Cambridge International AS & A Level

**Cambridge International Examinations**
Cambridge International Advanced Subsidiary and Advanced Level

CANDIDATE NAME

CENTRE NUMBER

CANDIDATE NUMBER

*2258633649*

**COMPUTER SCIENCE** **9608/41**

Paper 4 Further Problem-solving and Programming Skills **October/November 2018**

**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

**1** A declarative language is used to represent the following facts and rules about animals.

```
01 feature(dog, drinks_milk).

02 feature(dog, has_lungs).

03 feature(horse, has_lungs).

04 feature(tuna, lives_in_water).

05 feature(tuna, has_gills).

06 feature(crab, lives_in_water).

07 mammal(drinks_milk).

08 mammal(has_lungs).

09 fish(lives_in_water).

10 fish(has_gills).

11 is_a_mammal(X) IF (feature(X, Y) AND mammal(Y)) AND (feature(X, Z)
   AND mammal(Z)).
```

These clauses are explained in the following table.

| Clause | Explanation |
|---|---|
| 01 | A dog has the feature, drinks milk |
| 07 | A mammal drinks milk |
| 11 | X is a mammal, if:<br>• X has the feature Y and a mammal has a feature Y, **and**<br>• X has the feature Z and a mammal has the feature Z |

**(a)** More facts are to be included.

**(i)** A bird has wings, and a bird lays eggs.

Write the additional clauses to record these facts.

12 .....

13 .....

[2]

**(ii)** An eagle has all the features of a bird.

Write the additional clauses to record this fact.

14 .....

15 .....

[2]

**(b) (i)** Using the variable `B`, the goal

```
feature(B, drinks_milk)
```

returns

```
B = dog
```

Write the result returned by the goal

```
feature(B, lives_in_water)
```

`B =` ..... [2]

**(ii)** Write a goal, using the variable `C`, to find the feature(s) of tuna.

..... [2]

**(c)** An animal is a bird if it lays eggs **and** it has wings.

Complete the following rule.

`is_a_bird(X) IF` .....

..... [3]

**(d)** Declarative programming and object-oriented programming are two examples of programming paradigms.

**(i)** Define the term **programming paradigm**.

.....

..... [1]

**(ii)** Give **two** examples of programming paradigms, other than declarative and object-oriented programming.

1 .....

2 .....
[2]

**2** Kendra collects books. She is writing a program to store and analyse information about her books.

Her program stores information about each book as a record. The following table shows the information that will be stored about each book.

| Field name | Description |
|---|---|
| `Title` | The title of the book |
| `Author` | The first listed author of the book |
| `ISBN` | A 13-digit code that uniquely identifies the book, for example: `"0081107546738"` |
| `Fiction` | If the book is fiction (`TRUE`) or non-fiction (`FALSE`) |
| `LastRead` | The date when Kendra last read the book |

**(a)** Write **pseudocode** to declare an Abstract Data Type (ADT) named `Book`, to store the information in the table.

.....

.....

.....

.....

.....

.....

.....

.....

..... [4]

**(b)** The records are stored in a random access file.

The function, `Hash()`, takes as a parameter the ISBN and returns the hash value.

The disk address of the record in the hash table is calculated as: ISBN modulus 2000 plus 1.

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

Programming language .....

Program code .....

.....

.....

.....

.....

.....

.....

..... [4]

**(c)** The random access file, `MyBooks.dat`, stores the data about the books in the format:

```
<Title>
<Author>
<ISBN>
<Fiction>
<LastRead>
```

A procedure, `FindBook()`:

- prompts the user to input the ISBN of a book until the ISBN contains 13 numeric digits
- uses the function `Hash()` to calculate the disk address of the record
- reads the record for that book from `MyBooks.dat` into a variable of type `Book`
- outputs all the data about the book.

Use **pseudocode** to write the procedure `FindBook()`.

You can assume that the record exists at the disk address generated.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....[8]

**3** Joseph is taking a toy apart. Each time he removes an item from the toy, he writes the name of the item at the bottom of a paper list. When he rebuilds the toy, he puts the items back together working from the bottom of the list.

Joseph writes a computer program to create the list using a stack, `Parts`.

**(a)** Describe a stack structure.

.....

..... [1]

**(b)** The stack is represented as an array in the program, the first element in the array is `[0]`.

The current contents of the stack, `Parts`, and its pointer, `StackPointer` are shown.

| **StackPointer** | 5 |
|---|---|

| | **StackContents** |
|---|---|
| **0** | "Screw 1" |
| **1** | "Screw 2" |
| **2** | "Back case" |
| **3** | "Screw 3" |
| **4** | "Engine outer" |
| **5** | |
| **6** | |
| **7** | |

**(i)** Describe the purpose of the variable `StackPointer`.

.....

..... [1]

**(ii)** The procedure `POP()` removes an item from the stack. The procedure `PUSH(<identifier>)` adds an item to the stack.

