.ReadLine());
    numCount[num] += 1;
}

int leastCount = int.MaxValue;
int leastPos = -1;
bool multiple = false;
for (int i = 0; i < 10; i++)
{
    if (numCount[i] < leastCount && numCount[i] > 0)
    {
        leastCount = numCount[i];
        leastPos = i;
        multiple = false;
    }
    else if (numCount[i] == leastCount)
    {
        multiple = true;
```

```
    }
}

if (multiple)
{
    Console.WriteLine("More than one least-frequent number");
}
else
{
    Console.WriteLine("Least: " + leastPos);
    Console.WriteLine("Count: " + leastCount);
}
```

VB.Net

```
Console.Write("How many items in list? ")
Dim ListLen As Integer = Convert.ToInt32(Console.ReadLine())
Dim NumCount As New List(Of Integer)({0, 0, 0, 0, 0, 0, 0, 0,
0, 0})
For i = 1 To ListLen
    Console.Write("Type a number: ")
    Dim Num As Integer = Convert.ToInt32(Console.ReadLine())
    NumCount(Num) += 1
Next

Dim LeastCount = 10000
Dim LeastPos = -1
For i = 0 To 9
    If NumCount(i) <= LeastCount And NumCount(i) > 0 Then
        LeastCount = NumCount(i)
        LeastPos = i
    End If
Next

If NumCount.IndexOf(LeastCount) =
NumCount.LastIndexOf(LeastCount) Then
    Console.WriteLine("Least:" & LeastPos)
    Console.WriteLine("Count:" & LeastCount)
Else
    Console.WriteLine("More than one least-frequent number")
End If
```

Question	Part	Marking guidance	Total marks
08	**2**	*Evidence must match code from **08.1**, including prompts matching those in code. Code for **08.1** must be sensible.* Test evidence shows: • the number 6 being entered followed by the numbers 5, 3, 4, 5, 4, 5 (**I.** order of these six numbers) and then a message displayed saying 3 is the least-frequent number with a count of 1	**1** **AO3 = 1**

		• the number 7 being entered followed by the numbers 8, 2, 2, 7, 8, 7, 8 (**I.** order of these seven numbers) and then a message displayed saying that there is more than one least-frequent number (**I.** if message does not match question paper as long as it matches ***09.1***, **I.** if numbers also displayed).
		
		Exemplar Test Results
		
		`How many items in list? 6`
		`Type a number: 5`
		`Type a number: 3`
		`Type a number: 4`
		`Type a number: 5`
		`Type a number: 4`
		`Type a number: 5`
		`Least: 3`
		`Count: 1`
		`>>>`
		`= RESTART: leastfrequent.py`
		`How many items in list? 7`
		`Type a number: 8`
		`Type a number: 2`
		`Type a number: 2`
		`Type a number: 7`
		`Type a number: 8`
		`Type a number: 7`
		`Type a number: 8`
		`More than one least-frequent number`	

Question	Part	Marking guidance	Total marks
09	1	**For the menu and key:** 1. Menu displayed containing all four / five options. **I**. order or options 2. User input of option to execute and selection statement uses user input to determine block of code to run. **I**. not all options implemented 3. Loop repeatedly displays menu and executes selected option until exit option chosen by user, at which point program terminates. 4. Key value input and stored in a variable. **For one of encryption or decryption:** 5. Loop iterates through each character in plaintext or ciphertext.	**15** **AO3 = 15**

6. Program attempts to move at least one character backward or forwards in alphabet.

7. Program correctly moves all characters along alphabet the number of places indicated by the key – forward for encryption or backward for decryption. **I.** if does not work for letters that would wrap around at start/end of alphabet

8. Wrapping of letters around start or end of alphabet works correctly.

For the other one of encryption or decryption:

9. Program correctly moves all characters along alphabet the number of places indicated by the key – forward for encryption or backward for decryption. **I.** if does not work for letters that would wrap around at start/end of alphabet

10. Wrapping of letters around start or end of alphabet works correctly.

For cracking:

11. Loop iterates correctly through key values 1 to 25.

12. For each key value the correct plaintext is displayed. **I**. key values iterated through are incorrect

For program structure:

13. At least one user-created subroutine created and called, which has an appropriate meaningful name.

14. Subroutines created for all program options that the solution tackles (must be at least two subroutines).

15. Key is returned from subroutine that inputs it and passed as a parameter to other subroutines that use the key. Global variables are not used.

Max 14 if code contains any errors

<u>Exemplar Solutions</u>

Python

```
def input_key():
    key = int(input("Enter the key: "))
    return key

def encrypt_message(key):
    plaintext = input("Enter the message to
encrypt: ")
    ciphertext = ""
```

