Cambridge International A Level

Cambridge International Examinations
Cambridge International Advanced Level

CANDIDATE NAME

CENTRE NUMBER

CANDIDATE NUMBER

COMPUTER SCIENCE **9608/42**

Paper 4 Further Problem-solving and Programming Skills **May/June 2015**

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 International Examinations

[Turn over

Throughout the paper you will be asked to write either **pseudocode** or **program code**.

Complete the statement to indicate which high-level programming language you will use.

Programming language .....

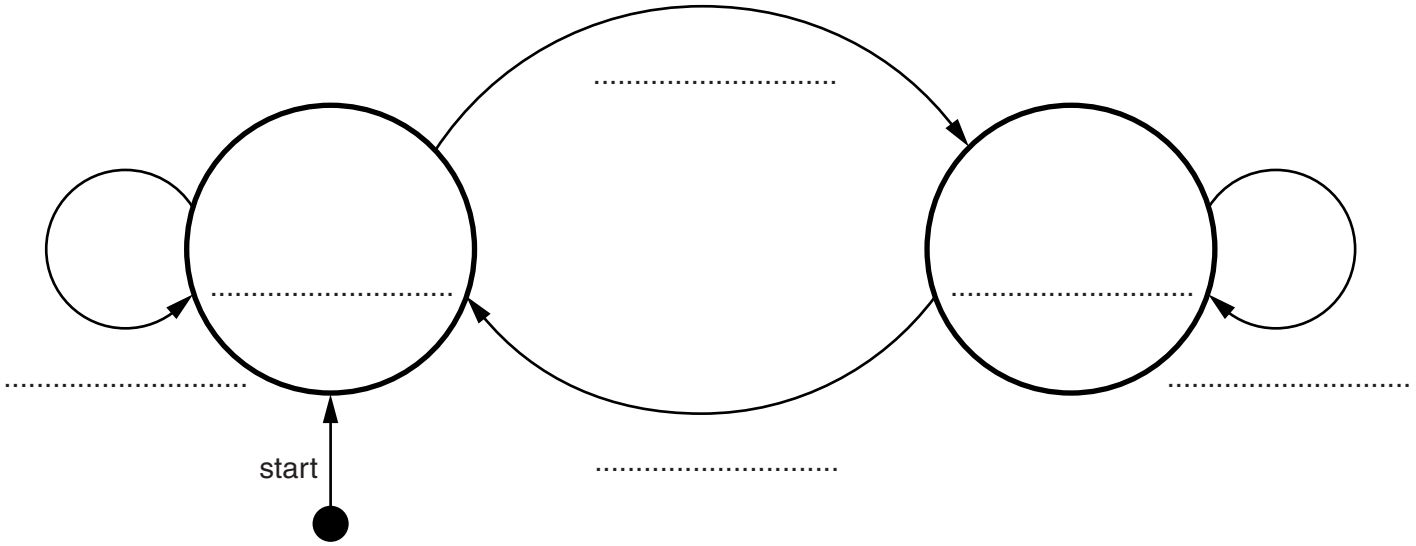
1 A turnstile is a gate which is in a locked state. To open it and pass through, a cust... a coin into a slot on the turnstile. The turnstile then unlocks and allows the customer to... turnstile and pass through the gate.

After the customer has passed through, the turnstile locks again. If a customer pushes the turnsti... while it is in the locked state, it will remain locked until another coin is inserted.

The turnstile has two possible states: **locked** and **unlocked**. The transition from one state to another is as shown in the table below.

Current state	**Event**	**Next state**
Locked	Insert coin	Unlocked
Locked	Push	Locked
Unlocked	Attempt to insert coin	Unlocked
Unlocked	Pass through	Locked

Complete the state transition diagram for the turnstile:



[5]

2 A declarative programming language is used to represent the knowledge base shown.

```
01  capital_city(amman).
02  capital_city(beijing).
03  capital_city(brussels).
04  capital_city(cairo).
05  capital_city(london).
06  city_in_country(amman, jordan).
07  city_in_country(shanghai, china).
08  city_in_country(brussels, belgium).
09  city_in_country(london, uk).
10  city_in_country(manchester, uk).
11  country_in_continent(belgium, europe).
12  country_in_continent(china, asia).
13  country_in_continent(uk, europe).
14  city_visited(amman).
15  city_visited(beijing).
16  city_visited(cairo).
```

These clauses have the following meaning:

Clause	Explanation
01	Amman is a capital city
06	Amman is a city in the country of Jordan
11	Belgium is a country in the continent of Europe
14	The travel writer visited Amman

(a) More facts are to be included.

The travel writer visited the city of Santiago which is the capital city of Chile, in the continent of South America.

Write additional clauses to record this.

17 .....

.....

18 .....

.....

19 .....

.....

20 .....

..... [4]

(b) Using the variable `ThisCountry`, the goal

```
country_in_continent(ThisCountry, europe)
```

returns

```
ThisCountry = belgium, uk
```

Write the result returned by the goal:

```
city_in_country(ThisCity, uk)
```

`ThisCity =` .....

