Cambridge International A Level

Cambridge International Examinations
Cambridge International Advanced Level

COMPUTER SCIENCE **9608/43**

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

PRE-RELEASE MATERIAL

This material should be given to candidates on receipt by the Centre.

1912082942

READ THESE INSTRUCTIONS FIRST

Candidates should use this material in preparation for the examination. Candidates should attempt the practical programming tasks using their chosen high-level, procedural programming language.

This document consists of **10** printed pages and **2** blank pages.

This material is intended to be read by teachers and candidates prior to the June 2015 examination for 9608 Paper 4.

Reminders
The syllabus states:

- there will be questions on the examination paper which do not relate to this pre-release material
- you must choose a high-level programming language from this list:
  - Visual Basic (Console Mode)
  - Python
  - Pascal / Delphi (Console Mode)

The practical skills covered in Paper 2 are a precursor to those required in Paper 4. It is therefore recommended that the high-level programming language chosen for this paper is the same as that for Paper 2. This allows for sufficient expertise to be acquired with the opportunity for extensive practice.

Questions on the examination paper may ask the candidate to write:

- structured English
- pseudocode
- program code

A program flowchart should be considered as an alternative to pseudocode for the documenting of an algorithm design.

Candidates should be confident with:

- the presentation of an algorithm using either a program flowchart or pseudocode
- the production of a program flowchart from given pseudocode (or the reverse)

TASK 1

A linked list Abstract Data Type (ADT) has these associated operations:

- create linked list
- add item to linked list
- remove item from linked list

Key focus: Linked lists

The linked list ADT consists of a linked list of nodes. Each node consists of data and a pointer to the next node.

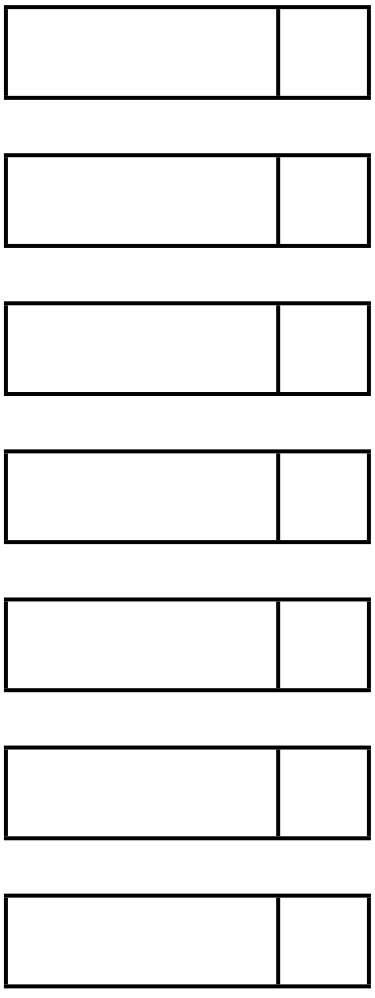
TASK 1.1
Consider the use of a linked list to store names in alphabetical order.

The following sequence of operations is carried out:

```
CreateLinkedList
AddName("Nabila")
AddName("Jack")
AddName("Kerrie")
AddName("Sara")
RemoveName("Kerrie")
AddName("Zac")
```

