Cambridge International AS & A Level

Cambridge International Examinations
Cambridge International Advanced Subsidiary and Advanced Level

COMPUTER SCIENCE **9608/43**

Paper 4 Further Problem-solving and Programming Skills **May/June 2017**

PRE-RELEASE MATERIAL

8302861001

No Additional Materials are required.

This material should be given to the relevant teachers and candidates as soon as it has been received at the Centre.

READ THESE INSTRUCTIONS FIRST

Candidates should use this material in preparation for the examination. Candidates should attempt the practical programming tasks using their chosen high-level, procedural programming language.

DC (ST/SW) 144517/1

Teachers and candidates should read this material prior to the June 2017 examination for 9608 Paper 4.

Reminders

The syllabus states:

- there will be questions on the examination paper which do not relate to this pre-release material
- you must choose a high-level programming language from this list:
    - Visual Basic (console mode)
    - Python
    - Pascal / Delphi (console mode)

Note: A mark of **zero** will be awarded if a programming language other than those listed is used.

The practical skills for Paper 4 build on the practical skills covered in Paper 2. We therefore recommend that candidates choose the same high-level programming language for this paper as they did for Paper 2. This will give candidates the opportunity for extensive practice and allow them to acquire sufficient expertise.

Questions on the examination paper may ask the candidate to write:

- structured English
- pseudocode
- program code

A program flowchart should be considered as an alternative to pseudocode for the documenting of an algorithm design.

Candidates should be confident with:

- the presentation of an algorithm using either a program flowchart or pseudocode
- the production of a program flowchart from given pseudocode and vice versa.

Candidates will also benefit from using pre-release materials from previous examinations. These are available on the teacher support site.

Declaration of variables

The syllabus document shows the syntax expected for a declaration statement in pseudocode.

```
DECLARE <identifier> : <data type>
```

If Python is the chosen language, each variable's identifier (name) and its intended data type must be documented using a comment statement.

Structured English – Variables

An algorithm in pseudocode uses variables, which should be declared. An algorithm in structured English does not always use variables. In this case, the candidate needs to use the information given in the question to complete an identifier table. The table needs to contain an identifier, data type and description for each variable.

TASK 1

Students at a college are given several tests during their course. A teacher wants to write object-oriented software to process data about the tests.
For each test, the following are to be stored:

- one or more questions, up to a maximum of 10 questions
- the maximum number of marks for the test
- the level (A, S, G)

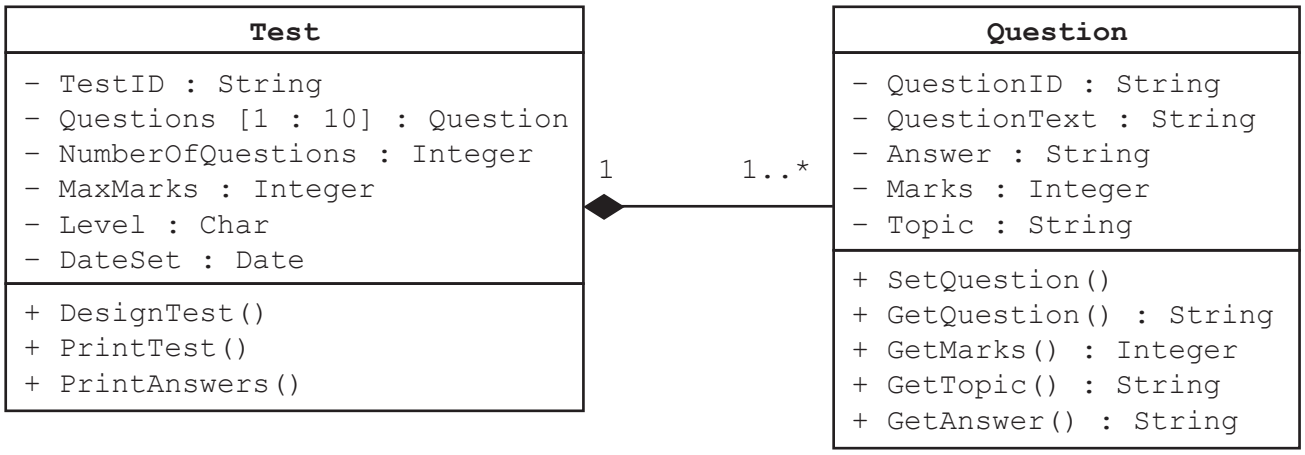
For each question, the following are to be stored:

- the question text
- the answer
- the maximum number of marks
- the topic

Key focus: Object-oriented programming

TASK 1.1

The relationship between `Test` and `Question` is shown in the following containment (aggregation) class diagram.



Explain what containment means in the context of OOP.

Investigate what other information this diagram conveys.

TASK 1.2

Write object-oriented program code to implement the classes.

Remember to use validation and error trapping where appropriate.

TASK 2

The table shows part of the instruction set for a processor which has one general purpose register, the Accumulator (ACC), and an Index Register (IX).

Note: these instructions are referred to in the syllabus sections 1.4.3 and 3.6.2.

Instruction		Explanation
Op code	**Operand**	
LDM	#n	Immediate addressing. Load the number n to ACC.
LDD	<address>	Direct addressing. Load the contents of the location at the given address to ACC.
LDI	<address>	Indirect addressing. The address to be used is at the given address. Load the contents of this second address to ACC.
LDX	<address>	Indexed addressing. Form the address from <address> + the contents of the index register. Copy the contents of this calculated address to ACC.
LDR	#n	Immediate addressing. Load the number n into IX.
STO	<address>	Store the contents of ACC at the given address.
ADD	<address>	Add the contents of the given address to the ACC.
INC	<register>	Add 1 to the contents of the register (ACC or IX).
DEC	<register>	Subtract 1 from the contents of the register (ACC or IX).
JMP	<address>	Jump to the given address.
CMP	<address>	Compare the contents of ACC with the contents of <address>.
CMP	#n	Compare the contents of ACC with number n.
JPE	<address>	Following a compare instruction, jump to <address> if the compare was TRUE.
JPN	<address>	Following a compare instruction, jump to <address> if the compare was FALSE.
AND	#n	Bitwise AND operation of the contents of ACC with the operand.
AND	<address>	Bitwise AND operation of the contents of ACC with the contents of <address>.
XOR	#n	Bitwise XOR operation of the contents of ACC with the operand.
XOR	<address>	Bitwise XOR operation of the contents of ACC with the contents of <address>.
OR	#n	Bitwise OR operation of the contents of ACC with the operand.
OR	<address>	Bitwise OR operation of the contents of ACC with the contents of <address>.
IN		Key in a character and store its ASCII value in ACC.
OUT		Output to the screen the character whose ASCII value is stored in ACC.
END		Return control to the operating system.

Notes:

# denotes immediate addressing
B denotes a binary number, for example, `B01001010`
& denotes a hexadecimal number, for example, `&4A`

Tasks 2.1 to 2.7 all use one of the following two formats for symbolic addressing.

Format			Example
`<label>:`	`<op code>`	`<operand>`	`START: LDM #0`
`<label>:`	`<data>`		`NUM1: B01001010`

Key focus: Low-level programming

Tasks 2.1 to 2.5 show high-level language constructs written in pseudocode. Each task consists of writing the assembly language equivalent of the given high-level language construct.

Write assembly language program code using the given instruction set.

