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

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

Cambridge will not enter into discussions about these mark schemes.

Cambridge is publishing the mark schemes for the October/November 2016 series for most Cambridge IGCSE®, Cambridge International A and AS Level components and some Cambridge O Level components.
1 (a) 1 mark for both Set code entered correct. 1 mark for each label.  

(b) (i) 1 mark per bullet to max 3

- Method header
- initialising Code to ""
- initialising State to "Open-NoCode"
  
edg.

**PYTHON:**

```python
def __init__(self):
    self.__code = ""
    self.__state = "Open-NoCode"
```

**PASCAL/DELPHI:**

```plaintext
constructor SafetyDepositBox.Create();
begin
    Code := '';
    State := 'Open-NoCode';
end;
```
VB:
Public Sub New()
    Code = ""
    State = "Open-NoCode"
End Sub

(ii) 1 mark per bullet to max 2
• method header
• Setting code to ""
e.g.

PYTHON:
    def reset(self):
        self.__code = ""

PASCAL/DELPHI:
procedure SafetyDepositBox.Reset();
begin
    Code := '';
end;

VB:
Public Sub Reset()
    Code = ""
End Sub

(iii) 1 mark per bullet to max 2
• method header with parameter
• setting state to parameter value
• Outputting state
e.g.

PYTHON:
    def SetState(self,NewState):
        self.__state = NewState
        print(self.__state)

PASCAL/DELPHI:
Procedure SetState(NewState : String);
begin
    State := NewState
    WriteLn(State)
end;
(iv) 1 mark per bullet to max 2

- setting code to **parameter**
- Outputting New cost set and code
e.g.

**PYTHON:**
```python
def SetNewCode(self, NewCode):
    self.__code = NewCode
    print("New code set: ", self.__code)
```

**PASCAL/DELPHI:**
```pascal
procedure SetNewCode(NewCode : String);
begin
    Code := NewCode;
    WriteLn('New code set: ', Code)
end;
```

**VB:**
```vb
Public Sub SetNewCode(NewCode)
    Code = NewCode
End Sub
```
(v) 1 mark per bullet to max 4

- function header taking string parameter, returns Boolean
- check length of string is 4
- check each character is a digit
- return of correct Boolean value for both cases

e.g

**PYTHON:**
def __valid(self, s):
digits = ['0','1','2','3','4','5','6','7','8','9']
isValid = False
if (len(s) == 4):
    if (s[0] in digits) & (s[1] in digits) & (s[2] in digits) &
        (s[3] in digits):
        isValid = True
    return(isValid)

**PASCAL/DELPHI:**

function Valid(s : string) : Boolean;
var isValid : Boolean; i : integer;
begin
    isValid := False
    if Length(s) = 4
    then
        begin
            isValid := True;
            For i := 1 to 4 do
                if (s[i] < '0') OR (s[i] > '9')
                then
                    isValid := False;
        end;
end;

**VB:** *ByVal optional*

Public Function valid(ByVal s As String) As Boolean
    If s Like "#####" Then
        Return True
    Else
        Return False
    End If
End Function
(vi) 1 mark per bullet to max 12

- read Chars from keyboard
- check if 'R' and state = Open-CodeSet
  - call method Reset() & method SetState
- if Chars is the set code:
  - check if locked
    - set state to Open-CodeSet
  - else if closed
    - then set state to Locked
- if Chars is empty and State is “Open-CodeSet” then setState to closed
- if Chars is a valid 4-digit code and state is Open-NoCode
  - call setNewCode and setState
- outputting correct error messages for not valid 4-digit and state is not Open-NoCode
  e.g.

