Cambridge International AS & A Level

Cambridge Assessment International Education
Cambridge International Advanced Subsidiary and Advanced Level

CANDIDATE NAME

CENTRE NUMBER | | | | | |

CANDIDATE NUMBER | | | | |

COMPUTER SCIENCE **9608/42**

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 **17** printed pages and **3** blank pages.

DC (LEG/FC) 168337/3

Cambridge Assessment International Education

[Turn over

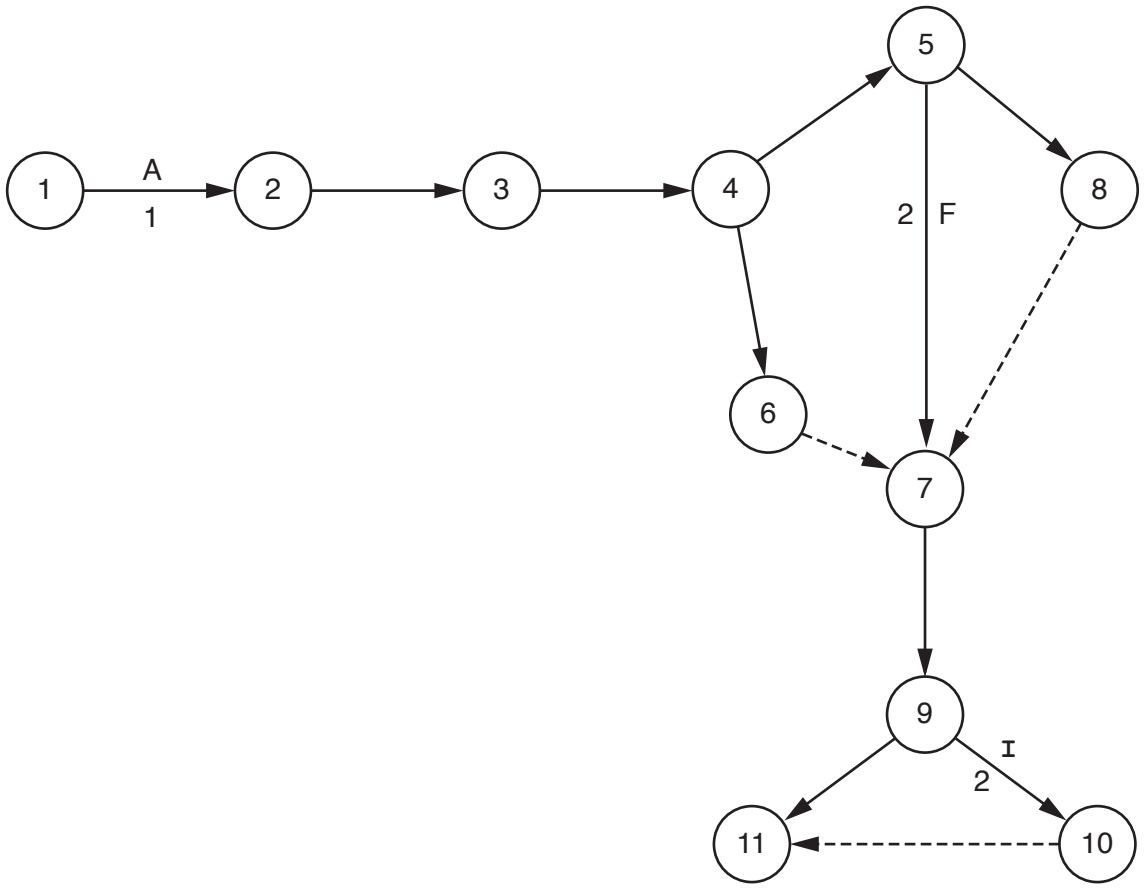
1 A technology company needs software to calculate how much each employee should be paid.

