Cambridge Assessment International Education
Cambridge International Advanced Subsidiary and Advanced Level

CANDIDATE NAME			
CENTRE NUMBER		CANDIDATE NUMBER	

COMPUTER SCIENCE **9608/41**

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

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 **16** printed pages.

Cambridge Assessment International Education

[Turn over

1 Each student at CIE University needs a printing account to print documents from university computers.

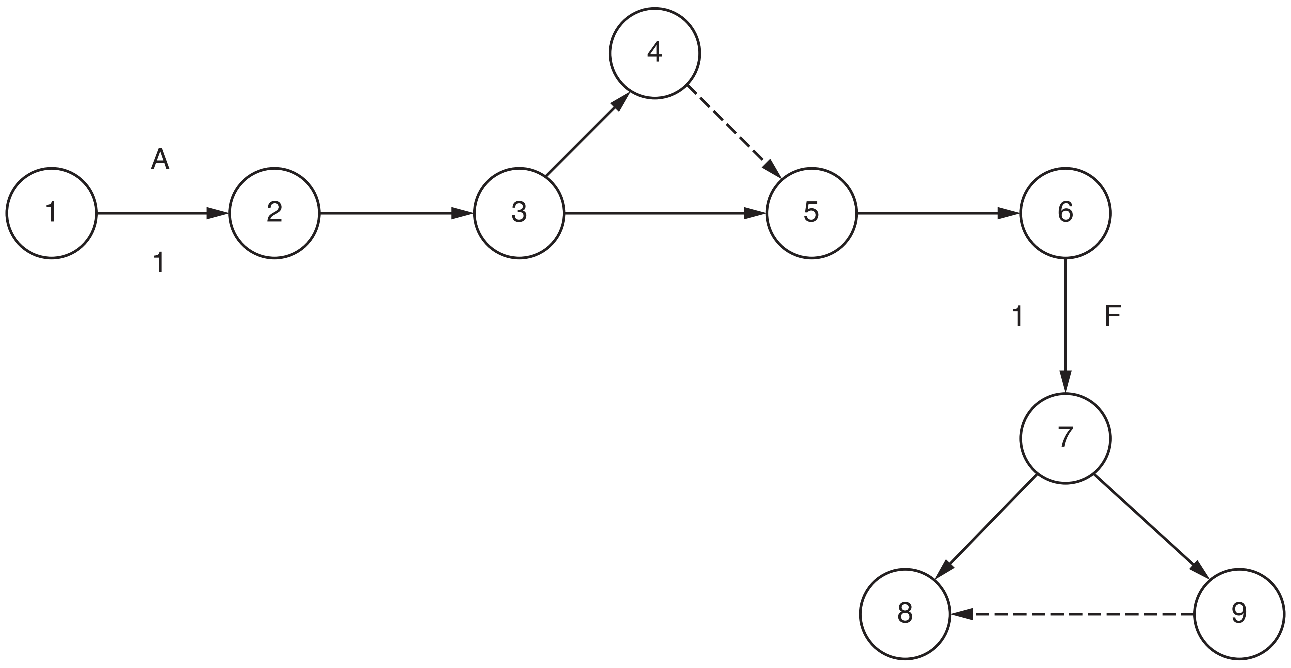
The university is developing software to manage each student's printing account and the printing process.

(a) Developing the software will include the following activities.

Activity	Description	Time in weeks	Predecessor
A	Identify requirements	1	-
B	Produce design	3	A
C	Write code	10	B
D	Test modules	7	B
E	Final system black-box testing	3	C, D
F	Install software	1	E
G	Acceptance testing	2	F
H	Create user documentation	2	F

(i) Add the correct activities and times to the following Program Evaluation Review Technique (PERT) chart for the software development.

Two activities and times have been done for you.



[6]

(ii) State what is meant by the **critical path** in a PERT chart.

....

.... [1]

(iii) Identify **and** describe a project planning technique, other than a PERT chart.

....

....

....