PYTHON:

def StateChange(self):
    Chars = input("Enter code: ")
    if Chars == "R":
        if self.__state == "Open-CodeSet":
            self.reset()
            self.SetState("Open-NoCode")
        elif Chars == self.__code:
            if self.__state == "Locked":
                self.SetState("Open-CodeSet")
            elif self.__state == "Closed":
                self.SetState("Locked")
            elif (Chars == "") & (self.__state == "Open-CodeSet"):
                self.SetState("Closed")
            else:
                self.__valid(Chars):
                    if self.__state == "Open-NoCode":
                        self.SetNewCode(Chars)
                        self.SetState("Open-CodeSet")
                    else:
                        print("Error - does not match set code")
                else:
                    print("Error - Code format incorrect")
PASCAL/DELPHI:
Procedure StateChange();
var Chars : String;
begin
ReadLn(Chars);
If Chars = 'R' Then
  If State = 'Open-CodeSet' Then
    begin
      Reset();
      SetState('Open-NoCode');
    end
  Else
    If Chars = Code Then
      If state = 'Locked' Then
        SetState('Open-CodeSet')
      Else
        If state = 'Closed' Then
          SetState('Locked')
    Else
      If (Chars = '') AND (State = 'Open-CodeSet') Then
        SetState('Closed')
      Else
        If Valid(Chars) Then
          begin
            If State == 'Open-NoCode' Then
              begin
                SetNewCode(Chars);
                SetState('Open-CodeSet');
              end
            else
              WriteLn('Error - does not match set code')
          end
        Else
          WriteLn('Error - Code format incorrect');
  end;
VB:
Public Sub StateChange()
    Dim Chars As String
    Chars = Console.ReadLine()
    If Chars = "R" Then
        If State = "Open-CodeSet" Then
            Reset()
            SetState("Open-NoCode")
        End If
    ElseIf Chars = Code Then
        If state = "Locked" Then
            SetState("Open-CodeSet")
        ElseIf state = "Closed" Then
            SetState("Locked")
        End If
    ElseIf (Chars = "") AND (State = "Open-CodeSet") Then
        SetState("Closed")
    ElseIf Valid(Chars) Then
        If State = "Open-NoCode" Then
            SetNewCode(Chars)
            SetState("Open-CodeSet")
        Else
            Console.WriteLine("Error - does not match set code")
        End If
    Else
        Console.WriteLine("Error - Code format incorrect")
    End If
End Sub

(vii) 1 mark per bullet to max 4

- method header
- Initialising ThisSafe to instance of SafetyDepositBox
- Loop forever
- Call method StateChange on ThisSafe
  e.g.

PYTHON:
def main():
    ThisSafe = SafetyDepositBox()
    while True:
        ThisSafe.StateChange()

PASCAL/DELPHI:
var ThisSafe : SafetyDepositBox;
ThisSafe := SafetyDepositBox.Create;
while True do
    ThisSafe.StateChange;
VB:
Sub Main()
    Dim ThisSafe As New SafetyDepositBox()
    Do
        ThisSafe.StateChange()
        Loop
End Sub

(c) (i) 1 mark per bullet to max 2: [2]
• The attributes can only be accessed in the class
• Properties are needed to get/set the data // It provides/uses encapsulation
• Increase security/integrity of attributes

(ii) 1 mark per bullet [2]
• The public methods can be called anywhere in the main program // Public methods
  can be inherited by sub-classes
• The private methods can only be called within the class definition // cannot be called
  outside the class definition // Private methods cannot be inherited by sub-classes

2 (a) (i) 1 mark per feature to max 3 [3]
   e.g.
   • auto-indent
   • auto-complete / by example
   • colour-coded keywords/ strings/ comments/ built-in functions/ user-defined function
     names
   • pop-up help
   • can set indent width
   • expand/collapse subroutines/code
   • block highlighting
   incorrect syntax highlighting/underlining //dynamic syntax checker

(ii) Read and mark the answer as one paragraph. Mark a 'how' and a 'when' anywhere in
     the answer. [2]
     1 mark for when, 1 mark for how.
     e.g.
     When:
     • the error has been typed
     • when the program is being run/compiled/interpreted
     How:
     • highlights/underlines
     displays error message/pop-up