```
    for pos in range(len(plaintext)):
        letter_code = ord(plaintext[pos])
        letter_code += key
        if letter_code > ord('Z'):
            letter_code -= 26;
        ciphertext += chr(letter_code)
    print("The encrypted message is", ciphertext)

def decrypt_message(key):
    ciphertext = input("Enter the message to
decrypt: ")
    plaintext = ""
    for pos in range(len(ciphertext)):
        letter_code = ord(ciphertext[pos])
        letter_code -= key
        if letter_code < ord('A') :
            letter_code += 26;
        plaintext += chr(letter_code)
    print("The decrypted message is", plaintext)

def crack_message():
    ciphertext = input("Enter the message to
decrypt: ")
    for key in range(1, 26):
        plaintext = ""
        for pos in range(len(ciphertext)):
            letter_code = ord(ciphertext[pos])
            letter_code -= key
            if letter_code < ord('A') :
                letter_code += 26;
            plaintext += chr(letter_code)
        print("Possible plaintext", key,
plaintext)

option = 0
while option != 5:
    print("1. Input key")
    print("2. Encrypt message")
    print("3. Decrypt message")
    print("4. Crack message")
    print("5. Exit")
    option = int(input("Select option: "))
    if option == 1:
        key = input_key()
    elif option == 2:
        encrypt_message(key)
    elif option == 3:
        decrypt_message(key)
    elif option == 4:
        crack_message()
```

C#

```
class Program
{
    static int InputKey()
    {
        Console.Write("Enter the key: ");
        int key = int.Parse(Console.ReadLine());
        return key;
    }

    static void EncryptMessage(int key)
    {
        Console.Write("Enter the message to encrypt: ");
        string plainText = Console.ReadLine();
        string cipherText = "";
        for (int pos = 0; pos < plainText.Length; pos++)
        {
            int letterCode = plainText[pos];
            letterCode += key;
            if (letterCode > 90) letterCode -= 26;
            cipherText += (char)letterCode;
        }
        Console.WriteLine("The encrypted message is " +
cipherText);
    }

    static void DecryptMessage(int key)
    {
        Console.Write("Enter the message to decrypt: ");
        string cipherText = Console.ReadLine();
        string plainText = "";
        for (int pos = 0; pos < cipherText.Length; pos++)
        {
            int letterCode = cipherText[pos];
            letterCode -= key;
            if (letterCode < 65) letterCode += 26;
            plainText += (char)letterCode;
        }
        Console.WriteLine("The decrypted message is " +
plainText);
    }

    static void CrackMessage()
    {
        Console.Write("Enter the message to decrypt: ");
        string cipherText = Console.ReadLine();
        for (int key = 1; key < 26; key++)
        {
            string plainText = "";
            for (int pos = 0; pos < cipherText.Length; pos++)
            {
                int letterCode = cipherText[pos];
                letterCode -= key;
                if (letterCode < 65) letterCode += 26;
                plainText += (char)letterCode;
            }
        Console.WriteLine("Possible plaintext " + key + " " +
plainText);
        }
```

```
    }

    static void Main(string[] args)
    {
        int option;
        int key = 0;
        do
        {
            Console.WriteLine("1. Input key");
            Console.WriteLine("2. Encrypt message");
            Console.WriteLine("3. Decrypt message");
            Console.WriteLine("4. Crack message");
            Console.WriteLine("5. Exit");
            Console.Write("Select option: ");
            option = int.Parse(Console.ReadLine());
            if (option == 1) key = InputKey();
            else if (option == 2) EncryptMessage(key);
            else if (option == 3) DecryptMessage(key);
            else if (option == 4) CrackMessage();
        } while (option != 5);
    }
}
```

VB.Net

```
Dim Alphabet As New List(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"})

Function InputKey() As Integer
    Console.Write("Enter the key: ")
    Dim Key As Integer = Convert.ToInt32(Console.ReadLine())
    Return Key
End Function

Sub EncryptMessage(Key As Integer)
    Console.Write("Enter the message to encrypt: ")
    Dim PlainText As String = Console.ReadLine()
    Dim CipherText As String = ""
    For Pos = 0 To PlainText.Length - 1
        Dim Letter As Char = PlainText(Pos)
        Dim LetterCode As Integer = Alphabet.IndexOf(Letter)
        LetterCode += Key
        If LetterCode >= 26 Then
            LetterCode -= 26
        End If
        CipherText += Alphabet(LetterCode)
    Next
    Console.WriteLine("The encrypted message is " & CipherText)
End Sub

