Cambridge International AS & A Level

Cambridge International Examinations
Cambridge International Advanced Subsidiary and Advanced Level

CANDIDATE NAME

CENTRE NUMBER | CANDIDATE NUMBER

5529018953

COMPUTER SCIENCE **9608/42**

Paper 4 Further Problem-solving and Programming Skills **October/November 2017**

2 hours

Candidates answer on the Question Paper.

No Additional Materials are required.

No calculators allowed.

READ THESE INSTRUCTIONS FIRST

Write your Centre number, candidate number and name in the spaces at the top of this page.
Write in dark blue or black pen.
You may use an HB pencil for any diagrams, graphs or rough working.
Do not use staples, paper clips, glue or correction fluid.
DO **NOT** WRITE IN ANY BARCODES.

Answer **all** questions.
No marks will be awarded for using brand names of software packages or hardware.

At the end of the examination, fasten all your work securely together.
The number of marks is given in brackets [ ] at the end of each question or part question.

The maximum number of marks is 75.

This document consists of **14** printed pages and **2** blank pages.

DC (LK/JG) 129979/4

CAMBRIDGE
International Examinations

1 Students are choosing their A Level subjects based on their IGCSE subject results.

A student can take:

- Computer Science, if they have a grade C in Maths or a grade C in Computer Science
- Maths, if they have a grade C in Maths
- Physics, if they have a grade C in Science and a grade C in Maths.

(a) Complete the decision table.

		Column							
		1	2	3	4	5	6	7	8
Conditions	Grade C in Computer Science	Y	Y	Y	Y	N	N	N	N
	Grade C in Maths	Y	Y	N	N	Y	Y	N	N
	Grade C in Science	Y	N	Y	N	Y	N	Y	N
Actions	Take Computer Science								
	Take Maths								
	Take Physics								

[4]

(b) Simplify your solution by removing redundancies.

		Column							
		S	T	U	V	W	X	Y	Z
Conditions	Grade C in Computer Science								
	Grade C in Maths								
	Grade C in Science								
Actions	Take Computer Science								
	Take Maths								
	Take Physics								

[3]

(c) Show how the columns from **part (a)** were simplified to create the columns in **part (b)**.

For example, if columns 5, 6 and 7 were simplified to create column X, then you state this in your answer.

....

....

....

....

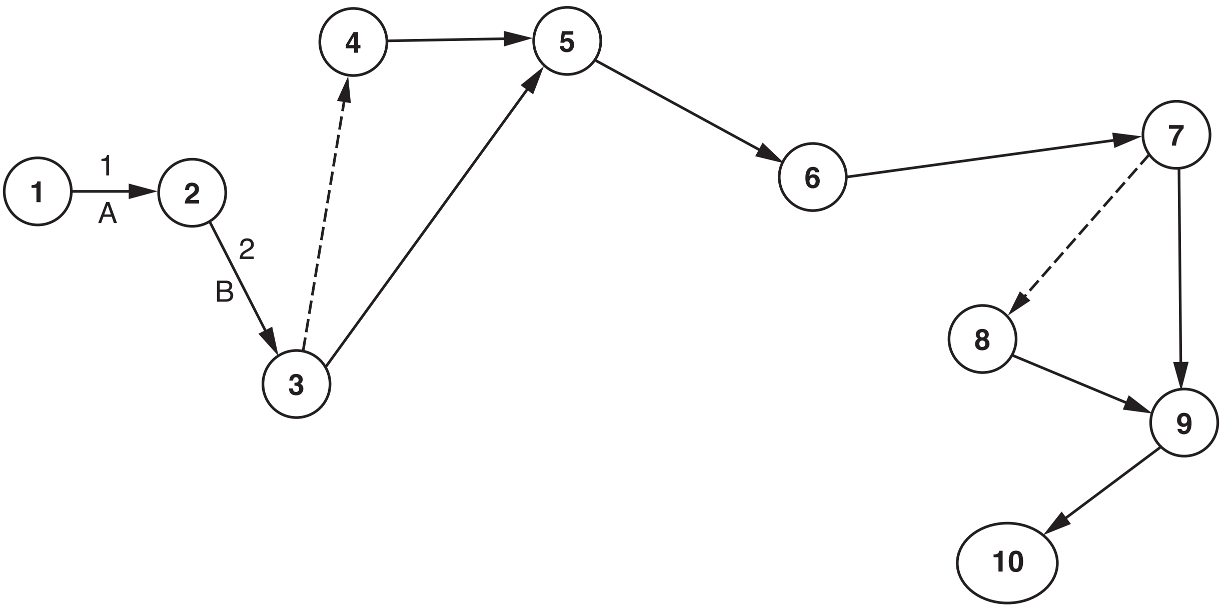
....

....[3]

2 (a) A project manager is planning to create a new computer game. The following table shows the activities and the estimated number of weeks to complete each activity.

Activity	Description	Weeks to complete
A	Interview end user	1
B	Produce requirements analysis	2
C	Design program structure	3
D	Design Interface	1
E	Program development	12
F	Black-box testing	2
G	Produce technical documentation	4
H	Acceptance testing	1
I	Installation	1

Complete the labelling of the Program Evaluation Review Technique (PERT) chart using the data in the table. The first two activities have been done for you.



[7]

(b) State what the dashed lines in the PERT chart represent.

....

