Cambridge International **A Level**

Cambridge International Examinations
Cambridge International Advanced Level

CANDIDATE NAME

CENTRE NUMBER

CANDIDATE NUMBER

6250551904

COMPUTER SCIENCE **9608/42**

Paper 4 Further Problem-solving and Programming Skills **May/June 2016**

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 **19** printed pages and **1** blank page.

CAMBRIDGE International Examinations

1 A linked list abstract data type (ADT) is to be used to store and organise surnames.

This will be implemented with a 1D array and a start pointer. Elements of the array consist of a user-defined type. The user-defined type consists of a data value and a link pointer.

Identifier	Data type	Description
LinkedList	RECORD	User-defined type
Surname	STRING	Surname string
Ptr	INTEGER	Link pointers for the linked list

(a) (i) Write **pseudocode** to declare the type `LinkedList`.

.....

.....

.....

.....[3]

(ii) The 1D array is implemented with an array `SurnameList` of type `LinkedList`.

Write the **pseudocode** declaration statement for `SurnameList`. The lower and upper bounds of the array are 1 and 5000 respectively.

.....[2]

(b) The following surnames are organised as a linked list with a start pointer `StartPtr`.

`StartPtr: 3`

	1	2	3	4	5	6	...	5000
Surname	Liu	Yang	Chan	Wu	Zhao	Huang	...	
Ptr	4	5	6	2	0	1	...	

State the value of the following:

(i) `SurnameList[4].Surname` .....[1]

(ii) `SurnameList[StartPtr].Ptr` .....[1]

(c) Pseudocode is to be written to search the linked list for a surname input by the user.

Identifier	Data type	Description
ThisSurname	STRING	The surname to search for
Current	INTEGER	Index to array SurnameList
StartPtr	INTEGER	Index to array SurnameList. Points to the element at the start of the linked list

(i) Study the pseudocode in **part (c)(ii)**.

Complete the table above by adding the missing identifier details. [2]

(ii) Complete the pseudocode.

```
01  Current ← .....
02  IF Current = 0
03     THEN
04        OUTPUT .....
05     ELSE
06        IsFound ← .....
07        INPUT ThisSurname
08        REPEAT
09           IF ..... = ThisSurname
10              THEN
11                 IsFound ← TRUE
12                 OUTPUT "Surname found at position ", Current
13              ELSE
14                 // move to the next list item
15                 .....
16           ENDIF
17        UNTIL IsFound = TRUE OR .....
18        IF IsFound = FALSE
19           THEN
20              OUTPUT "Not Found"
21        ENDIF
22  ENDIF
```

[6]

2 **(a)** **(i)** State what is meant by a recursively defined procedure.

.....

.....[1]

(ii) Write the line number from the pseudocode shown in **part (b)** that shows the procedure `X` is recursive. ..... [1]

(b) The recursive procedure `X` is defined as follows:

```
01  PROCEDURE X(Index, Item)
02     IF MyList[Index] > 0
03        THEN
04           IF MyList(Index) >= Item
05              THEN
06                 MyList[Index] ← MyList[Index + 1]
07           ENDIF
08           CALL X(Index + 1, Item)
09     ENDIF
10  ENDPROCEDURE
```

An array `MyList` is used to store a sorted data set of non-zero integers. Unused cells contain zero.

	1	2	3	4	5	6	7	8	9	10
MyList	3	5	8	9	13	16	27	0	0	0

(i) Complete the trace table for the dry-run of the pseudocode for the procedure `CALL X(1, 9)`.

		MyList									
Index	**Item**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	9	3	5	8	9	13	16	27	0	0	0

[4]

(ii) State the purpose of procedure `X` when used with the array `MyList`.

.....

.....[1]

3 A car hire company hires cars to customers. Each time a car is hired, this is treated as a transaction.

For each transaction, the following data are stored.