Sub DecryptMessage(Key As Integer)
    Console.Write("Enter the message to decrypt: ")
    Dim CipherText As String = Console.ReadLine()
    Dim PlainText As String = ""
    For Pos = 0 To CipherText.Length - 1
        Dim Letter As Char = CipherText(Pos)
```

```
        Dim LetterPos As Integer = Alphabet.IndexOf(Letter)
        LetterPos -= Key
        If LetterPos < 0 Then
            LetterPos += 26
        End If
        PlainText += Alphabet(LetterPos)
    Next
    Console.WriteLine("The decrypted message is " & PlainText)
End Sub

Sub CrackMessage()
    Console.Write("Enter the message to decrypt: ")
    Dim CipherText As String = Console.ReadLine()
    For Key = 1 To 25
        Dim PlainText As String = ""
        For Pos = 0 To CipherText.Length - 1
            Dim Letter As Char = CipherText(Pos)
            Dim LetterPos As Integer = Alphabet.IndexOf(Letter)
            LetterPos -= Key
            If LetterPos < 0 Then
                LetterPos += 26
            End If
            PlainText += Alphabet(LetterPos)
        Next
        Console.WriteLine("Possible plaintext " & Key & " " &
PlainText)
    Next
End Sub

Sub Main()
    Dim MenuOption As Integer
    Dim Key As Integer
    Do
        Console.WriteLine("1. Input key")
        Console.WriteLine("2. Encrypt message")
        Console.WriteLine("3. Decrypt message")
        Console.WriteLine("4. Crack message")
        Console.WriteLine("5. Exit")
        Console.Write("Selebct option: ")
        MenuOption = Console.ReadLine()
        If MenuOption = 1 Then
            Key = InputKey()
        ElseIf MenuOption = 2 Then
            EncryptMessage(Key)
        ElseIf MenuOption = 3 Then
            DecryptMessage(Key)
        ElseIf MenuOption = 4 Then
            CrackMessage()
        End If
    Loop Until MenuOption = 5
    Console.ReadLine()
End Sub
```

Question	Part	Marking guidance	Total marks

09	2	*Evidence must match code from **09.1**, including prompts matching those in code. Code for **09.1** must be sensible.*
		
		1 mark: Test evidence shows:
		
		• `PLUM` input and `TPYQ` output **I.** key not visible
		• `UDLUXJ` input and `OXFORD` output **I.** key not visible
		
		1 mark: Test evidence shows:
		
		• `WIGJONYL` input and all 25 possible plaintexts shown.
		
		Exemplar Test Results
		
		`1. Input key`
		`2. Encrypt message`
		`3. Decrypt message`
		`4. Crack message`
		`5. Exit`
		`Select option: 1`
		`Enter the key: 4`
		`1. Input key`
		`2. Encrypt message`
		`3. Decrypt message`
		`4. Crack message`
		`5. Exit`
		`Select option: 2`
		`Enter the message to encrypt: PLUM`
		`The encrypted message is TPYQ`
		`1. Input key`
		`2. Encrypt message`
		`3. Decrypt message`
		`4. Crack message`
		`5. Exit`
		`Select option: 1`
		`Enter the key: 6`
		`1. Input key`
		`2. Encrypt message`
		`3. Decrypt message`
		`4. Crack message`
		`5. Exit`
		`Select option: 3`
		`Enter the message to decrypt: UDLUXJ`
		`The decrypted message is OXFORD`
		`1. Input key`
		`2. Encrypt message`
		`3. Decrypt message`
		`4. Crack message`
			2
			
			AO3 = 2

```
5. Exit
Select option: 4
Enter the message to decrypt: WIGJONYL
Possible plaintext 1 VHFINMXK
Possible plaintext 2 UGEHMLWJ
Possible plaintext 3 TFDGLKVI
Possible plaintext 4 SECFKJUH
Possible plaintext 5 RDBEJITG
Possible plaintext 6 QCADIHSF
Possible plaintext 7 PBZCHGRE
Possible plaintext 8 OAYBGFQD
Possible plaintext 9 NZXAFEPC
Possible plaintext 10 MYWZEDOB
Possible plaintext 11 LXVYDCNA
Possible plaintext 12 KWUXCBMZ
Possible plaintext 13 JVTWBALY
Possible plaintext 14 IUSVAZKX
Possible plaintext 15 HTRUZYJW
Possible plaintext 16 GSQTYXIV
Possible plaintext 17 FRPSXWHU
Possible plaintext 18 EQORWVGT
Possible plaintext 19 DPNQVUFS
Possible plaintext 20 COMPUTER
Possible plaintext 21 BNLOTSDQ
Possible plaintext 22 AMKNSRCP
Possible plaintext 23 ZLJMRQBO
Possible plaintext 24 YKILQPAN
Possible plaintext 25 XJHKPOZM
```