TASK 2.1

```
X ← A + B
END
```

Label	Instruction		Comment
	Op code	Operand	
START:			// load the content of A into ACC
			// add the content of B to content of ACC
			// store content of ACC at address X
	END		// end of program
X:			
A:	5		
B:	3		

TASK 2.2

```
IF X = A
   THEN
      OUTPUT CHR(X)  // statements for THEN part
   ELSE
      A ← A + 1     // statements for ELSE part
ENDIF
END
```

Label	Instruction		Comment
	Op code	Operand	
START:			// load the content of X into ACC
			// compare the content of ACC with content of A
			// if not equal (FALSE), jump to ELSE
THEN:			// instruction for the THEN part goes here
	JMP	ENDIF	// jump over the ELSE part
ELSE:			// instructions for ELSE part start here
ENDIF:	END		// end of program
A:	65		
X:	67		

Note: the built-in function `CHR(X)` returns the character that is represented by the ASCII code held in `X`.

TASK 2.3

```
REPEAT
   OUTPUT CHR(X)
   X ← X - 1
UNTIL X = A
END
```

Label	Instruction		Comment
	Op code	Operand	
LOOP:			// instructions to be repeated start here
			// is content of ACC = content of A ?
			// if not equal (FALSE), jump to LOOP
	END		// end of program
X:	90		
A:	65		

TASK 2.4

```
FOR COUNT ← 1 TO 4
   OUTPUT CHARS[COUNT]
ENDFOR
END
```

Label	Instruction		Comment
	Op code	Operand	
			// set ACC to 1
			// store contents of ACC in COUNT
			// set IX to 0
LOOP:			// COUNT = 4 + 1 ? starts here
			// if equal (TRUE), jump to ENDFOR
			// instructions to be repeated start here
			// increment IX
			// increment COUNT starts here
			// jump to LOOP
ENDFOR:	END		// end of program
COUNT:			
CHARS:	72		// 'H'
	69		// 'E'
	76		// 'L'
	80		// 'P'

[Turn over

TASK 2.5

```
WHILE X <> B
   OUTPUT CHARS[B]
   B ← B + 1
ENDWHILE
END
```

Label	Instruction		Comment
	Op code	Operand	
LOOP:			// load contents of X into ACC
			// is contents of ACC = contents of B ?
			// if equal (TRUE), jump to ENDWHILE
			// instructions to be repeated start here
			// set IX to B starts here
			// set ACC to 1
			// store content of ACC in FORCOUNT
			// FORCOUNT = B + 1 ? starts here
			// decrement ACC to FORCOUNT - 1
			// FORCOUNT – 1 = B ?
			// if equal (TRUE), jump to ENDFOR
			// increment IX
			// output CHARS[B] starts here
			// increment B starts here
			// jump back to start of while loop
	END		// end of program
FORCOUNT:			// control variable for inner loop
X:	4		
B:	0		
CHARS:	72		// 'H'
	69		// 'E'
	76		// 'L'
	80		// 'P'

TASK 2.6

Output a string using indirect addressing.

Address	Instruction		Comment
	Op code	Operand	
LOOP:			// use indirect addressing to load contents of address found at address 100
			// output character with ASCII code held in ACC
			// load content of address 100
			// increment ACC
			// store content of ACC at address 100
			// is content of ACC = 107 ?
			// if not equal (FALSE), jump to LOOP
	END		// end of program
~	~		~
100	102		
101			
102	77		// 'M'
103	65		// 'A'
104	84		// 'T'
105	72		// 'H'
106	83		// 'S'
107			

[Turn over

TASK 2.7

Programmers use bitwise operations (AND, OR, XOR) to set or examine specific bits.

Example:

Label	Instruction		Comment
	Op code	Operand	
START:	LDD	X	// load content of X into ACC
	AND	MASK	// bitwise AND operation on content of ACC and content of MASK
	STO	Y	// store content of ACC in Y
	END		// end of program
X:	B10101111		
Y:			// what does the value of Y tell you about X ?
MASK:	B00000001		

Write simple programs using the different bitwise operations (AND, OR, XOR) and different MASK content.

Identify the operation and MASK bit pattern to:

- set a bit to 1, leaving all other bits unchanged
- set a bit to 0, leaving all other bits unchanged
- test whether a specific bit is 1
- test whether a specific bit is 0.