For the customer:

- customer name
- ID number

For the hire:

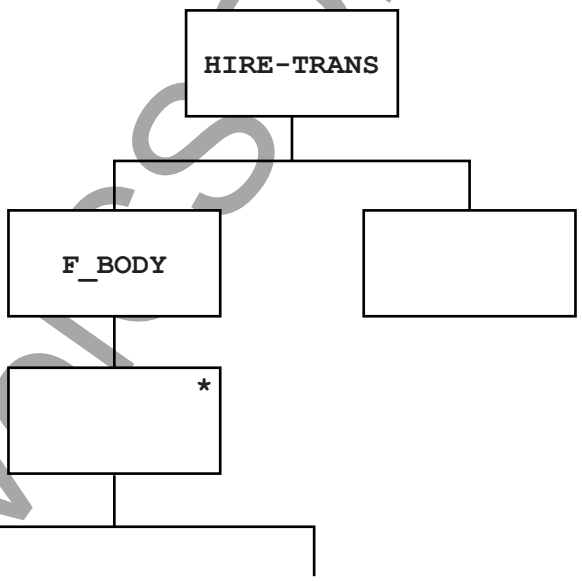
- car registration
- hire start date
- number of days hired

The transaction data are stored in a text file `HIRE-TRANS`. The file is made up of a file body, `F_BODY`, and a file trailer, `F_TRAILER`.

`F_BODY` has one transaction, `TRANS`, on each line.

(a) The first step in Jackson Structured Programming (JSP) design is to produce a JSP data structure diagram.

Complete the following JSP data structure diagram.



[7]

(b) The computer system will produce many printed reports.

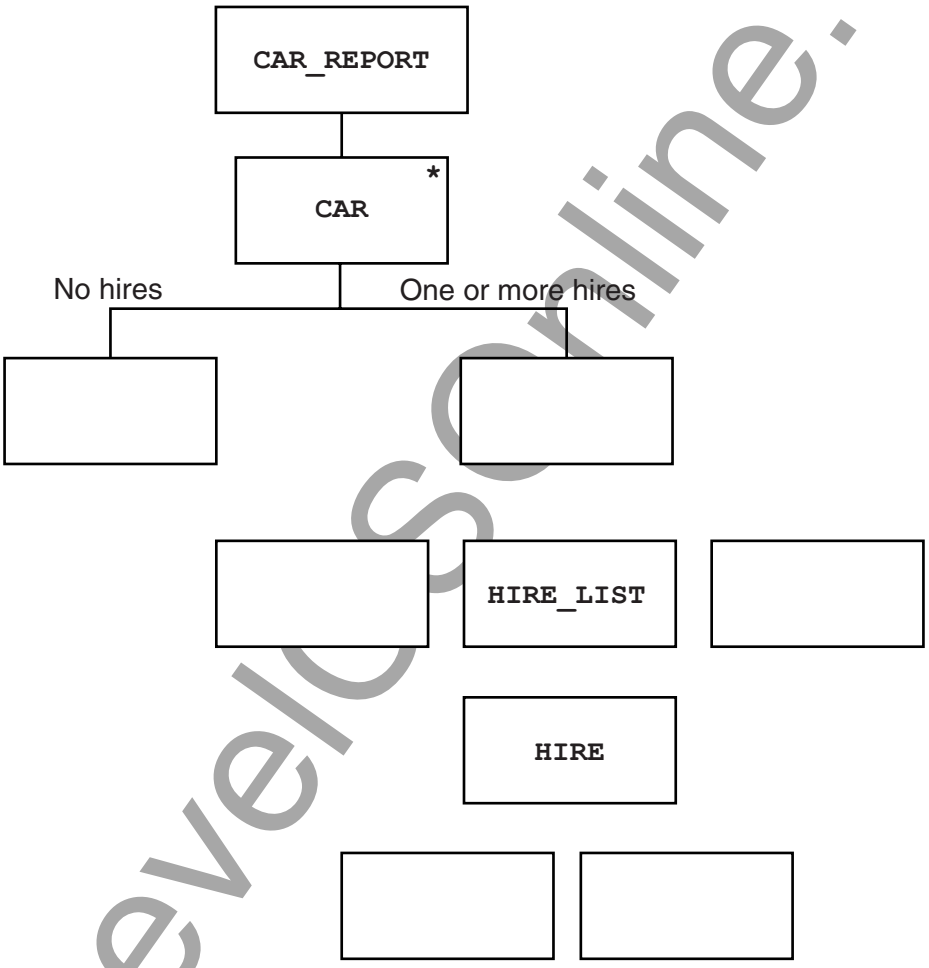
One report is `CAR_REPORT`. This displays all hire data for all cars.