.... [2]

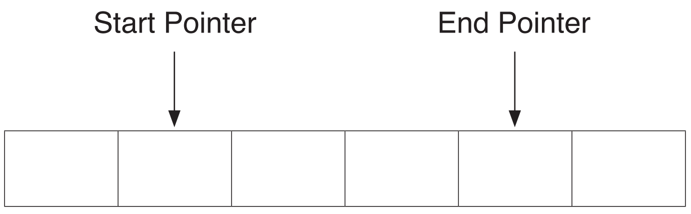
(b) When a student prints a document, a print job is created. The print job is sent to a print server.

The print server uses a queue to hold each print job waiting to be printed.

(i) The queue is circular and has six spaces to hold jobs.

The queue currently holds four jobs waiting to be printed. The jobs have arrived in the order A, B, D, C.

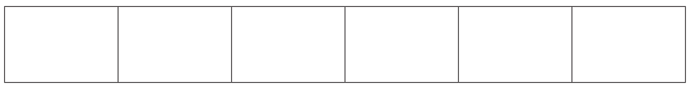
Complete the diagram to show the current contents of the queue.



[1]

(ii) Print jobs A and B are now complete. Four more print jobs have arrived in the order E, F, G, H.

Complete the diagram to show the current contents and pointers for the queue.



[3]

(iii) State what would happen if another print job is added to the queue in the status in **part (b)(ii)**.

....

.... [1]

(iv) The queue is stored as an array, `Queue`, with six elements. The following algorithm removes a print job from the queue and returns it.

Complete the following **pseudocode** for the function `Remove`.

```
FUNCTION Remove RETURNS STRING
   DECLARE PrintJob : STRING
   IF ..................................................................... = EndPointer
      THEN
         RETURN "Empty"
      ELSE
         PrintJob ← Queue[.....................................................................]
         IF StartPointer = .....................................................................
            THEN
               StartPointer ← .....................................................................
            ELSE
               StartPointer ← StartPointer + 1
         ENDIF
         RETURN PrintJob
   ENDIF
ENDFUNCTION
```

[4]

(v) Explain why the circular queue could not be implemented as a stack.

....

....

....

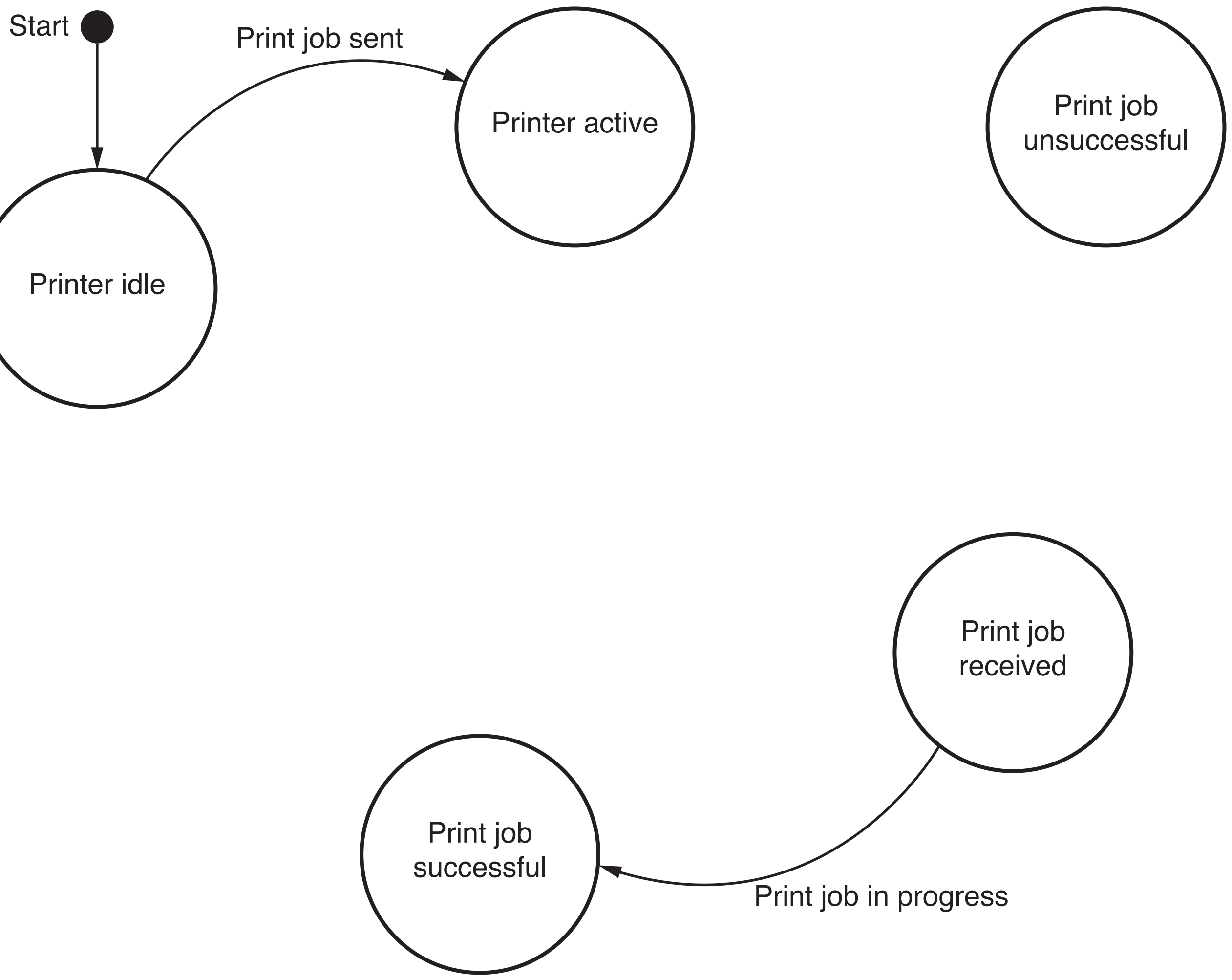
.... [2]