(a) Developing the software will involve the following activities:

Activity	Description	Time to complete (weeks)	Predecessor
A	Identify requirements	1	–
B	Observe current system	1	A
C	Create algorithm design	3	B
D	Write code	10	C
E	Test modules	7	C
F	White box testing	2	D
G	Black box testing	3	D
H	Install software	1	E, F, G
I	Acceptance testing	2	H
J	Create user documentation	2	H

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

Three of the activities and times have been done for you.



[7]

(ii) The dashed line connecting nodes 10 and 11 indicates a dummy activity.

State the purpose of a dummy activity.

.....

..... [1]

(b) A bonus payment may be added to an employee's salary. A pension payment may also be subtracted from an employee's salary.

The company needs to assess what additions and subtractions should be made to the salary of each employee. There are three conditions to check:

- If the employee has worked a public holiday, they receive a 3% bonus payment.
- If the employee has worked 160 or more hours in a month, they receive an additional 5% bonus payment.
- If the employee pays into a pension, the company subtracts 4% for the pension payment.

Complete the decision table to show the additions and subtractions.

		Rules							
Conditions	**Public holiday**	Y	Y	Y	Y	N	N	N	N
	Hours >= 160	Y	Y	N	N	Y	Y	N	N
	Pension	Y	N	Y	N	Y	N	Y	N
Actions	**3% bonus payment**								
	5% bonus payment								
	4% pension payment								

[3]

(c) The company decides to implement a program for the software using object-oriented programming (OOP).

Each employee has a unique employee ID, name, address and date of birth. There are two types of employee: salary and apprenticeship.

Salaried employees are paid a fixed monthly payment. The hours a salary employee works in a month are recorded to calculate bonus payments. They may receive bonus payments and make pension payments (given in **part(b)**).

Apprenticeship employees are paid weekly. They receive an hourly rate of pay. Apprenticeship employees do not receive bonus payments or make pension payments.

(i) Complete the following class diagram for the program.

Employee
EmployeeID : STRING Name : STRING Address : STRING DateOfBirth : Date
Constructor() GetEmployeeID() GetName() GetAddress() GetDateOfBirth() SetEmployeeID() SetName() SetAddress() SetDateOfBirth()

SalaryEmployee
MonthlyPayment : CURRENCY HoursThisMonth: REAL PublicHoliday : BOOLEAN Pension : BOOLEAN
Constructor() GetMonthlyPayment() GetHoursThisMonth() GetPublicHoliday() GetPension() SetMonthlyPayment() SetHoursThisMonth() SetPublicHoliday() SetPension()

ApprenticeshipEmployee
.....
.....
.....
.....
GetHourlyRate() GetHoursThisWeek() SetHourlyRate()

[3]

(ii) Write **program code** for the `Constructor()` in the `Employee` class.

All properties are sent as parameters.

Programming language .....

Program code

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [4]

(iii) Write **program code** for the `GetEmployeeID()` method in the `Employee` class.

The get method returns the value of the `EmployeeID` property.

Programming language .....

Program code

.....

.....

.....

.....

.....

..... [2]

(iv) Write **program code** for the `SetEmployeeID()` method in the `Employee` class.

The set method takes the new value as its parameter.

Programming language .....

Program code

.....

.....

.....

.....

.....

..... [2]

(v) Write **program code** for the `SetPension()` method in the `SalaryEmployee()` class.

- The method takes a new value for `Pension` as a parameter.
- If the parameter's value is valid (it is `TRUE` or `FALSE`), the method returns `TRUE` and sets the parameter's value.
- Otherwise the method returns `FALSE` and does not set `Pension`.

Programming language .....

Program code

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [4]

Question 1 continues on the next page.

(vi) A `SalaryEmployee` is paid a fixed monthly payment.

- If the employee has worked a public holiday, they receive a 3% bonus payment. This is calculated from their `MonthlyPayment`.
- If the employee has worked 160 or more hours in a month, they receive an additional 5% bonus payment, calculated from their `MonthlyPayment`.
- If the employee pays into a pension, 4% will be subtracted from their `MonthlyPayment`.

Monthly salary is the final payment the employee receives.

