Cambridge International A Level

Cambridge International Examinations
Cambridge International Advanced Level

COMPUTER SCIENCE **9608/42**

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

MARK SCHEME

Maximum Mark: 75

Published

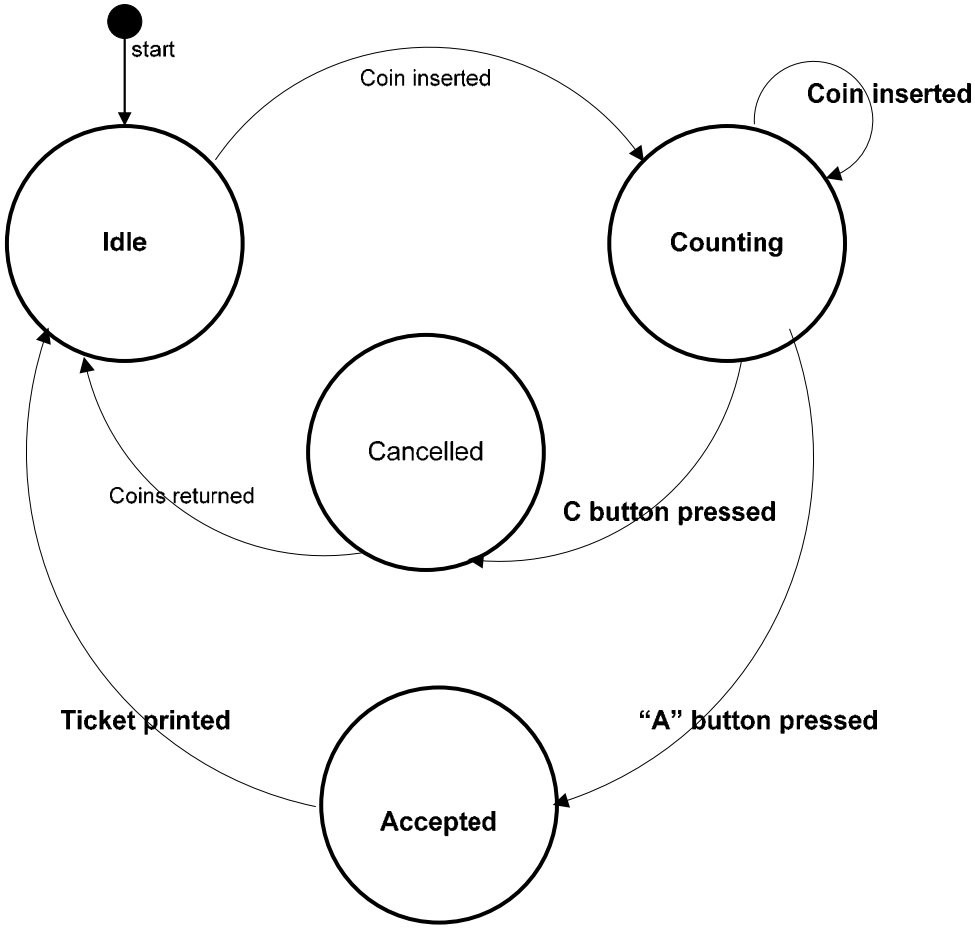
This mark scheme is published as an aid to teachers and candidates, to indicate the requirements of the examination. It shows the basis on which Examiners were instructed to award marks. It does not indicate the details of the discussions that took place at an Examiners' meeting before marking began,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



1 mark for each label

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

- method header and close
- initialising amount to 0
- initialising state to "Idle"

e.g.