Key focus: Conceptual diagrams of linked lists

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



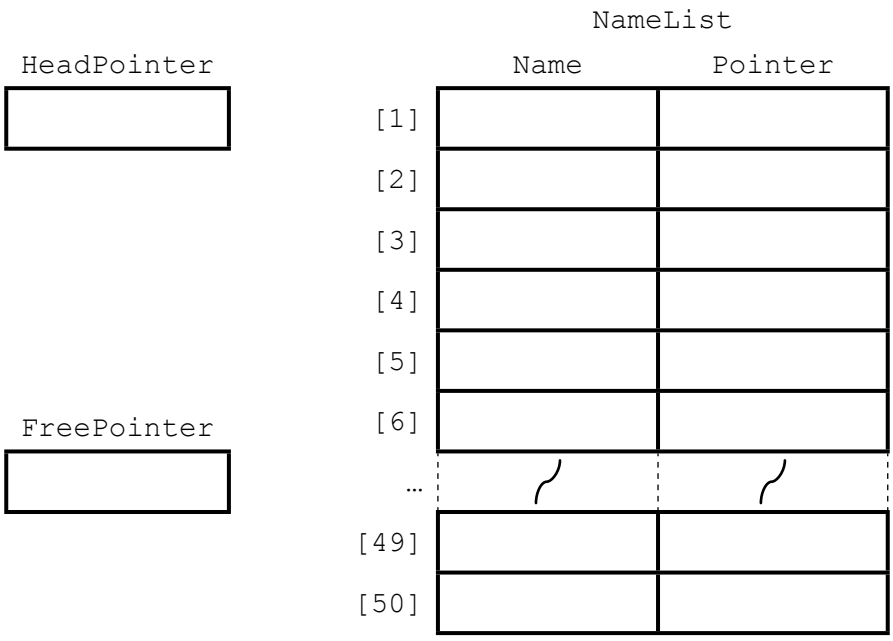
TASK 1.2

The linked list is to be implemented as an array of records, where each record represents a node.

The `CreateLinkedList` operation links all nodes to form the free list and initialises the `HeadPointer` and `FreePointer`.

Complete the diagram to show the values of all pointers after the `CreateLinkedList` operation has been carried out.

Key focus: Implementation of linked lists using an array of records



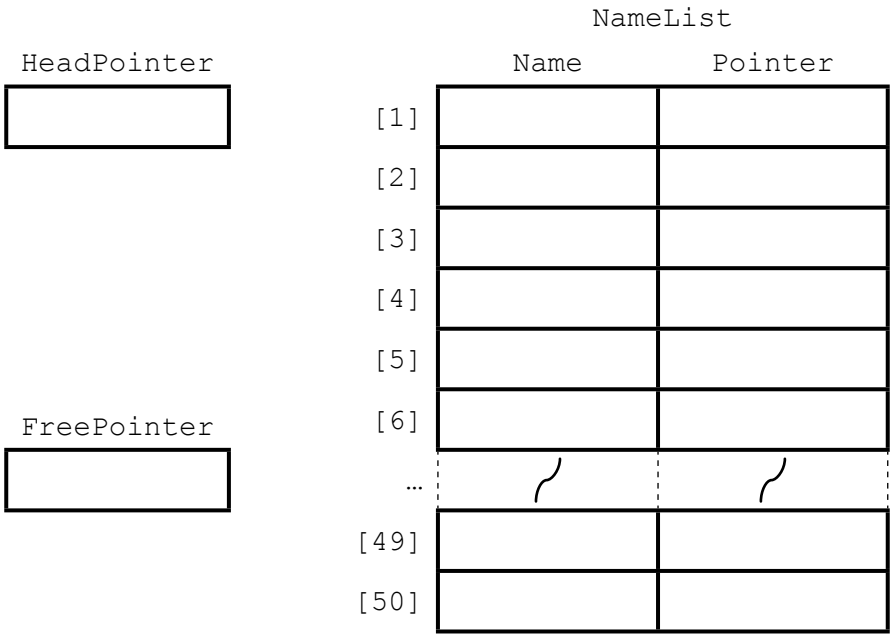
Write **pseudocode** for the `CreateLinkedList` operation.

Write **program code** from your pseudocode design.

TASK 1.3

Complete the diagram to show the values of all pointers after the following operations have been carried out:

```
AddName("Nabila")
AddName("Jack")
AddName("Kerrie")
AddName("Sara")
RemoveName("Kerrie")
AddName("Zac")
```



TASK 1.4
Complete the identifier table for the pseudocode given below.

Identifier	Data type	Description
		Array to store node data
		Name to be added
		Pointer to next free node in array
		Pointer to first node in list
		Pointer to current node
		Pointer to previous node
		Pointer to new node

```
01 PROCEDURE AddName(NewName)
02 // New name placed in node at head of free list
03   NameList[FreePointer].Name ← NewName
04   NewNodePointer ← FreePointer
05   FreePointer ← NameList[FreePointer].Pointer
06
07 // initialise current pointer to start of list
08   CurrentPointer ← HeadPointer
09
10 // check that it is not the special case of adding to empty list
11   IF HeadPointer > 0
12 // loop to locate position of new name
13 // saves current pointer and then updates current pointer
14     WHILE NameList[CurrentPointer].Name < NewName
15       PreviousPointer ← CurrentPointer
16       CurrentPointer ← NameList[CurrentPointer].Pointer
17     ENDWHILE
18   ENDIF
19
20 // check to see whether new name is first in linked list
21 // if first item then place item at head of list
22 // if not first item then adjust pointers to place it in correct
23 // position in list
24   IF CurrentPointer = HeadPointer
25     THEN
26       NameList[NewNodePointer].Pointer ← HeadPointer
27       HeadPointer ← NewNodePointer
28     ELSE
29       NameList[NewNodePointer].Pointer ← NameList[PreviousPointer].Pointer
30       NameList[PreviousPointer].Pointer ← NewNodePointer
31   ENDIF
32 ENDPROCEDURE
```

TASK 1.5
Write **program code** for the pseudocode given in Task 1.4.

