UNIVERSITY OF CAMBRIDGE INTERNATIONAL EXAMINATIONS

GCE Advanced Subsidiary Level and GCE Advanced Level

MARK SCHEME for the May/June 2011 question paper

for the guidance of teachers

9691 COMPUTING

9691/23 Paper 2 (Written Paper), maximum raw mark 75

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 must be read in conjunction with the question papers and the report on the examination.

- Cambridge will not enter into discussions or correspondence in connection with these mark schemes.

Cambridge is publishing the mark schemes for the May/June 2011 question papers for most IGCSE, GCE Advanced Level and Advanced Subsidiary Level syllabuses and some Ordinary Level syllabuses.

1 **(a)**
- sensible request
- space to enter password
- space for attempt counter
- suitably labelled
- login message space
- title bar label
- return button
- use of all screen / well laid out / logical sequence [5]

(b)

Attempt	Password	Password ="poppy"	Attempt =3	Password ="poppy" OR Attempt=3	Output
1					
	poppy				
2					
		True			
			False		
				True	
					password correct

[1]

(c)

Attempt	Password	Password ="poppy"	Attempt =3	Password ="poppy" OR Attempt=3	Output
1					
	cat				
2					
		False			
			False		
				False	
	poppy				
3					
		True			
			True		
				True	
					password correct

1 mark for correct value at first condition
1 mark for correct value at 2nd condition
1 mark for correct value at 3rd condition
1 mark for correct logic for poppy, true, true, true
1 mark for correct output
1 mark for correct number of tries [6]

(d) (i) Attempt ← 0

(ii) Logic error

(e) (i)
- more characters
- at least two character types
- meaningless / hard to guess [2]

(ii) Any suitable obeying above rules [1]

(f) e.g. Pascal

```
CASE Attempt OF
   1: Writeln('First try is wrong. Please try again');
   2: Writeln('Password is still wrong. One more chance');
   3: Writeln('No valid password entered');
END;
```

e.g. VB 2005

```
SELECT CASE Attempt
   CASE 1
      Console.WriteLine("First try is wrong. Please try again")
   CASE 2
      Console.WriteLine ("Password is still wrong. One more
                          chance")
   CASE 3
      Console.WriteLine ("No valid password entered")
END SELECT
```

e.g. C#

```
switch (Attempt)
{
    case 1:
       Console.WriteLine("First try is wrong. Please try again")
       break;
    case 2:
       Console.WriteLine ("Password is still wrong. One more
                           chance")
       break;
    case 3:
       Console.WriteLine ("No valid password entered")
       Break;
}
```

1 mark for correct initial CASE statement
1 mark for correct first condition
1 mark for correct second condition
1 mark for correct end of of case statement(s) [4]

2 (a) (i) Any appropriate, such as "" or "x"

(ii) e.g. Pascal

```
VAR Track: ARRAY [1..150] OF STRING;
FOR i:= 1 TO 150
   DO
      Track[i] := 'xxx';
```

e.g. VB 2005

```
DIM Track(150) AS STRING;
FOR i = 1 TO 150
   Track(i) = "xxx";
NEXT
```

Alternative:

```
DIM Track(150) AS STRING;
FOR EACH i IN Track
   Track(i) = "xxx";
NEXT
```

e.g. C#

```
string[] track= new string[150];
for (int i = 1; i <= 150, i++)
{
   Track[i] = "xxx";
}
```

Alternative:

```
string[] track= new string[150];
foreach (int i in track)
{
   Track[i] = "xxx";
}
```

1 mark for sensible array name
1 mark for correct declaration range
1 mark for correct data type
1 mark for loop to address full range of array
1 mark for correct assignment [4]

(b) e.g. Pascal

```
i := 0;
Write('Which track do you want? ');
Readln(RequiredTrack);
REPEAT
   i := i + 1;
UNTIL Track[i] = RequiredTrack;
Writeln('The track is at position: ', i);
```

e.g. VB 2005

```
i = 0
Console.Write("Which track do you want? ")
Console.ReadLine(RequiredTrack)
DO
   i = i + 1
LOOP UNTIL (Track(i) = RequiredTrack)
Console.WriteLine('Track position is: ', i)
```

e.g. C#

```
i = 0;
Console.Write("Which track do you want? ");
requiredTrack = Console.ReadLine();
do
{
   i = i + 1;
}
while Track[i] != RequiredTrack;
Console.WriteLine('"Track position is: ", i);
```

1 mark for correct initialisation of index & incrementing
1 mark for sensible variable name for required track
1 mark for correct loop (REPEAT or WHILE loop acceptable)
1 mark for identifying search item
1 mark for output position [5]

(c)

Field Name	Data Type	Size of Field (bytes)
TrackID	Integer	4
TrackName	String / alphanumeric / text	20– 30
DateBought	Date / integer	8
Cost	Currency / integer / real / decimal / float	8
SoloArtist	Boolean	1

1 mark per box
NOT variant (as a data type) [10]

3 (a) – correct names in order: HOURS, TOTAL, TAX [3]

(b) – PRINT has two boxes under:
CASH and BANK [2]

(c) – indenting / white space
  – so it is easy to see blocks / to see structure of whole code
– meaningful names/identifiers
  – to help relate variables to problem/to help understand code
– annotation
  – to tell what a statement does without knowing the language
– good formatting (lower case, upper case) / reserved words are capitalised / in capitals
  – to highlight key words [4]

Any 2 x 2

(d) possible tests
- 5 values, all between 1 and 9, total <40
- 5 values, total >40
- 5 values, total close to 40
- 5 values, total =40
- 5 values, some values –ve
- 5 values, some values>9
- 5 values, all zero
- 5 values, total <0
- 5 values, total just over 0

Any 5 + reason. Reason must be correct for test values it relates to [10]

(e) (i)
- each variable has local scope // scope within block only
- does not affect same variable name in a different block [2]

(ii)
- different programmers can use the same name
- without affecting other uses of that name
- don't need to plan all variable names through whole program [3]

4 (a) (i) 1 [1]

(ii) 6 [1]

(b) (i)
- cannot end
- infinite loop
- produces error message (heap / stack overflow) [2]

(ii) Second line needs to be changed to …
- … if n<=1 (or comparable) [2]

(c)
```
FUNCTION calc(n)
   x ← 1
   FOR i ← 1 TO n
      x ← x * i
   NEXT i
   calc ← x
ENDFUNCTION     // RETURN
```

1 mark for initialisation
1 mark for correct loop from 1 to n
1 mark for multiplying current value by n
1 mark for assigning return value [4]