For example, Chris is a `SalaryEmployee`. His fixed `MonthlyPayment` is $1000. He has worked a public holiday. He has worked 165 hours this month. He pays into a pension.

- The public holiday bonus is $30 (3% of $1000)
- The hours worked bonus payment is $50 (5% of $1000)
- The pension payment is $40 (4% of $1000)

Chris's monthly salary is calculated as ($1000 + $30 + $50) – $40 = $1040

The function `CalculateMonthlySalary()` is used to calculate the monthly salary. It:

- takes a `SalaryEmployee` as a parameter
- calculates the bonus payments and pension payment
- outputs the pension payment and total bonus payment
- calculates and returns the monthly salary.

Write **program code** for the function `CalculateMonthlySalary()`.

Programming language .....

Program code

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

[8]

(d) Noona describes an example of a feature of object-oriented programming (OOP). She says:

"One method exists in the parent class but is overwritten in the child class, to behave differently."

Identify the feature Noona has described.

..... [1]

2 The number of cars that cross a bridge is recorded each hour. This number is placed in a circular queue before being processed.

(a) The queue is stored as an array, `NumberQueue`, with eight elements. The function `AddToQueue` adds a number to the queue. `EndPointer` and `StartPointer` are global variables.

Complete the following **pseudocode** algorithm for the function `AddToQueue`.

```
FUNCTION AddToQueue(Number : INTEGER) RETURNS BOOLEAN

   DECLARE TempPointer : INTEGER

   CONSTANT FirstIndex = 0

   CONSTANT LastIndex = .....

   TempPointer ← EndPointer + 1

   IF ..... > LastIndex

       THEN

          TempPointer ← .....

   ENDIF

   IF TempPointer = StartPointer

       THEN

          RETURN .....

       ELSE

          EndPointer ← TempPointer

          NumberQueue[EndPointer] ← .....

          RETURN TRUE

   ENDIF

ENDFUNCTION
```

[5]

(b) Describe how a number is removed from the circular queue to be processed.

.....

.....

.....

.....

.....

.....

.....

..... [4]

(c) A queue is one example of an Abstract Data Type (ADT).

Identify **three other** Abstract Data Types.

1 .....

2 .....

3 .....

[3]

3 A company wants to test a program to check that it works. They can use different types of test data to do this.

(a) Identify **three** different types of test data that the company can use.

1 .....

2 .....

3 .....

[3]

(b) The programmer will make use of debugging features, when building and testing a program.

(i) Two debugging features are described in the table.

Write the correct name for **each** debugging feature.

Description	Name of debugging feature
A point where the program can be halted to see if the program works to this point.	.....
One statement is executed and then the program waits for input from the programmer to move on to the next statement.	.....

[2]

(ii) Identify **and** describe **one other** debugging feature.

Debugging feature .....

Description .....

.....

