Cambridge Assessment International Education
Cambridge International Advanced Subsidiary and Advanced Level

COMPUTER SCIENCE **9608/41**

Paper 4 Further Problem-solving and Programming Skills **October/November 2019**

PRE-RELEASE MATERIAL

9588995533

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.

This document consists of **11** printed pages and **1** blank page.

DC (SC/CB) 168339/4

Cambridge Assessment International Education

[Turn over

Teachers and candidates should read this material prior to the November 2019 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:
    - 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 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 documenting a high-level 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 – Program Evaluation Review Technique (PERT) Charts

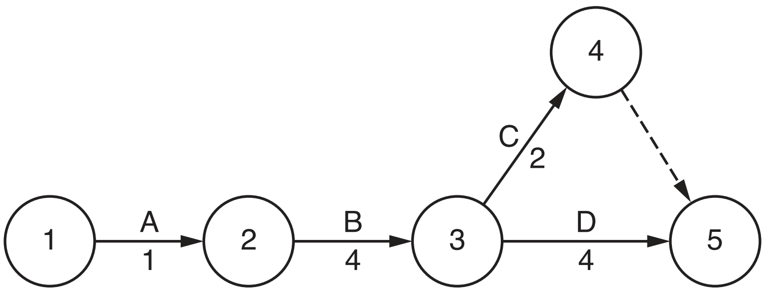
One of the techniques that can be used in project management is a PERT chart.

Each activity that needs to be completed is listed with the time it will take to complete the activity and the predecessor activity.

These data can be represented in a table as shown.

Activity	Description	Time to complete (weeks)	Predecessor
A	Identify requirements	1	-
B	Build module 1	4	A
C	Test module 1	2	B
D	Build module 2	4	B

A PERT chart can then be created to represent the table. For example:



The dashed line in the diagram represents a dummy activity.

TASK 1.1

Create a table to show the activities, times to complete and predecessors for a program that you are creating, or plan to create in class.

TASK 1.2

Create a PERT chart to represent the table that you have created.

TASK 1.3

Analyse the PERT chart that you have created.

Identify the critical path of the project.

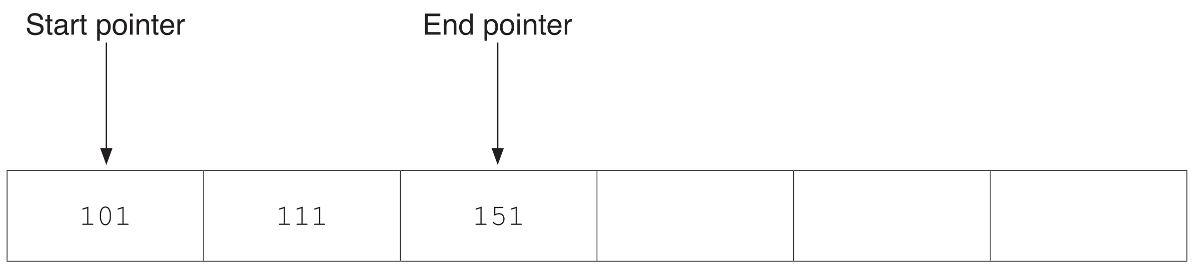
Calculate the minimum time in which the project can be completed.

TASK 2 – Queues

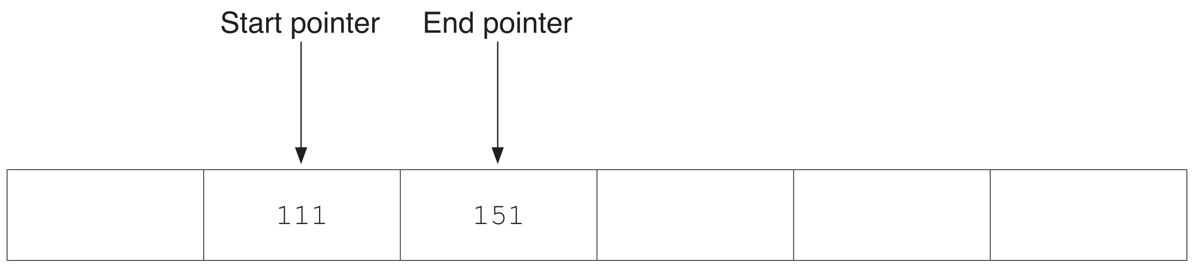
TASK 2.1

A queue is an abstract data type (ADT) that stores and manipulates a collection of data.

The values `101`, `111` and `151` are currently stored in a queue as follows:



A command has been carried out on the queue. This is the new state of the queue.