PYTHON:
```
def __init__(self):
    self.__amount = 0
    self.__state = "Idle"
```

PASCAL/DELPHI:
```
constructor TicketMachine.Create();
begin
    Amount := 0;
    State := 'Idle';
end;
```

VB:
```
Public Sub New()
    Amount = 0
    State = "Idle"
End Sub
```

VB:
```
Public Sub Create()
    Amount = 0
    state = "Idle"
End Sub
```

(ii) 1 mark per bullet to max 2: [2]

- method header, close 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 TicketMachine.SetState(NewState : string);
begin
    State := NewState;
    Writeln(State);
end;
```

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

(iii) 1 mark per bullet to max 2: [2]

- output Amount
- set amount to zero

e.g.

PYTHON:
```
   def ReturnCoins(self):
      print(self.__Amount)
      self.__Amount = 0
```

PASCAL/DELPHI:
```
procedure TicketMachine.ReturnCoins();
begin
   Writeln(Amount);
   Amount := 0;
end;
```

VB:
```
Public Sub ReturnCoins()
   Console.WriteLine(Me.Amount)
   Me.Amount = 0
End Sub
```

(iv) 1 mark per bullet to max 3: [3]

- function header, take string as parameter, return Boolean
- check the parameter is a valid coin
- return of value for both cases

e.g.

PYTHON:

```
def __validCoin(self, s):
    coins = ['10','20','50','100']
    if s in coins:
        isValid = True
    else:
        isValid = False
    return(isValid)
```

PASCAL/DELPHI:

```
function TicketMachine.ValidCoin(s : string) : boolean;
begin
    if ((s = '10') or (s = '20') or (s = '50') or (s = '100'))
then
        ValidCoin:= True;
    else
        ValidCoin := False;
end;
```

VB:

```
Public Function ValidCoin(ByVal s As String) As Boolean
    If s = "10" or s = "20" or s = "50" or s = "100" Then
        ValidCoin = True
    Else
        ValidCoin = False
    End If
End Sub
```

(v) 1 mark per bullet to max 2 [2]

- Cast parameter as integer
- Add value to amount

e.g.

PYTHON:

```
def coinInserted(self, s):
    coinValue = int(s)
    self.__amount = self.__amount + coinValue
```

PASCAL/DELPHI:

```
procedure TicketMachine.CoinInserted(s : string);
var
   CoinValue, Code : integer;
begin
   Val(s, CoinValue, Code);
   Amount := Amount + CoinValue;
end;
```

VB:

```
Public Sub CoinInserted(ByVal S As String)
   CoinValue = INT(s)
   Me.Amount = Me.Amount + CoinValue
End Sub
```

(vi) 1 mark per bullet to max 12 [12]

- read NewInput from keyboard
- check if input 'C' and state = Counting
  - then set state to cancelled
  - call method returnCoins() and set state to Idle
- check if input 'A'
  - then check if amount = 0 then output no coins
  - else set state to Accepted
  - call PrintTicket method and Set state to Idle
- else if input is a valid coin
  - call CoinInserted method with NewInput as parameter
  - set state to Counting
  - error message if not a valid coin

e.g.

PYTHON:

```
def stateChange(self):
    newInput = input("Insert coin: ")
    if newInput == "C":
        if self.__state == "Counting":
            self.setState("Cancelled")
            self.returnCoins()
        self.setState("Idle")
    elif newInput == "A":
        if self.__amount == 0:
            print("no coins inserted")
        else:
            self.setState("Accepted")
            self.__PrintTicket()
        self.setState("Idle")
    elif self.__validCoin(newInput):
        self.coinInserted(newInput)
        self.setState("Counting")
    else:
        print("Error - not a valid coin")
```

PASCAL/DELPHI:

```
procedure TicketMachine.StateChange();
var
   NewInput : string;
begin
   Write('Insert coin: ');
   Readln(NewInput);
   if (NewInput = 'C') then
   begin
      if (State = 'Counting') then
      begin
         State := 'Cancelled';
         ReturnCoins();
      end;
      SetState('Idle')
   end
   else
      if (NewInput = 'A') then
```

```
        begin
           if (Amount = 0) then
              Writeln('No coins inserted')
           else
           begin
              SetState('Accepted');
              PrintTicket();
           end;
           SetState('Idle');
        end
        else
           if (ValidCoin(NewInput)) then
           begin
              CoinInserted(NewInput);
              SetState('Counting')
           end
        else
           Writeln('Error - not a valid coin')
end;
```

VB:

```
Public Sub StateChange()
   Dim NewInput As String
   NewInput = Console.Readline()
   If NewInput = "C" Then
      If State = "Counting" Then
         SetState("Cancelled")
         ReturnCoins()
      End If
      SetState("Idle")
   Elseif NewInput = "A" Then
      If Amount = 0 Then
         Console.Writeline("No coins inserted")
      Else
         SetState("Accepted")
         PrintTicket()
      Endif
         SetState("Idle")
      Elseif ValidCoin(NewInput) Then
         CoinInserted(NewInput)
         SetState("Counting")
      Else
         Console.Writeline("Error – not a valid coin")
      EndIf
End Sub
```

(vii) 1 mark per bullet to max 4 [4]

- declaration of main method header
- Initialising ParkingMeter as instance of TicketMachine
- Looping while true/until false
  - Calling stateChange method on ParkingMeter

e.g.

PYTHON:

```
def main():
    ParkingMeter = TicketMachine()
    while True:
        ParkingMeter.stateChange()
```

PASCAL/DELPHI:

```
begin
    ParkingMeter := TicketMachine.Create();
    while True do
        ParkingMeter.StateChange();
end.
```

VB:

```
Sub Main()
    Dim ParkingMeter As New TicketMachine
    ParkingMeter.Create()
    While (True)
        Call ParkingMeter.StateChange()
    End While
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 [6]

	(i) Alpha testing	**(ii) Beta testing**
Who	In house testers / developers / programmers	(potential) (end) user(s)/client(s)
When	Near the end of development // program is nearly fully-usable // after integration and before beta	Before general release of software // passed Alpha testing
Purpose	To find errors not found in earlier testing // ensure ready for beta testing	For constructive comments/ feedback // to test in real-life scenarios/situations/ environments // ensure it is ready for release // ensure it meets users' needs

3 (a) (i) 1 mark per bullet to max 2: [2]

- `11011111`
- `AND`

(ii) 1 mark per bullet to max 2: [2]

- `00100000`
- `OR`

(b) 1 mark per line

START:	LDR	#0	// initialise index register to zero	1
	LDX	WORD	// get first character of WORD	1
	AND	MASK1	// ensure it is in upper case using MASK1	1
	OUT		// output character to screen	
	INC	IX	// increment index register	1
	LDM	#1	// load 1 into ACC	1
	STO	COUNT	// store in COUNT	1
LOOP:	LDX	WORD	// load next character from indexed address WORD	1
	OR	MASK2	// make lower case using MASK2	1
	OUT		// output character to screen	
	LDD	COUNT	// increment COUNT	1
	INC	ACC	//	
	STO	COUNT	//	
	CMP	LENGTH	// is COUNT = LENGTH?	1
	JPN	LOOP	// if FALSE – jump to LOOP	1
	END		// end of program	1
COUNT:	0			
MASK1:	B11011111		// bit pattern for upper case	1
MASK2:	B00100000		// bit pattern for lower case	
LENGTH:	4			
WORD:	B01100110		//ASCII code in binary for 'f'	
	B01101000		//ASCII code in binary for 'r'	
	B01000101		//ASCII code in binary for 'E'	
	B01000100		//ASCII code in binary for 'D'	

[max 12]

4 (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.
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) 1 mark for identifying the correct line, 1 mark for writing the corrected line

A - Line 5	**B** - Line 6	**C** - Line 5	[1]
`for i in range(Max-1):`	`FOR i := 1 TO (Max-1) DO`	`For i = 0 To (Max - 1)`	[1]

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

(ii) 1 mark for naming error, 1 mark for line number and correction

A Logic error	**B** Logic error	**C** Logic error	[1]
`7` `NoMoreSwaps = False`	`10` `NoMoreSwaps := FALSE;`	`7` `NoMoreSwaps = False`	[1]

(iii) 1 mark for naming, 1 for 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