For each car, the following data are displayed:

- the car data
- a list of all the hires
- the total number of hires

A car with zero hires is not included on the report.

Complete the following `CAR_REPORT` JSP data structure diagram.



[5]

4 When a car reaches a certain age, a safety assessment has to be carried out. A car's brakes and tyres must be tested. The tyre test result and the brakes test result for each car are recorded. If the car passes the assessment, a safety certificate is issued.

Cars have a unique three-character registration.

The following knowledge base is used:

```
01  car(a05).
02  car(h04).
03  car(a03).
04  car(h07).
05  car(a23).
06  car(p05).
07  car(b04).
08  carRegYear(a05, 2015).
09  carRegYear(h04, 2013).
10  carRegYear(a03, 2008).
11  carRegYear(h07, 2011).
12  carRegYear(a23, 2008).
13  carRegYear(p05, 2014).
14  carRegYear(b04, 2014).
15  testBrakes(h07, pass).
16  testTyres(h07, fail).
17  testBrakes(a03, fail).
18  testTyres(a03, fail).
19  testBrakes(a23, pass).
20  testTyres(a23, pass).
21  carAssessmentDue if carRegYear(Car, RegYear)
                                  and RegYear <= DeadlineYear.
22  issueCertificate(Car) if testTyres(Car, Result) and
                             testBrakes(Car, Result) and Result = pass.
```

(a) (i) `DeadlineYear` is assigned value `2011`.

Identify the car registrations for cars which are due to be tested.

.....[1]

(ii) State how clause `22` determines whether or not a safety certificate will be issued.

.....

.....[1]

(b) If a car fails one of the two tests, a retest is allowed.

Write a new rule for this.

`retestAllowed(.....) if .....`

.....

.....[3]

(c) Logic programming uses a data structure called a list.

A new fact is added to the knowledge base.

```
23 carList = [a03,p05,b04,h04,h07,a23].
```

The following notation and operators are to be used with a list:

> `[X|Y]` denotes a list with:
>
> - `X` the first list element
> - `Y` the list consisting of the remaining list elements
>
> `[]` denotes an empty list

(i) The list `[a07,p03]` is denoted by `[A|B]`

State the value of `A` and `B`.

`A` = ...

`B` = ... [2]

(ii) The lists `[c03,d02,n05|C]` and `[c03,d02,n05,p05,m04]` are identical.

State the value of `C`.

`C` = ... [1]

(iii) The list `[a06,a02]` is denoted by `[D,E|F]`

State the value of `F`.

`F` = ... [1]

(d) The predicate `conCatCompare` is defined as a rule and returns `TRUE` or `FALSE` as follows:

> `conCatCompare(X, Y, Z)`
>
> Concatenates the lists `X` and `Y` and compares the new list with list `Z`.
>
> If equal, the clause evaluates to `TRUE`, otherwise `FALSE`.

Consider the clause:

```
conCatCompare(X, Y, [a7,b6,c4])
```

If:

- the clause evaluates to `TRUE`
- and `Y` represents the list `[a7, b6, c4]`

State the value of `X`.

`X` = ...[1]

5 **(a)** A program calculates the exam grade awarded from a mark input by the user. The code is written as a function `CalculateGrade`.

The function:

- has a single parameter `Mark` of `INTEGER` data type
- returns the grade awarded `Grade` of `STRING` data type

The logic for calculating the grade is as follows:

Mark	Grade
Under 40	FAIL
40 and over and under 55	PASS
55 and over and under 70	MERIT
70 and over	DISTINCTION

The programmer designs the following table for test data:

Mark	Description	Expected result (Grade)
	Normal	
	Abnormal	
	Extreme/Boundary	

(i) Complete the table above. [3]

(ii) State why this table design is suitable for black box testing.

.....

.....[1]

(b) When designing and writing program code, explain what is meant by:

- an exception
- exception handling

.....

.....

.....

.....

.....

.....[3]

(c) A program is to be written to read a list of exam marks from an existing text file into a 1D array.

Each line of the file stores the mark for one student.

