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

9691/22 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 2011 question papers for most IGCSE, GCE Advanced Level and Advanced Subsidiary Level syllabuses and some Ordinary Level syllabuses.

1 (a)

Field Name	Data Type	Size of Field (bytes)
JobID	Integer	4
JobDescription	String / alphanumeric / text	20–50
Price	Currency / integer / real / decimal / float	8
ExpectedCompletionDate	Date / integer	8
Paid	Boolean	1

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

(b)
- Result (e.g. 4+29+8+8+1=50 – size of 1 record)
- Multiplied by 200 (e.g. 10,000)
- Add (10%) (e.g. 11,000)
- Divided by 1024 (e.g. 11,000 ÷ 1024)
- Result between 6.2 and 59.7KB (e.g. 10.7KB) [5]

(c) e.g. Pascal

```
TYPE JobRecord = RECORD
   JobID: Integer;
   JobDescription: String;
   Price: Currency;
   ExpectedCompletionDate: TDateTime;
   Paid: Boolean
END;
```

e.g. VB6

```
Type JobRecord
   DIM JobID AS Integer
   DIM JobDescription AS String
   DIM Price AS Decimal
   DIM ExpectedCompletionDate AS Date
   DIM Paid AS Boolean
END Type
```

e.g. VB 2005

```
STRUCTURE JobRecord
   DIM JobID AS Integer
   DIM JobDescription AS String
   DIM Price AS Decimal
   DIM ExpectedCompletionDate AS Date
   DIM Paid AS Boolean
END STRUCTURE
```

e.g. C#

```
struct jobRecord
{
   public int jobID;
   public string jobDescription;
   public decimal price;
   public datetime expectedCompletionDate;
   public bool paid;
}
```

1 mark for heading
1 mark for structure
1 mark for all 5 fields correct [3]

(d) **(i)** – to check that data is reasonable / acceptable / follows rules
– to check data is complete [1]

NOT correctness

(ii) – range check explanation
– length check explanation
– format check explanation

Max 2 marks [2]
NOT presence check

(e) (JobID > 0) AND (JobID <= 1000)

Alternative answers:
(JobID > 0) AND (JobID < 1001)
(JobID >= 1) AND (JobID <= 1000)
(JobID >= 1) AND (JobID < 1001)

Correct brackets 1 mark; correct operator 1 mark

(Paid=True) OR (Paid=False)
Accept (Paid=yes) OR (Paid=no) *(ignore speech marks)*
Accept (Paid=1) OR (Paid=0)

Correct brackets 1 mark; correct operator 1 mark [4]

(f) Any sensible + reason accepted
e.g. 500 – valid data – within acceptable range / normal
1 – valid data – lower boundary included / extreme
1000 – valid data – upper boundary included / extreme
– 1 – invalid data – below boundary
1001 – invalid data – above boundary

1 mark per data item, 1 mark per matching reason [8]

2 (a) (i)

Word	Count	Index	Word(Index)	Word(Index)= 'a'
banana				
	0			
		1		
			b	
				false
		2		
			a	
				true
	1			
		3		
			n	
				false
		4		
			a	
				true
	2			
		5		
			n	
				false
		6		
			a	
				true
	3			

1 mark for each correct column (except Word column)
1 mark for correct sequence
1 mark for readable presentation [6]

(ii)

Word	Count	Index	Word(Index)	Word(Index)= 'a'
Ant				
	0			
		1		
			A	
				false
		2		
			n	
				false
		3		
			t	
				false

1 mark for correct Count column
1 mark for correct Word(Index)='a' column (need false only once after A)
1 mark for Index column and Word(Index) column correct [3]

(b) IF (Word(Index) = 'a') OR (Word(Index) = 'A')
1 mark for OR (allow lower case or)
1 mark for separate decisions correct
// 2 marks for If Uppercase(Word(Index))='A'
// 2 marks for If Lowercase(Word(Index))='a'
must reflect existing pseudocode style [2]

(c) (i)
– meaningful variable names
– indentation / white space
– structured English
– good formatting (lower case, upper case)
– reserved words are capitalised / in capitals [2]

(ii) Annotation / comments [1]

(iii)
– to make it easier to find / correct errors
– to make it easier to modify the program / maintenance [2]

(d) (i)
– numeric/binary (code where each character has a unique value) [1]

(ii)
– letter a-z have increasing ASCII codes
– Each character's ASCII value is compared
– the character with the smaller value is the first character / the character with the larger value is the second character / (letters are sorted) [3]

(iii)
– characters are compared in turn …
– from left hand side / start of each word
– … until two characters are different
– the lower code value determines the first word
– if 2 words are the same when one ends …
– … this is the first word [4]

3 **(a)** 0 (zero)

(b) e.g. Pascal

```
VAR Letter: ARRAY [1..26] OF Integer;
FOR I := 1 TO 26
   DO
      Letter[i] := 0;
```

Alternative:

```
VAR Letter: ARRAY ['a'..'z'] OF Integer;
FOR l := 'a' TO 'z'
   DO
      Letter[l] := 0;
```

e.g. VB 2005

```
DIM Letter(26) AS Integer
FOR i = 1 TO 26
   Letter(i) = 0
NEXT
```

e.g. C#

```
string[] letter = new string[26]
for (int i = 1; i <= 26; i++)
{
   letter[i] = 0
}
```

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]

(c) e.g. Pascal

```
ThisLetterIndex :=
                ASCII(ThisLetter)-ASCII('a') + 1;
Letter[ThisLetterIndex] :=
                    Letter[ThisLetterIndex] + 1;
```

Alternative: (if character range used for array index)

```
Letter[ThisLetter] := Letter[ThisLetter] + 1;
```

e.g. VB 2005

```
ThisLetterIndex = ASC(ThisLetter)-ASC("a") + 1
Letter(ThisLetterIndex) =
                    Letter(ThisLetterIndex) + 1
```

e.g. C#

```
thisLetterIndex = asc(thisLetter) - asc('a') + 1;
letter[thisLetterIndex] =
                    letter[thisLetterIndex] + 1;
```

1 mark for finding correct array element
1 mark for incrementing running total correctly
1 mark for correct overall logic [1]

4 (a) (i) 1 [1]

(ii) 6 [1]

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

(ii)
- Before second line extra code needs to be added
- … if n<1 (OR if n<0)
- then error (or equivalent) [2]

(c)

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

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