

Cambridge International Examinations Cambridge International Advanced Level

COMPUTER SCIENCE

9608/42 October/November 2016

Paper 4 Further Problem-solving and Programming Skills

PRE-RELEASE MATERIAL

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.



This document consists of 12 printed pages.



Teachers and candidates should read this material prior to the November 2016 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 for 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 (or the reverse)

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

When writing high-level language programs you are asked to use one of:

- Python
- Visual Basic (console mode)
- Pascal/Delphi (console mode)

Key focus: Programming environments

Syllabus section 2.4.1 refers to an Integrated Development Environment (IDE). Syllabus section 4.3.4 refers to the use of development tools and programming environments.

Make sure you know the name of the programming environment that you use to write programs in your chosen programming language.

TASK 1.1

Explore the features of your editor that help you to write program code. When and how does your programming environment report syntax errors?

TASK 1.2

When you write program code, explore debugger features available to you and practise using them to step through programs and explore the state of the variables after each instruction.

Explore how the debugger can help to find logic errors and run-time errors.

TASK 1.3

Write program code for the following insertion sort algorithm. Correct any syntax errors. Then use the debugger to find other types of error.

You will need to write a main program that initialises and populates an array of names to be sorted. The program then calls the procedure Sort.

```
PROCEDURE Sort(BYREF Names : ARRAY OF STRING, BYVAL Start : INTEGER,
BYVAL Finish : INTEGER)
FOR ThisPointer ← Start + 1 TO Finish
ThisName ← Names[ThisPointer]
Pointer ← ThisPointer - 1
WHILE Names[Pointer] > ThisValue OR Pointer > 0
Names[Pointer + 1] ← Names[Pointer]
Pointer ← Pointer - 1
ENDWHILE
Names[Pointer + 1] ← ThisName
ENDFOR
ENDPROCEDURE
```

Decide on some test data. Write down the expected results after each repetition (iteration) of the FOR loop. You should test your program with several different sets of data. Use the debugger each time to step through the program.

Extension task

Explore what program libraries are available for your chosen programming language. How can you make use of a library routine that is not part of the basic set-up of your programming environment?

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 all referred to in syllabus sections 1.4.3 and 3.6.2.

Instruction				
Op code	Operand	Explanation		
LDM	#n	Immediate addressing. Load the number n to ACC		
LDD	<address></address>	Direct addressing. Load the contents of the given address to ACC		
LDI	<address></address>	Indirect addressing. The address to be used is at the given address. Load the contents of this second address to ACC		
LDX	<address></address>	Indexed addressing. Form the address from <address> + the contents of the index register. Copy the contents of this calculated address to ACC</address>		
LDR	#n	Immediate addressing. Load the number n into IX		
STO	<address></address>	Store the contents of ACC at the given address		
ADD	<address></address>	Add the contents of the given address to the ACC		
INC	<register></register>	Add 1 to the contents of the register (ACC or IX)		
DEC	<register></register>	Subtract 1 from the contents of the register (ACC or IX)		
JMP	<address></address>	Jump to the given address		
CMP	<address></address>	Compare the contents of ACC with the contents of <address></address>		
CMP	#n	Compare the contents of ACC with number n		
JPE	<address></address>	Following a compare instruction, jump to <address> if the compare was TRUE</address>		
JPN	<address></address>	Following a compare instruction, jump to <address> if the compare was FALSE</address>		
AND	#n	Bitwise AND operation of the contents of ACC with the operand		
AND	<address></address>	Bitwise AND operation of the contents of ACC with the contents of <address></address>		
XOR	#n	Bitwise XOR operation of the contents of ACC with the operand		
XOR	<address></address>	Bitwise XOR operation of the contents of ACC with the contents of <address></address>		
OR	#n	Bitwise OR operation of the contents of ACC with the operand		
OR	<address></address>	Bitwise OR operation of the contents of ACC with the contents of <address></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 $\&4 {\tt A}$

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

	Example		
<label>:</label>	<op code=""></op>	<operand></operand>	START: LDA #0
<label>:</label>	<data></data>		NUM1: B01001010

Key focus: Low-level Programming

Write assembly language program code using the instruction set provided on page 4.

Tasks 2.1 to 2.5 show a high-level language construct written in pseudocode. Each of these tasks consists of writing the assembly language.

 $X \leftarrow A + B$ END

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

```
IF X = A

THEN

OUTPUT CHR(X) // statements for THEN part

ELSE

A \leftarrow A + 1 // statements for ELSE part

ENDIF
```

END

	Instruction		
Label	Op code	Operand	Comment
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 THEN part start here
	JMP	ENDIF	// jump over the ELSE part
ELSE:			// instructions for ELSE part start here
ENDIF:			// remainder of program continues from here
	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.

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

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

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

	Instruction			
Label	Op code	Operand	Comment	
			// set ACC to zero	
			// store content of ACC in COUNT	
LOOP:			// increment COUNT starts here	
			<pre>// instructions to be repeated start here</pre>	
			// COUNT = 4 ?	
			// if not equal (FALSE), jump to LOOP	
	END		// end of program	
COUNT:				
CHARS:	72		// 'H'	
	69 76		// 'E'	
			// 'L'	
	80		// 'P'	

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

Label	Instruction		
	Op code	Operand	Comment
LOOP:			// load content of X into ACC
			<pre>// is content of ACC = content of B ?</pre>
			// if equal (TRUE) jump to ENDWHILE
			<pre>// instructions to be repeated start here</pre>
			// jump back to start of loop
	END		// end of program
х:	4		
В:	0		
CHARS:	72		// 'H'
	69		// 'E'
	76		// 'L'
	80		// 'P'

Output a string using indirect addressing.

	Instruction		
Address	Op code	Operand	Comment
LOOP:			<pre>// use indirect addressing to load contents of</pre>
			// output character with ASCII code held in ACC
			// load content of address 100
			// increment ACC
			<pre>// store content of ACC at address 100</pre>
			<pre>// is content of ACC = 107 ?</pre>
			// 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			

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

Example:

Label	Instruction		
	Op code	Operand	Comment
LOOP:	LDD	Х	// load content of X into ACC
	AND	MASK	<pre>// bitwise AND operation on content of ACC and</pre>
	STO	Y	// store content of ACC in Y
	END		// end of program
х:	B10101111		
Y:			// what does the value of Y tell you about X ?
MASK:	B0000001		

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

Permission to reproduce items where third-party owned material protected by copyright is included has been sought and cleared where possible. Every reasonable effort has been made by the publisher (UCLES) to trace copyright holders, but if any items requiring clearance have unwittingly been included, the publisher will be pleased to make amends at the earliest possible opportunity.

To avoid the issue of disclosure of answer-related information to candidates, all copyright acknowledgements are reproduced online in the Cambridge International Examinations Copyright Acknowledgements Booklet. This is produced for each series of examinations and is freely available to download at www.cie.org.uk after the live examination series.

Cambridge International Examinations is part of the Cambridge Assessment Group. Cambridge Assessment is the brand name of University of Cambridge Local Examinations Syndicate (UCLES), which is itself a department of the University of Cambridge.