State **three** exceptions that a programmer should anticipate for this program.

1 .....

.....

2 .....

.....

3 .....

.....[3]

(d) The following pseudocode is to read two numbers:

```
01  DECLARE Num1   : INTEGER
02  DECLARE Num2   : INTEGER
03  DECLARE Answer : INTEGER
04  TRY
05     OUTPUT "First number..."
06     INPUT Num1
07     OUTPUT "Second number..."
08     INPUT Num2
09     Answer ← Num1 / (Num2 - 6)
10     OUTPUT Answer
11  EXCEPT ThisException : EXCEPTION
12     OUTPUT ThisException.Message
13  FINALLY
14     // remainder of the program follows

…

29
30  ENDTRY
```

The programmer writes the corresponding program code.

A user inputs the number 53 followed by 6. The following output is produced:

```
First number...53
Second number...6
Arithmetic operation resulted in an overflow
```

(i) State the pseudocode line number which causes the exception to be raised.

..... [1]

(ii) Explain the purpose of the pseudocode on lines `11` and `12`.

.....

.....

.....

.....

.....

.....[3]

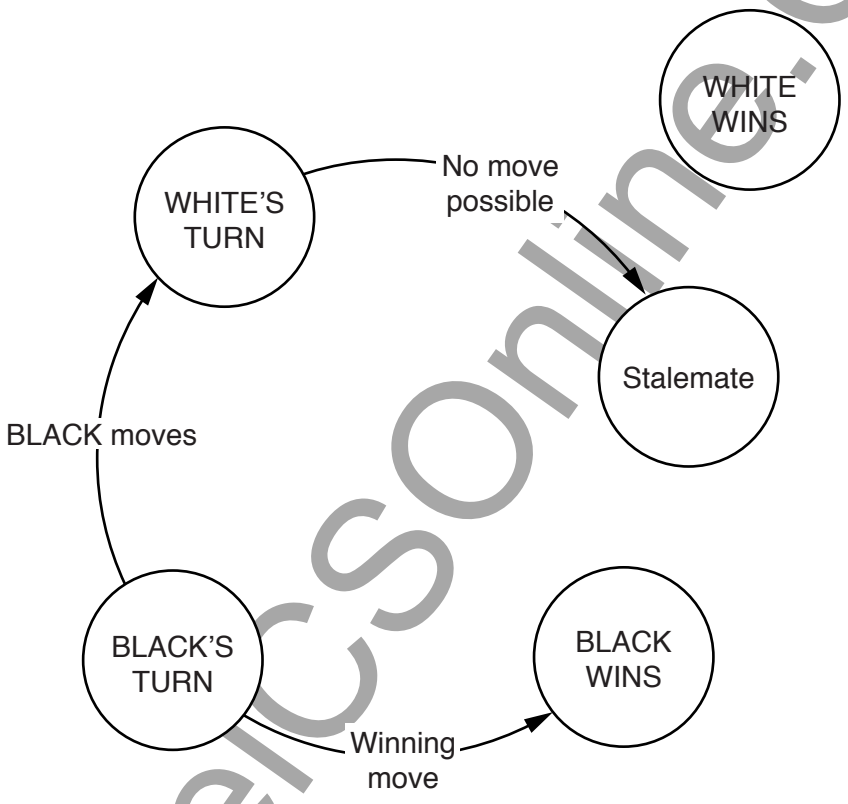
6 In a board game, one player has white pieces and the other player has black pieces. Players take alternate turns to move one of their pieces. White always makes the first move.

The game ends if:

- a player is unable to make a move when it is their turn. In this case, there is no winner. This is called 'stalemate'.
- a player wins the game as a result of their last move and is called a 'winner'.

