UNIVERSITY OF CAMBRIDGE INTERNATIONAL EXAMINATIONS

GCE Advanced Subsidiary Level and GCE Advanced Level

MARK SCHEME for the May/June 2012 question paper

for the guidance of teachers

9691 COMPUTING

9691/22 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 2012 question papers for most IGCSE, GCE Advanced Level and Advanced Subsidiary Level syllabuses and some Ordinary Level syllabuses.

1 (a)
–labelled box for name
–calendar for date of birth//drop-down lists for day,month,year//formatted boxes o
indication of how to write the date
–drop-down list for type of book//radio buttons (Accept tick boxes)
–yes/no radio buttons or (drop-down) list
–button to move from screen [Max 4]

(b)
–easy to use
–clear instructions
–appropriate for the purpose
–easy to understand
–to reduce errors

Reject consistent
Description alone is not enough [Max 3]

(c)

Field Name	Data Type	Field Size (bytes)
FirstName	String/alphanumeric/text	8–20
DateOfBirth	Date/string/integer	4, 6, 8, 10
BookType	String/alphanumeric/text	10
ReadsNovels	Boolean/char	1

[8]

(d) e.g.

```
PrintedTotal ← 0
AudioBookTotal ← 0
EBookTotal ← 0

REPEAT
    READ next record
    IF BookType = "printed"
        THEN PrintedTotal ← PrintedTotal + 1
        ELSE
            IF BookType = "audio-book"
                THEN AudioBookTotal ← AudioBookTotal + 1
                ELSE
                    IF BookType = "e-book"
                        THEN EBookTotal ← EBookTotal + 1
                    ENDIF
            ENDIF
    ENDIF
UNTIL no more student records
```

Marking guidelines:
–initialising each total before REPEAT
–nested IFs
–3 incrementations
–correct ENDIFs
–sensible identifiers and indenting [5]

(e) *Marking guidelines:*
–title
–3 totals boxes/lines
–3 percentage boxes/lines
–labels for all [3]

(f) (*File handling statement – 1 mark; explanation – 1 mark*) × 3

e.g. Pascal

```
AssignFile(Channel, ExternalFileName); -gives the FileNam
Channel ID through which access can be made
Reset(Channel); -opens existing file
Write(Channel, Record); -writes record to file
Read (Channel, Record); -reads record from file
Seek (Channel, RecordAddress); -goes directly to record at
specified address
CloseFile (Channel); -closes file
```

e.g. VB 2005

```
Channel = New FileStream(ExternalFileName,FileMode.Open)
FileReader = New BinaryReader(Channel)
NewFile = New FileStream(ExternalFileName, FileMode.Create
FileWriter = New BinaryWriter (NewFile)
Record.Field = FileReader.ReadString()
Record.Field = FileReader.ReadDecimal()
Record.Field = FileReader.ReadInt32()
FileWriter.Write(Field)
Channel.Close()
FileReader.Close()
FileWriter.Close()
NewFile.Close()
```

e.g. C#

```
channel = new FileStream(externalFileName,FileMode.Open)
fileReader = new BinaryReader(channel)
newFile = new FileStream(externalFileName, fileMode.Create
fileWriter = new BinaryWriter (newFile)
record.Field = FileReader.ReadString()
record.Field = FileReader.ReadDecimal()
record.Field = FileReader.ReadInt32()
fileWriter.Write(field)
channel.Close()
fileReader.Close()
fileWriter.Close()
newFile.Close()
```

[6]

2 (a)

ArraySize	Element	Element<ArraySize	Number		
			[1]	[2]	[3]
3					
	1				
		true			
			24		
	2				
		true			
				57	
	3				
		false			

1 mark for Element values 2, 3
1 mark for correct true
1 mark for correct false
1 mark for Number[1] set to 24
1 mark for Number[2] set to 57 [5]

(b) (i) –Logic/logical [1]

(ii) –WHILE Element <= ArraySize DO (or equivalent) [1]

(c)

```
Element ← 1
REPEAT
   INPUT Number[Element]
   Element ← Element + 1
UNTIL Element > ArraySize
```

Marking guidelines:
–correct initialisation of Element
–correct condition to end REPEAT loop [2]

(d) –check starting condition
–check state at iteration 499
–check state at iteration 500
–check state at iteration 501 [Max 3]

3 (a) Method of marking:
–inputting 2 strings
*–identifying * in each*
–identifying last part of first word
–adding second part of second word
–meaningful variable names
–output result
–indented code
–correct use of specified language

e.g. Pascal

```
ReadLn(String1);
ReadLn(String2);
i := 0;
REPEAT
   i := i + 1
UNTIL String1[i] = '*'; {or use i = Pos(String1,'*')}
String1 := RightString(String1, Length(String1)-i);
{or use Delete(String1,1,i)}
i := 0;
REPEAT
   i := i + 1
UNTIL String2[i] = '*';
String2 := RightString(String2, Length(String2)-i);
NewString := Concat(String1, String2);
WriteLn(NewString);
```

e.g. VB 2005

```
String1 = Console.ReadLine()
String2 = Console.ReadLine()
i = 0
DO
   i = i + 1
LOOP UNTIL (String1(i) = "*")
String1 = String1.SubString(i+1,String1.Length-i)
i = 0
DO
   i = i + 1
LOOP UNTIL (String2(i) = "*")
String2 = String2.SubString(i+1,String2.Length-i)
NewString = String.Concat(String1,String2)
Console.WriteLine(NewString)
```

e.g. C#

```
string1 = Console.ReadLine();
string2 = Console.ReadLine();
i = 1;
while (string1[i] != "*")
   {
   i = i+1;
   }
string1 = string1.SubString(i+1, string1.Length-i)
[Note: could also write string1 = string1.Remove(1,i)]
i = 1;
while (string2[i] != "*")
   {
```

```
    i = i+1;
    }
string2 = string2.SubString(i+1, string2.Length-i)
newString = String.Concat(string1,string2)
```
[Note: can also write newString = string1 + string2]
Console.WriteLine(newString) [max 8]

(b) (i) –String1, String2 (their input string names) [2]

(ii) e.g. Pascal
```
Function JoinStrings (String1, String2): String
```
e.g. VB 2005
```
Function JoinStrings(ByVal String1, String2 As String) As String
```
e.g. C#
```
static string joinStrings(string string1, string2)
```

Mark points
–function check type as appropriate
–parameters in brackets [2]

(iii) –single output makes this appropriate [1]

4 (a) (i) 3.33333 (or equivalent)

(ii) 1

(iii) 3 [3]

(b) (i) Y DIV X [1]

(ii) Y MOD X [1]

5 (a) *There are many different ways to represent the working*
Example:
–Happening (4) becomes Happening (3) + 4
–Happening (3) becomes Happening (2) + 3
–Happening (2) becomes Happening (1) + 2
–Ends at 1
–Diagram works back through function calls
–Happening (4) = 10 [6]

(b) (i) –4
–6
–function name takes a value [3]

(ii) –6
–function defined in terms of itself [2]

(c) –infinite loop//runs out of stack space [1]

(d) –FOR loop usually simpler to understand
–usually simpler to write
–iteration less chance of error
–large number of function calls could cause stack overflow
–... this is not a problem for small values of Num
–recursion could be quicker
–recursive solution is a more elegant solution [4]