(c) The university wants to analyse how a printer and a print server deal with the print jobs.

The following table shows the transitions from one state to another for the process.

Current state	**Event**	**Next state**
Printer idle	Print job sent	Printer active
Printer active	Print job added to queue	Print job received
Print job received	Print job in progress	Print job successful
Print job received	Print job in progress	Print job unsuccessful
Print job successful	Check print queue	Printer active
Print job unsuccessful	Error message displayed	Printer active
Printer active	Timeout	Printer idle

Complete the state-transition diagram for the table.



[5]

(d) The university wants to assess troubleshooting issues with a printer. It wants to use a decision table to do this.

The troubleshooting actions are:

- check the connection from computer to printer, if the error light is flashing **and** the document has not been printed
- check the ink status, if the quality is poor
- check whether there is a paper jam, if the error light is flashing **and** the document has not been printed
- check the paper size selected, if the paper size is incorrect.

(i) Describe the purpose of a decision table.

....

....

....

.... [2]

(ii) Complete the rules for the actions in the following decision table.

		Rules							
Conditions	Document printed but the quality is poor	Y	Y	Y	Y	N	N	N	N
	Error light is flashing on printer	Y	Y	N	N	Y	Y	N	N
	Document printed but paper size is incorrect	Y	N	Y	N	Y	N	Y	N
Actions	Check connection from computer to printer								
	Check ink status								
	Check if there is a paper jam								
	Check the paper size selected								

[4]

(iii) Simplify your solution by removing redundancies.

		Rules							
Conditions	Document printed but the quality is poor								
	Error light is flashing on printer								
	Document printed but paper size is incorrect								
Actions	Check connection from computer to printer								
	Check ink status								
	Check if there is a paper jam								
	Check the paper size selected								

[5]

(e) There are 1000 students at the university. They will each require a printing account.

Students need to buy printing credits that will be added to their account. Each page printed uses one printing credit.

The university needs software to keep track of the number of printing credits each student has in their account. The university has decided to implement the software using object-oriented programming (OOP).