Explain what has happened to the queue.

TASK 2.2

Further commands are carried out as follows:

```
ADD 175

REMOVE

REMOVE

ADD 150
```

Draw a representation to show the queue after the commands have been carried out.

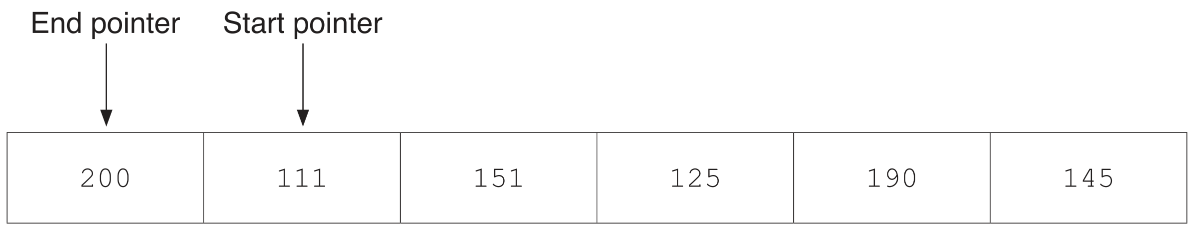
TASK 2.3

Queues can be linear or circular structures.

Explain the difference between a linear queue and a circular queue to another student.

TASK 2.4

The current status of a circular queue is as follows:



The following commands are carried out:

```
REMOVE

REMOVE

ADD 99

ADD 180
```

Draw a diagram to show the queue after the commands have been carried out.

TASK 2.5

Discuss what should happen if a program were to execute the following command on the queue that you have drawn in **TASK 2.4**.

```
ADD 120
```

TASK 2.6

Write **program code** in the language of your choice to implement the circular queue. Your code must:

- use a 1D array
- allow items of data to be added and removed from the queue
- not be able to add data to the queue if it is full
- not be able to remove data from the queue if it is empty.

TASK 3 – Assembly code

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		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 to 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>. <address> can be an absolute address or a symbolic address.
`LSL`	`#n`	Bits in ACC are shifted n places to the left. Zeros are introduced on the right hand end.
`LSR`	`#n`	Bits in ACC are shifted n places to the right. Zeros are introduced on the left hand end.
`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.

TASK 3.1

Write the assembly language program code that represents the following high-level language construct written in pseudocode.

```
X ← A + B
END
```

Label	Op code	Operand	Comment
			// load contents from A
			// add the contents from B
			// store the contents of ACC in X
	END		// end program
A:	2		
B:	10		
X:			

TASK 3.2

Write the assembly language program code that represents the following high-level language construct written in pseudocode.

```
X ← A + B + (C * 2)
END
```

Label	Op code	Operand	Comment
			// load
			// add
			// add
			// add
			// store
			// end program
A:	2		
B:	10		
C:	15		
X:			

TASK 3.3

The first pseudocode statement in **TASK 3.2** changes to:

```
X ← A + B + (C * 8)
```

It would be inefficient to add the value stored in `C` eight times.

Examine the instruction set on page 6. Discuss how you could create a more efficient solution.

TASK 3.4

Write the assembly language program code that represents the following high-level language construct written in pseudocode.

```
B ← B * 2
END
```

Label	Op code	Operand	Comment
			// load contents B
			// perform shift to multiply value by 2
			// store the contents of ACC in B
			// end program
B:	B00110011		

TASK 3.5

Write the assembly language program code that represents the following pseudocode algorithm.

```
COUNT ← 0
OUTPUT B
WHILE COUNT <> 5
   OUTPUT B
   COUNT ← COUNT + 1
ENDWHILE
OUTPUT A
END
```

Label	Op code	Operand	Comment
			// load 0
			// store in COUNT
			// load B
			// output B
LOOP:			// load COUNT
			// is COUNT = 5 ?
			// jump to ENDLOOP if TRUE
			// load B
			// output B
			// load COUNT
			// increment ACC
			// store in COUNT
			// jump to LOOP
ENDLOOP:			// load A
			// output A
			// end program
A:	1		
B:	0		
COUNT:			

TASK 3.6

Write the assembly language program code to find out whether or not a number is odd.

The number is a binary value and is stored in location `A`.

The program must only output the number if it is odd.

Label	Op code	Operand	Comment
			`// load A`
			`// perform bitwise AND operation with MASK`
			`// check if result is equal to MASK`
			`// if FALSE, jump to ENDP`
			`// load A`
			`// output A`
`ENDP:`			`// end program`
`A:`	`B01011001`		
`MASK:`	`B00000001`		

BLANK PAGE