TASK 1.6
The structured English algorithm for the operation to remove a name from the linked list is as follows:

```
If list is not empty
    Find the name to be removed
    If it is the first name in the linked list
        Adjust the head pointer
    If it is not the first name in the linked list
        Adjust pointers to exclude the name to be removed from the list
    Link released node into free list
```

TASK 1.6.1
Write the algorithm, as a procedure in **pseudocode**, from the structured English given above.

TASK 1.6.2
Write **program code** from your pseudocode design.

TASK 1.6.3
Test your program code for creating a linked list, adding and removing names, using the data given in Task 1.3.

Suggested extension task
Queues, stacks, binary trees and dictionaries can be implemented as linked lists of nodes.

Design **pseudocode** and write **program code** for these data structures.

[Turn over

TASK 2

A vehicle hire company has cars and trucks for hire.

The unique registration and the engine size (in litres, to the nearest 0.1 of a litre) are stored for all vehicles.

Data stored about cars also include the hire charge per day (in $) and the number of passengers allowed.

Data stored about trucks also include the hire charge per hour (in $) and the maximum payload (in kg).

Object-oriented software is to be written to process data about vehicles hired, including calculating the hire fee.

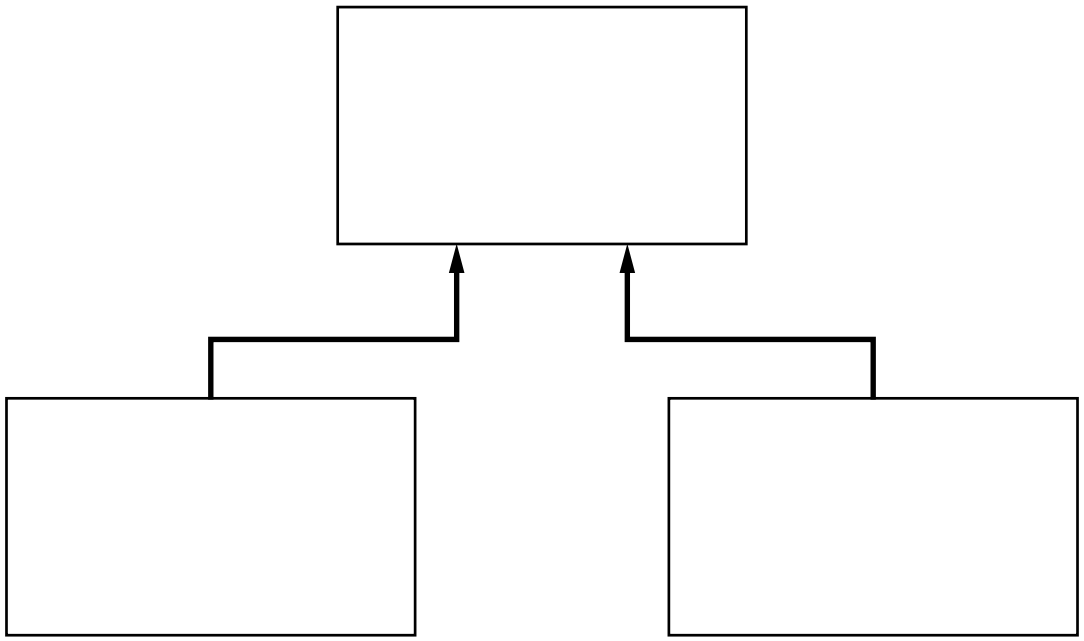
The superclass (also known as base class or parent class) `Vehicle` is designed.