....[1]

3 A declarative programming language is used to represent the knowledge base:

```
01      room(master_bedroom).
02      room(ensuite_bathroom).
03      room(office).
04      room(spare_bedroom).
05      room(nursery).
06      furniture(bed).
07      furniture(desk).
08      furniture(cot).
09      furniture(wardrobe).
10      furniture(computer).
11      located(bed, master_bedroom).
12      located(bed, spare_bedroom).
13      located(cot, nursery).
14      located(computer, office).
15      located(computer, master_bedroom).
```

These clauses have the following meanings:

Clause	Explanation
01	Master bedroom is a room
06	Bed is an item of furniture
11	Bed is located in the master bedroom

(a) Corridor is a room that contains a table and a lamp.

Write additional clauses to represent this information.

16 ....

17 ....

18 ....

19 ....

20 ....

[5]

(b) Using the variable `WhatItem`, the goal:

```
located(WhatItem, master_bedroom).
```

returns:

```
WhatItem = bed, computer
```

Write the result returned by the goal:

```
located(bed, WhichRoom).
```

`WhichRoom =` ....

....[2]

(c) (i) Clauses to identify rooms that are next to each other need to be stored.

The nursery is next to the master bedroom. This information is stored as:

```
21 nextTo(nursery, master_bedroom).
22 nextTo(master_bedroom, nursery).
```

Explain why both clauses are necessary.

....

....

....

....[2]

(ii) The corridor is next to the main bathroom.

Write additional clauses for this fact.

23 ....

24 ....

25 ....

[3]

(d) B can be moved into A, if B is furniture, A is a room and B is not already in A.

Write this as a rule.

`canBeMovedTo(`.................................... , ...`)`

`IF` ....

....[6]

4 **(a)** The array `Numbers[0 : Max]` stores numbers. An insertion sort can be used to sort these numbers into ascending order.

Complete the following **pseudocode** for the insertion sort algorithm.

```
FOR Pointer ← 1 TO (Max - 1)
   ItemToInsert ← ..........................................................................
   CurrentItem ← ..........................................................................
   WHILE (CurrentItem > 0) AND (Numbers[CurrentItem – 1] > ItemToInsert)
      Numbers[......................................................] ← Numbers[CurrentItem - 1]
      CurrentItem ← CurrentItem – 1
   ENDWHILE
   Numbers[CurrentItem] ← .......................................................................
ENDFOR
```

[4]

(b) Identify **two** features of the array `Numbers` that would have an impact on the performance of this insertion sort algorithm.

1 ....

2 ....

[2]

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

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.
`STX`	`<address>`	Indexed addressing. Form the address from <address> + the contents of the index register. Copy the contents from ACC to this calculated 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.
`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.

(a) Six letters are stored, starting at the location labelled `LETTERS`. A program is needed to perform a linear search on `LETTERS` to find the letter `'x'`. The program counts the number of times `'x'` appears in `LETTERS`.

The following is the pseudocode for the program.

```
FOR COUNT ← 0 TO 5
   IF LETTERS[COUNT] = LETTERTOFIND
      THEN
         FOUND ← FOUND + 1
   ENDIF
ENDFOR
```

Write this program. Use the op codes from the given instruction set.

Label	Op code	Operand	Comment
START:	LDR	#0	// initialise Index Register
LOOP:			// load LETTERS
			// is LETTERS = LETTERTOFIND ?
			// if not, go to NOTFOUND
			// increment FOUND
NOTFOUND:			// increment COUNT
			// is COUNT = 6 ?
			// if yes, end
			// increment Index Register
			// go back to beginning of loop
ENDP:	END		// end program
LETTERTOFIND:	'x'		
LETTERS:	'd'		
	'u'		
	'p'		
	'l'		
	'e'		
	'x'		
COUNT:	0		
FOUND:	0		

[10]

(b) Six values are stored, starting at the location `VALUES`. A program is needed to divide each of the values by 8 and store them back in their original location.

Write this program. Use the op codes from the instruction set on the next page.

Label	Op code	Operand	Comment
START:			// initialise the Index Register
			// load the value from VALUES
			// divide by 8
			// store the new value in VALUES
			// increment the Index Register
			// increment REPS
			// is REPS = 6 ?
			// repeat for next value
	END		
REPS:	0		
VALUES:	22		
	13		
	5		
	46		
	12		
	33		

[10]

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.
STX	<address>	Indexed addressing. Form the address from <address> + the contents of the index register. Copy the contents from ACC to this calculated 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.
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.

6 A bank has a range of customer accounts, which includes current accounts and savings accounts.

All accounts have:

- an account number
- a balance (amount of money in an account).

A current account has a level (bronze, silver or gold). A monthly fee ($) is taken from each account.

Savings account customers pay a regular amount ($) into their account. The payment interval is a number of weeks (for example, 4).

An object-oriented program will be written to process data about the accounts.

(a) Complete the class diagram.

Account
AccountNumber : STRING Balance : CURRENCY
Constructor() GetAccountNumber() GetBalance() SetAccountNumber() SetBalance()

CurrentAccount	**SavingsAccount**
....	....
Constructor()	.... GetPaymentInterval() SetPaymentInterval()

[3]

(b) Write **program code** to declare the `Account` class.

Programming language ....

Program code ....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....[5]

(c) Write **program code** to declare the `SavingsAccount` class. Do not write any get or set methods.

Programming language ....

Program code ....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....[5]

BLANK PAGE

BLANK PAGE