..... [2]

(c) Complete the rule below to list the countries the travel writer has visited.

```
countries_visited(ThisCountry)
```

`IF` .....

.....

.....

.....

..... [4]

3 A shop gives some customers a discount on goods totalling more than $20.
The discounts are:

- 5% for goods totalling more than $100
- 5% with a discount card
- 10% with a discount card and goods totalling more than $100

(a) Complete the decision table.

Conditions	goods totalling more than $20	Y	Y	Y	Y	N	N	N	N
	goods totalling more than $100	Y	Y	N	N	Y	Y	N	N
	have discount card	Y	N	Y	N	Y	N	Y	N
Actions	No discount								
	5% discount								
	10% discount								

[4]

(b) Simplify your solution by removing redundancies.

Conditions	goods totalling more than $20								
	goods totalling more than $100								
	have discount card								
Actions	No discount								
	5% discount								
	10% discount								

[5]

(c) The simplified table produced in **part (b)** is used as a design for program code.

The following identifier table shows the parameters to be passed to the function `Dis`
This function returns the discount amount as an integer.

Identifier	Data type
`GoodsTotal`	`INTEGER`
`HasDiscountCard`	`BOOLEAN`

Write **program code** for this function.

Programming language .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [6]

4 A payroll program is to be written using an object-oriented programming language. A class is designed. Two subclasses have been identified:

- `HourlyPaidEmployee` who is paid a monthly wage calculated from their hourly rate and the number of hours worked during the month
- `SalariedEmployee` who is paid a monthly wage which is one 12th of their annual salary

(a) Draw an inheritance diagram for these classes.

[3]

(b) The design for the `Employee` class consists of:

- properties
  - `EmployeeName`
  - `EmployeeID`
  - `AmountPaidThisMonth`
- methods
  - `SetEmployeeName`
  - `SetEmployeeID`
  - `CalculatePay`

Write **program code** for the class definition of the superclass `Employee`.

Programming language .....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [5]

(c) **(i)** State the properties and/or methods required for the subclass `HourlyPaid`.

.....

.....

.....

..... [4]

(ii) State the properties and/or methods required for the subclass `SalariedEmployee`.

.....

.....

.....

..... [2]

(d) Name the feature of object-oriented program design that allows the method `CalculatePay` to be declared in the superclass `Employee`.

.....

..... [1]

5 Data is stored in the array `NameList[1:10]`. This data is to be sorted.

(a) (i) Complete the pseudocode algorithm for an insertion sort.

```
FOR ThisPointer ← 2 TO .....
    // use a temporary variable to store item which is to
    // be inserted into its correct location
    Temp ← NameList[ThisPointer]
    Pointer ← ThisPointer - 1

    WHILE (NameList[Pointer] > Temp) AND .....
        // move list item to next location
        NameList[.....] ← NameList[.....]
        Pointer ← .....
    ENDWHILE

    // insert value of Temp in correct location
    NameList[.....] ← .....
ENDFOR
```

[7]

(ii) A special case is when `NameList` is already in order. The algorithm in **part (a)(i)** is applied to this special case.

Explain how many iterations are carried out for each of the loops.

.....

.....

.....

.....

.....

..... [3]

(b) An alternative sort algorithm is a bubble sort:

```
FOR ThisPointer ← 1 TO 9
   FOR Pointer ← 1 TO 9
       IF NameList[Pointer] > NameList[Pointer + 1]
          THEN
             Temp ← NameList[Pointer]
             NameList[Pointer] ← NameList[Pointer + 1]
             NameList[Pointer + 1] ← Temp
       ENDIF
    ENDFOR
ENDFOR
```

(i) As in **part (a)(ii)**, a special case is when `NameList` is already in order. The algorithm in **part (b)** is applied to this special case.

Explain how many iterations are carried out for each of the loops.

.....

.....

.....

..... [2]

(ii) Rewrite the algorithm in **part (b)**, using **pseudocode**, to reduce the number of unnecessary comparisons. Use the same variable names where appropriate.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [5]

6 A queue Abstract Data Type (ADT) has these associated operations:

- create queue
- add item to queue
- remove item from queue

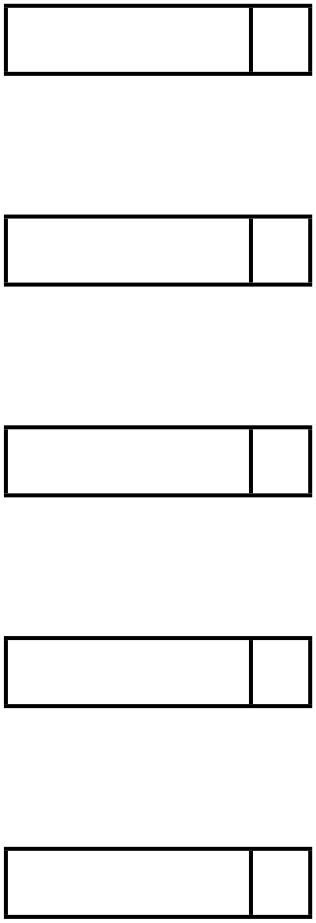
The queue ADT is to be implemented as a linked list of nodes.