The following diagram shows the design for the class `PrintAccount`. This includes the attributes and methods.

```
                               PrintAccount
FirstName  :  STRING    //  parameter sent to Constructor()
LastName   :  STRING    //  parameter sent to Constructor()
PrintID    :  STRING    //  parameter sent to Constructor()
Credits    :  INTEGER   //  initialised to 50

Constructor()     //  instantiates an object of the PrintAccount class,
                  //  and assigns initial values to the attributes
GetName()         //  returns FirstName and LastName concatenated
                  //  with a space between them
GetPrintID()      //  returns PrintID
SetFirstName()    //  sets the FirstName for a student
SetLastName()     //  sets the LastName for a student
SetPrintID()      //  sets the PrintID for a student
AddCredits()      //  increases the number of credits for a student
RemoveCredits()   //  removes credits from a student account
```

(i) Write **program code** for the `Constructor()` method.

Programming language ....

Program code

....

....

....

....

....

....

....

....

.... [4]

(ii) Write **program code** for the `SetFirstName()` method.

Programming language ....

Program code

....

....

....

....

.... [2]

(iii) Write **program code** for the `GetName()` method.

Programming language ....

Program code

....

....

....

....

....

.... [2]

(iv) The method `AddCredits()` calculates the number of printing credits a student buys and adds the printing credits to the student's account.

- Credits cost $1 for 25 credits.
- If a student buys $20 or more of credits in a single payment, they receive an extra 50 credits.
- If a student buys between $10 and $19 (inclusive) of credits in a single payment, they receive an extra 25 credits.

Payment from a student is stored in the variable `MoneyInput`. This is passed as a parameter.

Write **program code** for `AddCredits()`. Use constants for the values that do not change.

Programming language ....

Program code

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

.... [6]

(v) A global array, `StudentAccounts`, stores 1000 instances of `PrintAccount`.

Write **pseudocode** to declare the array `StudentAccounts`.

....

.... [2]

(vi) The main program has a procedure, `CreateID()`, that:

- takes the first name and last name as parameters
- creates `PrintID` that is a concatenation of:
  - the first three letters of the first name in lower case
  - the first three letters of the last name in lower case
  - the character '1'
    for example, the name Bill Smith would produce `"bilsmi1"`
- checks if the `PrintID` created already exists in the global array `StudentAccounts`:
  - If `PrintID` does not exist, it creates an instance of `PrintAccount` in the next free index in `StudentAccounts`.
  - If `PrintID` does exist, the number is incremented until a unique ID is created, for example, `"bilsmi2"`. It then creates an instance of `PrintAccount` in the next free index in `StudentAccounts`.

The global variable `NumberStudents` stores the number of print accounts that have currently been created.

Write **program code** for the procedure `CreateID()`. Do not write the procedure header.

Programming language ....

Program code

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

[8]

2 The following table shows part of the instruction set for a processor, which 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.
`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.

A programmer writes a program that multiplies two numbers together and outputs the result. The numbers are stored as `NUMONE` and `NUMTWO`.

The programmer has started to write the program in the following table. The comment column contains explanations for some of the missing program instructions and data.

Complete the program using the given instruction set.

Label	Op code	Operand	Comment
`LOOP:`			`// load the value from ANSWER`
			`// add the value from NUMONE`
			`// load the value from COUNT`
			`// increment the Accumulator`
			`// is NUMTWO = COUNT ?`
			`// if false, jump to LOOP`
			`// load the value from ANSWER`
			`// output ANSWER to the screen`
			`// end of program`
`NUMONE:`	`2`		
`NUMTWO:`	`4`		
`COUNT:`	`0`		
`ANSWER:`	`0`		

[9]

3 Software may not perform as expected. One reason for this is that a syntax error exists in the code.

Identify **three other** reasons why software may not perform as expected.

1 ....

....

2 ....

....

3 ....

....

[3]

4 The following table contains definitions related to testing terminology.

Complete the table with the correct testing term to match the definition.

Definition	**Term**
Software is tested by an in-house team of dedicated testers.	....
Software is tested by the customer before it is signed off.	....
Software is tested by a small selection of users before general release.	....

[3]