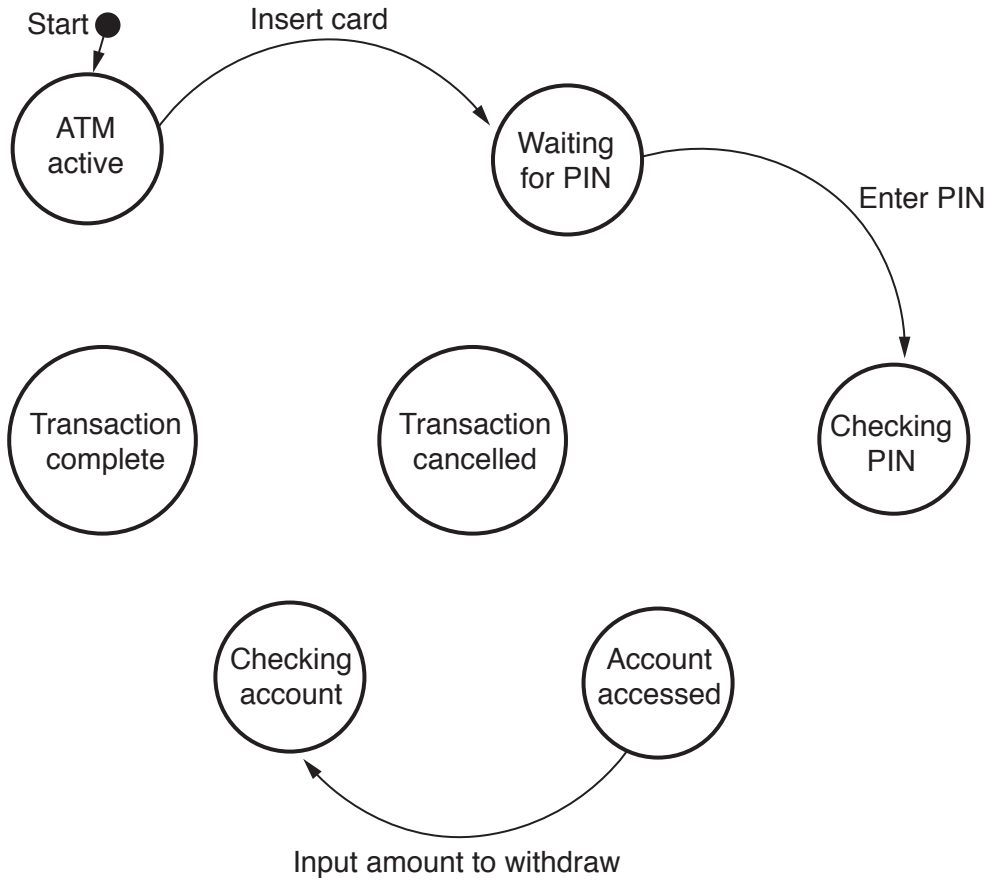
[2]

4 A bank wants to analyse how an automated teller machine (ATM) deals with transactions.

The following state-transition table shows the transitions from one state to another for a transaction.

Current state	Event	Next state
ATM active	Insert card	Waiting for PIN
Waiting for PIN	Enter PIN	Checking PIN
Waiting for PIN	Cancel selected	Transaction cancelled
Checking PIN	PIN valid	Account accessed
Checking PIN	PIN invalid	Waiting for PIN
Account accessed	Cancel selected	Transaction cancelled
Account accessed	Input amount to withdraw	Checking account
Checking account	Funds available	Transaction complete
Transaction complete	Return card and dispense cash	ATM active
Checking account	Funds not available	Account accessed
Transaction cancelled	Return card	ATM active

Complete the state-transition diagram to correspond with the table.



[8]

5 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) A programmer needs a program that multiplies a binary number by 4.

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

Write the program using the given instruction set.

Label	Instruction		Comment
	Op code	**Operand**	
			`// load contents of NUMBER`
			`// perform shift to multiply by 4`
			`// store contents of ACC in NUMBER`
			`// end program`
`NUMBER:`	`B00110110`		

[5]

Note:

`#` denotes immediate addressing
`B` denotes a binary number, e.g. `B01001010`
`&` denotes a hexadecimal number, e.g. `&4A`

(b) A programmer needs a program that counts the number of lower case letters in a string.

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

Complete the program using the given instruction set. A copy of the instruction set is provided on the opposite page.

Label	Instruction		Comment
	Op code	Operand	
	LDR	#0	// initialise Index Register to 0
START:			// load the next value from the STRING
			// perform bitwise AND operation with MASK
			// check if result is equal to MASK
			// if FALSE, jump to UPPER
			// increment COUNT
UPPER:	INC	IX	// increment the Index Register
			// decrement LENGTH
			// is LENGTH = 0 ?
			// if FALSE, jump to START
	END		// end program
MASK:	B00100000		// if bit 5 is 1, letter is lower case
COUNT:	0		
LENGTH:	5		
STRING:	B01001000		// ASCII code for 'H'
	B01100001		// ASCII code for 'a'
	B01110000		// ASCII code for 'p'
	B01110000		// ASCII code for 'p'
	B01011001		// ASCII code for 'Y'

[8]

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.

BLANK PAGE

BLANK PAGE

BLANK PAGE