(iii)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line 3</td>
<td><strong>while (Index == -1) &amp; (Low &lt;= High):</strong></td>
<td><strong>WHILE (Index = -1) AND (Low &lt;= High):</strong></td>
<td><strong>DO WHILE (Index = -1) AND (Low &lt;= High):</strong></td>
</tr>
<tr>
<td>Line 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Line 4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(b) (i) Python: compiled/interpreted
VB.NET: compiled
Pascal: compiled/interpreted
Delphi: compiled/interpreted

(ii)

<table>
<thead>
<tr>
<th>Logic error</th>
<th>Logic error</th>
<th>Logic error</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 return(Index)</td>
<td>14 Result := Index;</td>
<td>14 BinarySearch = Index</td>
</tr>
</tbody>
</table>

(iii) 1 mark for each name, 1 for each description

- breakpoint
- a point where the program can be halted to see if the program works at this point
- stepping / step through
- executes one statement at a time and then pauses to see the effect of each statement
- variable watch window
- observe how variables changed during execution
### START:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDR #0</td>
<td>Initialise index register to zero.</td>
</tr>
<tr>
<td>LDM #0</td>
<td>Initialise COUNT to zero.</td>
</tr>
<tr>
<td>STO COUNT</td>
<td></td>
</tr>
</tbody>
</table>

#### LOOP1:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDX NAME</td>
<td>Load character from indexed address NAME.</td>
</tr>
<tr>
<td>OUT</td>
<td>Output character to screen.</td>
</tr>
<tr>
<td>INC IX</td>
<td>Increment index register.</td>
</tr>
<tr>
<td>LDD COUNT</td>
<td>Increment COUNT starts here.</td>
</tr>
<tr>
<td>INC ACC</td>
<td></td>
</tr>
<tr>
<td>STO COUNT</td>
<td></td>
</tr>
<tr>
<td>CMP MAX</td>
<td>Is COUNT = MAX?</td>
</tr>
<tr>
<td>JPN LOOP1</td>
<td>If FALSE, jump to LOOP1.</td>
</tr>
</tbody>
</table>

#### REVERSE:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC IX</td>
<td>Decrement index register.</td>
</tr>
<tr>
<td>LDM #0</td>
<td>Set ACC to zero.</td>
</tr>
<tr>
<td>STO COUNT</td>
<td>Store in COUNT.</td>
</tr>
</tbody>
</table>

#### LOOP2:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDX NAME</td>
<td>Load character from indexed address NAME.</td>
</tr>
<tr>
<td>OUT</td>
<td>Output character to screen.</td>
</tr>
<tr>
<td>DEC IX</td>
<td>Decrement index register.</td>
</tr>
<tr>
<td>LDD COUNT</td>
<td>Increment COUNT starts here.</td>
</tr>
<tr>
<td>INC ACC</td>
<td></td>
</tr>
<tr>
<td>STO COUNT</td>
<td></td>
</tr>
<tr>
<td>CMP MAX</td>
<td>Is COUNT = MAX?</td>
</tr>
<tr>
<td>JPN LOOP2</td>
<td>If FALSE, jump to LOOP2.</td>
</tr>
</tbody>
</table>

### END

// end of program

**COUNT:**

4

**MAX:**

4

**NAME:**

B01000110 // ASCII code in binary for 'F'
B01010010 // ASCII code in binary for 'R'
B01000101 // ASCII code in binary for 'E'
B01000100 // ASCII code in binary for 'D'

[Max 15]
<table>
<thead>
<tr>
<th></th>
<th>Acceptance testing</th>
<th>Integration testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Who</td>
<td>The end user // user of the software</td>
<td>The programmer / in-house testers</td>
</tr>
<tr>
<td>When</td>
<td>When the software is finished/ when it is installed</td>
<td>When the separate modules have been written and tested</td>
</tr>
<tr>
<td>Purpose</td>
<td>To ensure the software is what the customer ordered // to check that the software meets the user requirements</td>
<td>To ensure the modules work together as expected</td>
</tr>
</tbody>
</table>