Each node consists of data and a pointer to the next node.

(a) The following operations are carried out:

```
CreateQueue
AddName("Ali")
AddName("Jack")
AddName("Ben")
AddName("Ahmed")
RemoveName
AddName("Jatinder")
RemoveName
```

Add appropriate labels to the diagram to show the final state of the queue. Use the space on the left as a workspace. Show your final answer in the node shapes on the right:



[3]

(b) Using pseudocode, a record type, `Node`, is declared as follows:

```
TYPE Node
    DECLARE Name    : STRING
    DECLARE Pointer : INTEGER
ENDTYPE
```

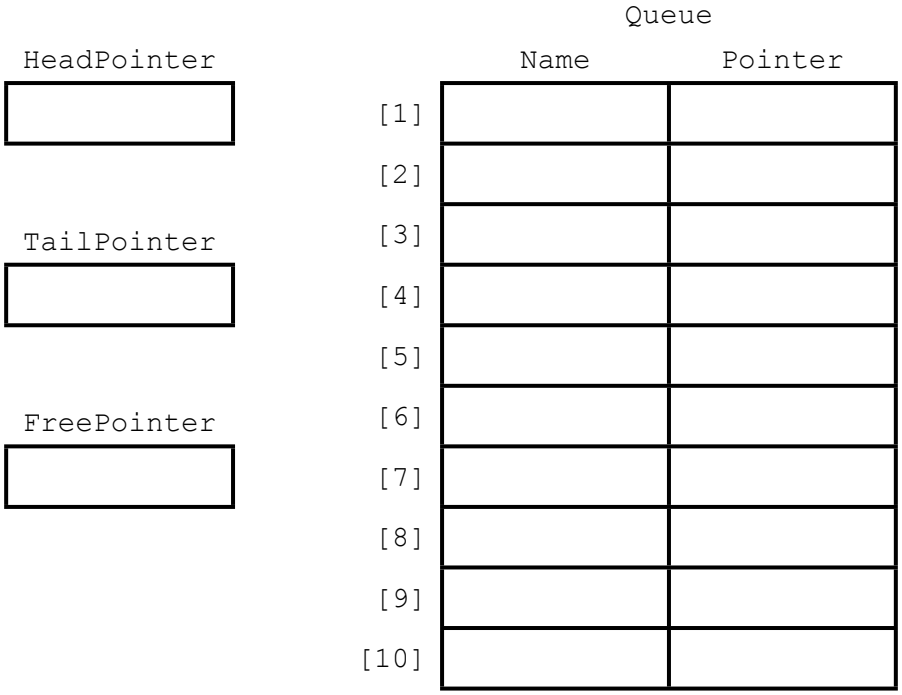
The statement

```
DECLARE Queue : ARRAY[1:10] OF Node
```

reserves space for 10 nodes in array `Queue`.

(i) The `CreateQueue` operation links all nodes and initialises the three pointers that need to be used: `HeadPointer`, `TailPointer` and `FreePointer`.

Complete the diagram to show the value of all pointers after `CreateQueue` has been executed.



[4]

(ii) The algorithm for adding a name to the queue is written, using pseudo[illegible] procedure with the header:

```
PROCEDURE AddName(NewName)
```

where `NewName` is the new name to be added to the queue.

The procedure uses the variables as shown in the identifier table.

Identifier	Data type	Description
`Queue`	`Array[1:10] OF Node`	Array to store node data
`NewName`	`STRING`	Name to be added
`FreePointer`	`INTEGER`	Pointer to next free node in array
`HeadPointer`	`INTEGER`	Pointer to first node in queue
`TailPointer`	`INTEGER`	Pointer to last node in queue
`CurrentPointer`	`INTEGER`	Pointer to current node

```
PROCEDURE AddName(BYVALUE NewName : STRING)
   // Report error if no free nodes remaining
   IF FreePointer = 0
      THEN
         Report Error
   ELSE
      // new name placed in node at head of free list
      CurrentPointer ← FreePointer
      Queue[CurrentPointer].Name ← NewName
      // adjust free pointer
      FreePointer ← Queue[CurrentPointer].Pointer
      // if first name in queue then adjust head pointer
      IF HeadPointer = 0
         THEN
            HeadPointer ← CurrentPointer
      ENDIF
      // current node is new end of queue
      Queue[CurrentPointer].Pointer ← 0
      TailPointer ← CurrentPointer
   ENDIF
ENDPROCEDURE
```

Complete the **pseudocode** for the procedure `RemoveName`. Use the variable names in the identifier table.

```
PROCEDURE RemoveName()
   // Report error if Queue is empty
.....
.....
.....
.....
   OUTPUT Queue[.....].Name
   // current node is head of queue
.....
   // update head pointer
.....
   // if only one element in queue then update tail pointer
.....
.....
.....
.....
   // link released node to free list
.....
.....
.....
ENDPROCEDURE
```

[6]