Two subclasses (also known as derived classes or child classes) have been identified:

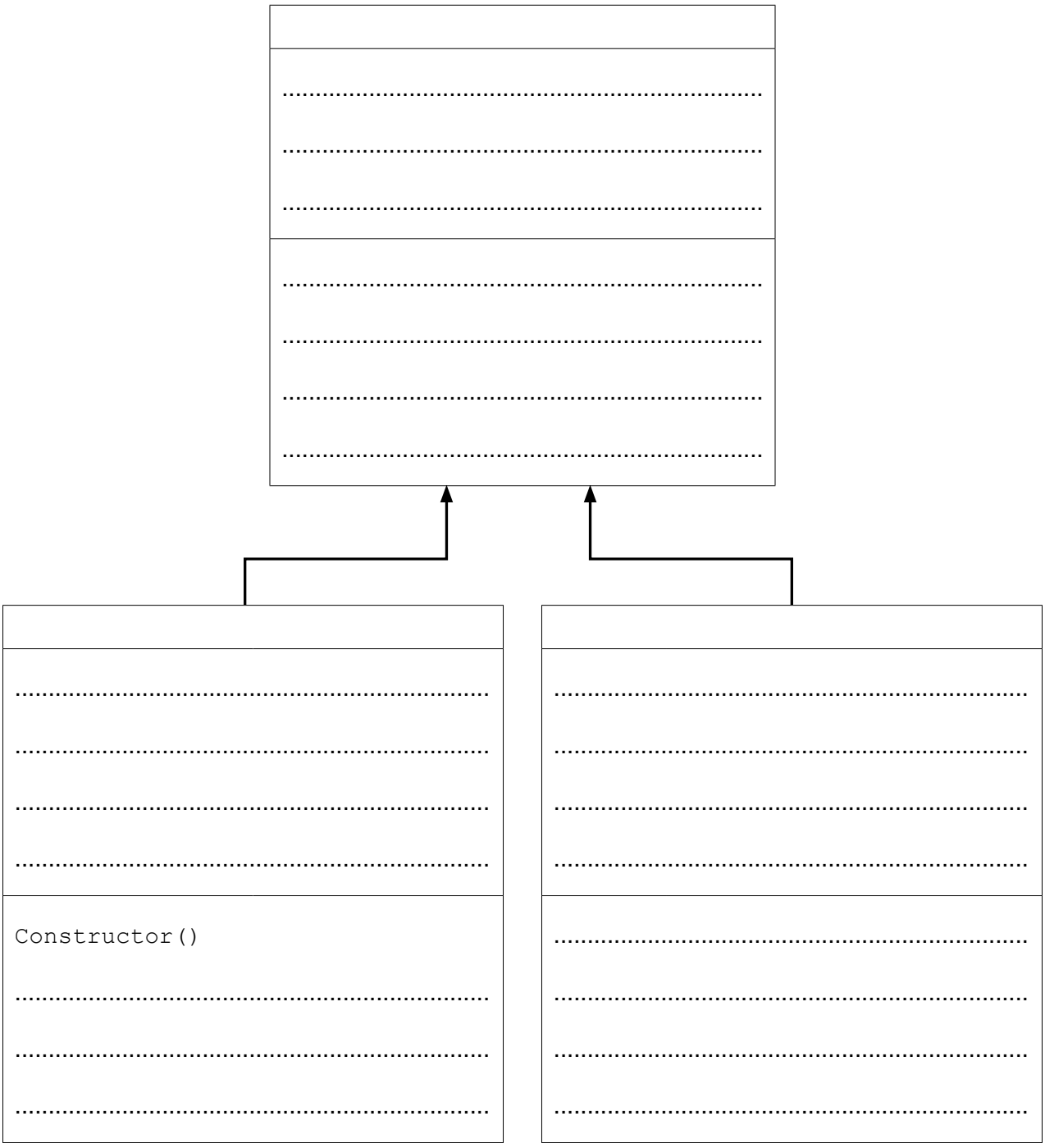
- `Car`
- `Truck`

Key focus: Object-oriented programming

TASK 2.1
Complete the **inheritance diagram**.



TASK 2.2
Complete the **class diagram** showing the appropriate properties and methods.



Note: a constructor is a method that creates a new instance of a class and initialises it.

TASK 2.3
Write **program code** for the class definitions. Make use of polymorphism and inheritance where appropriate.

TASK 2.4
Write **program code** to create a new instance of `Car`.

Suggested extension task
Write **program code** to display the properties of the object you created in Task 2.4.

