Cambridge International A Level

**Cambridge International Examinations**
Cambridge International Advanced Level

**COMPUTER SCIENCE** **9608/41**

Paper 4 Written Paper **October/November 2016**

MARK SCHEME

Maximum Mark: 75

**Published**

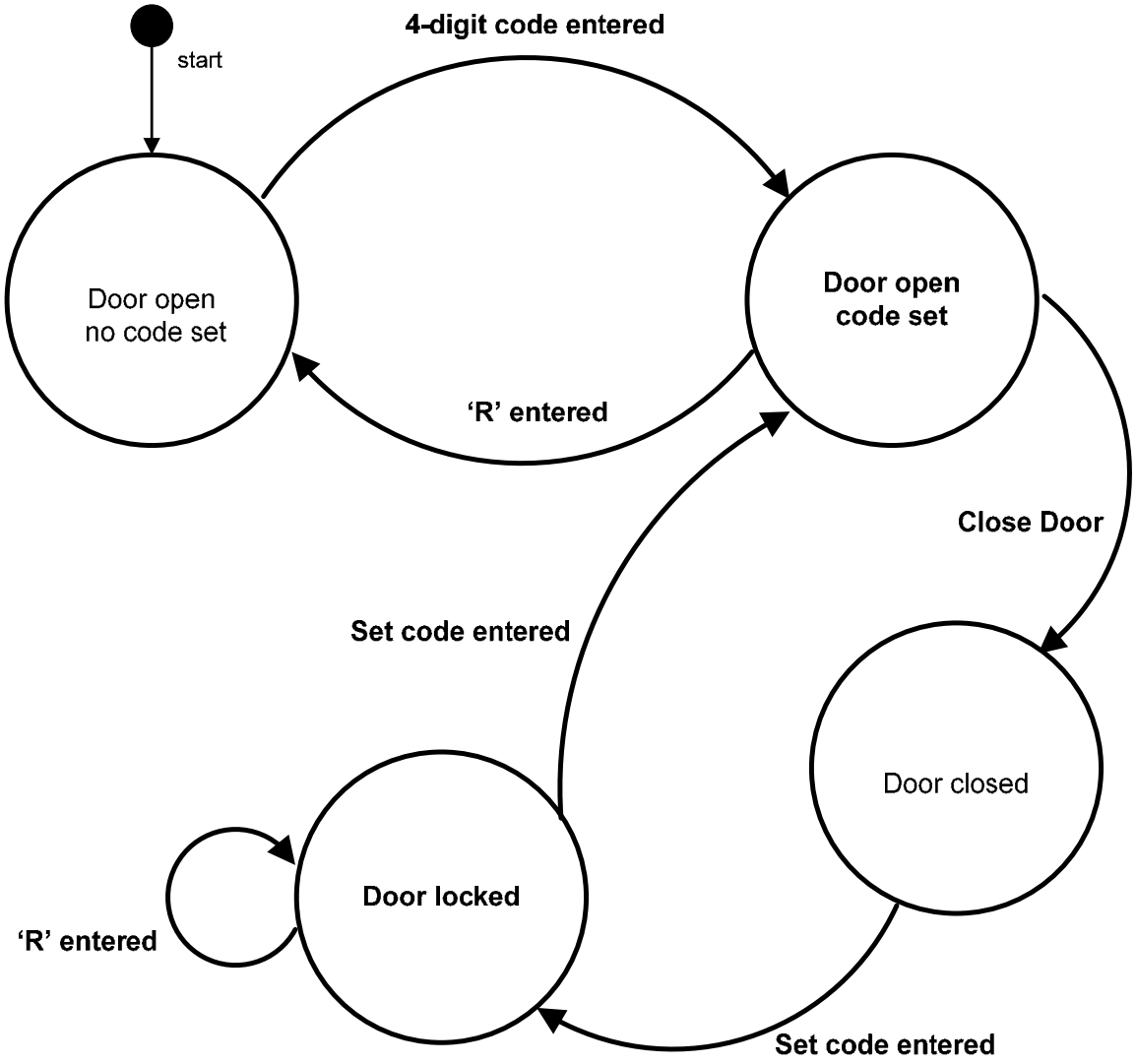
This mark scheme is published as an aid to teachers and candidates, to indicate the requirements of the examination. It shows the basis on which Examiners were instructed to award marks. It does not indicate the details of the discussions that took place at an Examiners' meeting before marking began, 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. **[7]**




**(b)** **(i)** 1 mark per bullet to max 3 **[3]**

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

e.g.