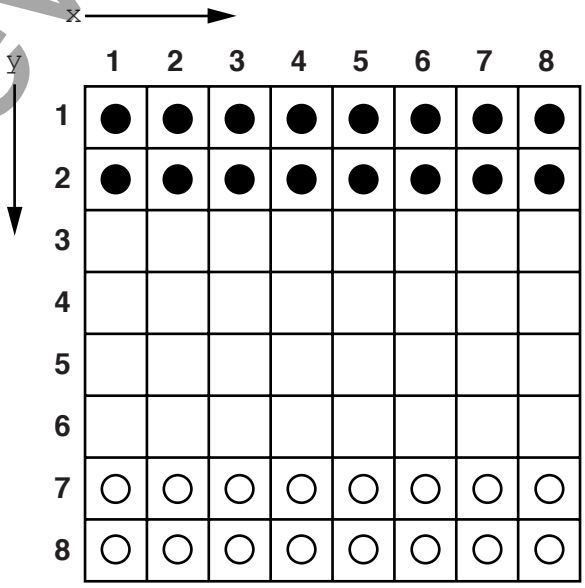
(a) A state-transition diagram is drawn to clarify how the game is played.

Complete the following state-transition diagram.



[4]

(b) The layout of the board at the start of the game is shown below:



The programmer decides to use a 2D array to represent the board. The index numbering to be used is as shown.

Each square on the board is either occupied by one piece only, or is empty.

The data stored in the array indicate whether or not that square is occupied, and if so, with a black piece or a white piece.

(i) Write **program code** to initialise the contents of the array to represent the board at the start of the game. Use characters as follows for each square:

- `'B'` represents a black piece ●
- `'W'` represents a white piece ○
- `'E'` represents an empty square

Visual Basic and Pascal: You should include the declaration statements for variables.
Python: You should show a comment statement for each variable used with its data type.

Programming language .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....[4]

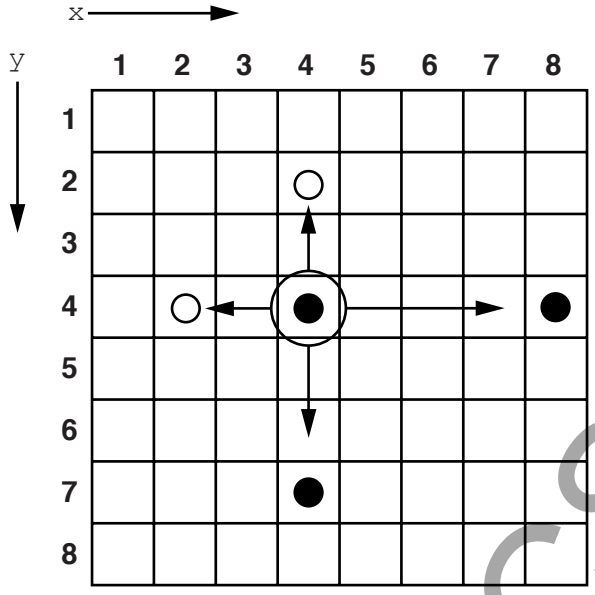
(ii) When a piece is to be moved, a procedure will calculate and output the possible destination squares for the moving piece.

A piece can move one or more squares, in the `x` or `y` direction, from its current position.

This will be a move:

- either to an empty square, with no occupied squares on the way
- or to a square containing a piece belonging to another player, with no occupied squares on the way. The other player's piece is then removed.

For example, for the circled black piece there are nine possible destination squares. Each of the two destination squares contains a white piece which would be removed.



The program requires a procedure `ValidMoves`.

It needs three parameters:

- `PieceColour` – colour of the moving piece
- `xCurrent` – current `x` position
- `yCurrent` – current `y` position

The procedure will calculate all possible destination squares **in the `x` direction only**.

Example output for the circled black piece is:

```
Possible moves are:
Moving LEFT
3 4
2 4 REMOVE piece
Moving RIGHT
5 4
6 4
7 4
```

Write **program code** for procedure `ValidMoves` with the following procedure header:

```
PROCEDURE ValidMoves(PieceColour : CHAR, xCurrent : INTEGER,
                                          yCurrent : INTEGER).
```

Visual Basic and Pascal: You should include the declaration statements for variables.
Python: You should show a comment statement for each variable used with its data type.

Programming language .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....[5]

(c) The problem is well suited to an object-oriented design followed by object-oriented programming.

(i) Describe how classes and objects could be used in this problem.

.....

.....

.....

.....

.....[2]

(ii) For a class you identified in **part(c)(i)**, state **two** properties and **two** methods.

Class .....

Properties

1 .....

2 .....

Methods

1 .....

2 .....

[2]

BLANK PAGE