TASK 3

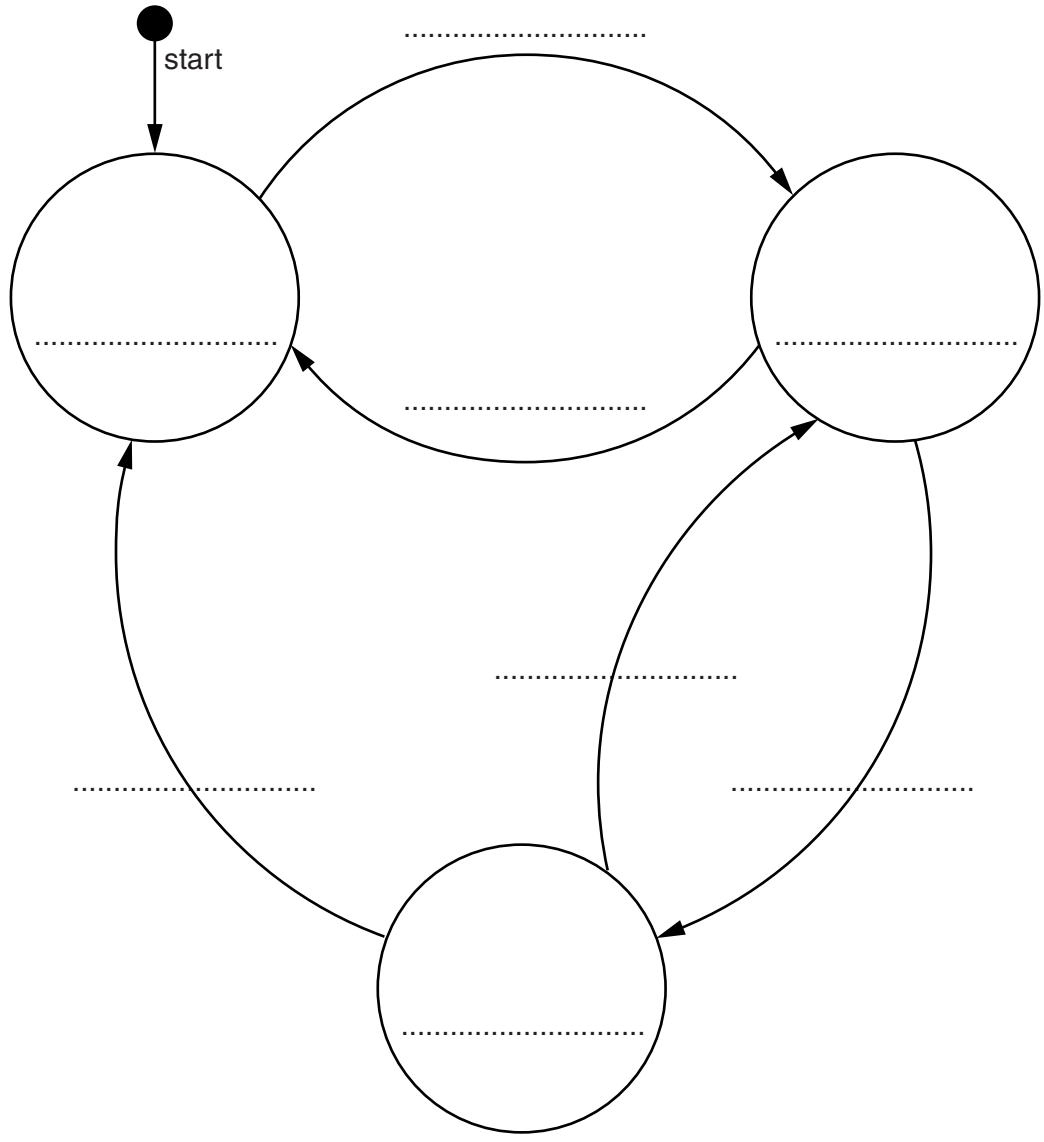
An intruder detection system is inactive when the power is switched off. The system is activated when the power is switched on. When the system senses an intruder the alarm bell rings. A reset button is pressed to turn the alarm bell off and return the system to the active state.

The transition from one state to another is as shown in the state transition table below.

Current state	Event	Next state
System inactive	Switch power on	System active
System active	Senses intruder	Alarm bell rings
System active	Switch power off	System inactive
Alarm bell rings	Press reset button	System active
Alarm bell rings	Switch power off	System inactive

Key focus: State transition diagrams

Complete the diagram.



BLANK PAGE

BLANK PAGE