PYTHON:

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

PASCAL/DELPHI:

```
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 **[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 **[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;
```

VB:
```
Public Sub SetState(ByVal
NewState As String)
   State = NewState
Console.WriteLine(State)
End Sub
```

VB:
```
Private _State As String
   Public Property State() As
String
      Get
         Return _State
      End Get
      Set(value As String)
         _State = value
      End Set
End Property
Public Sub SetState()
   Console.WriteLine(Me.State)
End Sub
```

**(iv)** 1 mark per bullet to max 2 **[2]**

- setting code to <u>parameter</u>
- Outputting New cost set and code

e.g.

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

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

VB:
```
Public Sub SetNewCode(NewCode)
   Code = NewCode
   Console.WriteLine("New code set: " & Code)
End Sub
```

**(v)** 1 mark per bullet to max 4 **[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 **[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")
    elif 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** **[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)**

| A | B | C | |
|---|---|---|---|
| Line 3 | Line 5 | Line 4 | **[1]** |
| `while (Index == -1) & (Low <= High):` | `WHILE (Index = -1) AND (Low <= High) DO` | `DO WHILE (Index = -1) AND (Low <= High)` | **[1]** |

**(b) (i)** Python: compiled/interpreted **[1]**
VB.NET: compiled
Pascal: compiled/interpreted
Delphi: compiled/interpreted

**(ii)**

| | | | |
|---|---|---|---|
| Logic error | Logic error | Logic error | **[1]** |
| `11 return(Index)` | `14 Result := Index;` | `14 BinarySearch = Index` | **[1]** |

**(iii) 1 mark for each name, 1 for each description** **[4]**

- 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

**3**

| | | | | |
|---|---|---|---|---|
| START: | LDR | #0 | // initialise index register to zero | [1] |
| | LDM | #0 | // initialise COUNT to zero | [1] |
| | STO | COUNT | | |
| LOOP1: | LDX | NAME | // load character from indexed address NAME | [1] |
| | OUT | | // output character to screen | [1] |
| | INC | IX | // increment index register | [1] |
| | LDD | COUNT | // increment COUNT starts here | |
| | INC | ACC | | [1] |
| | STO | COUNT | | |
| | CMP | MAX | // is COUNT = MAX? | [1] |
| | JPN | LOOP1 | // if FALSE, jump to LOOP1 | [1] |
| REVERSE: | DEC | IX | // decrement index register | [1] |
| | LDM | #0 | // set ACC to zero | [1] |
| | STO | COUNT | // store in COUNT | |
| LOOP2: | LDX | NAME | // load character from indexed address NAME | [1] |
| | OUT | | // output character to screen | |
| | DEC | IX | // decrement index register | [1] |
| | LDD | COUNT | // increment COUNT starts here | |
| | INC | ACC | // | [1] |
| | STO | COUNT | // | |
| | CMP | MAX | // is COUNT = MAX? | [1] |
| | JPN | LOOP2 | // if FALSE, jump to LOOP2 | |
| | END | | // end of program | [1] |
| COUNT: | | | | |
| 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]**

**4**

| | Acceptance testing | Integration testing | |
|---|---|---|---|
| Who | The end user // user of the software | The programmer / in-house testers | **[1]** + **[1]** |
| When | When the software is finished/ when it is installed | When the separate modules have been written and tested | **[1]** + **[1]** |
| Purpose | To ensure the software is what the customer ordered // to check that the software meets the user requirements | To ensure the modules work together as expected | **[1]** + **[1]** |