The current contents of the stack, `Parts`, and its pointer, `StackPointer` are shown.

**StackPointer** 5

**StackContents**

| | |
|---|---|
| **0** | "Screw 1" |
| **1** | "Screw 2" |
| **2** | "Back case" |
| **3** | "Screw 3" |
| **4** | "Engine outer" |
| **5** | |
| **6** | |
| **7** | |

Use the table below to show the contents of the stack, `Parts`, and its pointer after the following code is run.

```
POP()
POP()
PUSH("Light 1")
PUSH("Light 2")
PUSH("Wheel 1")
POP()
POP()
```

**StackPointer**

**StackContents**

| | |
|---|---|
| **0** | |
| **1** | |
| **2** | |
| **3** | |
| **4** | |
| **5** | |
| **6** | |
| **7** | |

[2]

**(c)** A 1D array, `Parts`, is used to implement the stack. `Parts` is declared as:

```
DECLARE Parts : ARRAY[0 : 19] OF STRING
```

**(i)** The procedure `POP` outputs the last element that has been pushed onto the stack and replaces it with a `'*'`.

Complete the **pseudocode** for the procedure `POP`.

```
PROCEDURE POP

    IF ..... = .....

       THEN

          OUTPUT "The stack is empty"

       ELSE

          StackPointer ← .....

          OUTPUT .....

          Parts[StackPointer] ← .....

    ENDIF

ENDPROCEDURE
```

[5]

**(ii)** The procedure `PUSH()` puts the parameter onto the stack.

Complete the **pseudocode** for the procedure `PUSH()`.

```
PROCEDURE PUSH(BYVALUE Value : String)

    IF StackPointer > .....

       THEN

          OUTPUT "Stack full"

       ELSE

          ..... ← .....

          StackPointer ← .....

    ENDIF

ENDPROCEDURE
```

[4]

**4** The recursive algorithm for the `Calculate()` function is defined as follows:

```
01  FUNCTION Calculate(BYVALUE Number : INTEGER) RETURNS INTEGER
02      IF Number = 0
03          THEN
04              Calculate ← -10
05          ELSE
06              Calculate ← Number * Calculate(Number - 1)
07      ENDIF
08  ENDFUNCTION
```

**(a) (i)** State what is meant by a **recursive algorithm**.

.....

..... [1]

**(ii)** State the line number in `Calculate()` where the recursive call takes place.

..... [1]

**Question 4(b) begins on the next page.**

**(b)** The function is called with `Calculate(3)`.

Dry run the function **and** complete the trace table below. State the final value returned. Show your working.

```
01  FUNCTION Calculate(BYVALUE Number : INTEGER) RETURNS INTEGER
02      IF Number = 0
03          THEN
04              Calculate ← -10
05          ELSE
06              Calculate ← Number * Calculate(Number - 1)
07      ENDIF
08  ENDFUNCTION
```

Working .....

.....

.....

.....

Trace table:

| Call number | Function call | Number = 0 ? | Return value |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Final return value .....

[6]

**(c)** A recursive algorithm within a subroutine can be replaced with an iterative algorithm.

**(i)** Describe **one** problem that can occur when running a subroutine that has a recursive algorithm.

.....

.....

.....

..... [2]

**(ii)** Rewrite the `Calculate()` function in **pseudocode**, using an **iterative algorithm**.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [5]

**5** A game uses a set of cards. Each card has a number (between `0` and `9` inclusive) and a shape (`"square"`, `"triangle"` or `"circle"`).

The game is written using object-oriented programming.

The class, `Cards`, has the private properties:

- `Number`
- `Shape`

and the methods:

- `Constructor()`
- `GetNumber()`
- `GetShape()`

The purpose of each method in the class `Cards` is given in the following table.

| Method | Purpose |
|---|---|
| `Constructor()` | Takes a number and a shape as parameters<br>Checks that the number and the shape are valid and:<br>• either assigns the parameters to `Number` and `Shape`<br>• or reports an error. |
| `GetNumber()` | A public method that returns the number for that card. |
| `GetShape()` | A public method that returns the shape for that card. |

**(a)** Explain why the properties are private.

.....

.....

.....

.....

..... [2]

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

Programming language .....

Program code

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [5]

**(c)** Write **program code** for the `GetNumber()` method.

Programming language .....

Program code

.....

.....

.....

.....

..... [2]

**(d)** A card, `OneS`, has the value `1` for `Number` and the value `"square"` for `Shape`.

Write **program code** to instantiate an instance of `Cards` for `OneS`.

Programming language .....

Program code

.....

.....

..... [2]

**(e)** The game has a function, `Compare()` that takes two cards as parameters and compares them.

If the cards are identical, the function outputs `"SNAP"` and returns `-1`. If they are not identical, and the card numbers are different, it returns the `Number` of the card with the higher value or the `Number` for the cards if they are the same.

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

Programming language .....

Program code

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [6]