Cambridge Assessment International Education

# Cambridge International AS & A Level

CANDIDATE NAME

CENTRE NUMBER | | | | | |

CANDIDATE NUMBER | | | | |

*4026563148*

**COMPUTER SCIENCE** **9618/22**

Paper 2 Fundamental Problem-solving and Programming Skills **October/November 2023**

**2 hours**

You must answer on the question paper.

You will need: Insert (enclosed)

**INSTRUCTIONS**

- Answer **all** questions.
- Use a black or dark blue pen.
- Write your name, centre number and candidate number in the boxes at the top of the page.
- Write your answer to each question in the space provided.
- Do **not** use an erasable pen or correction fluid.
- Do **not** write on any bar codes.
- You may use an HB pencil for any diagrams, graphs or rough working.
- Calculators must **not** be used in this paper.

**INFORMATION**

- The total mark for this paper is 75.
- The number of marks for each question or part question is shown in brackets [ ].
- No marks will be awarded for using brand names of software packages or hardware.
- The insert contains all the resources referred to in the questions.

This document has **20** pages. Any blank pages are indicated.




**[Turn over**

Refer to the **insert** for the list of pseudocode functions and operators.

**1** A shop sells car accessories. A customer order is created if an item cannot be supplied from current stock. A program is being developed to create and manage the customer orders.

**(a)** The following identifier table shows some of the data that will be stored for each order.

Complete the identifier table by adding meaningful variable names and appropriate data types.

| Example value | Explanation | Variable name | Data type |
|---|---|---|---|
| `"Mr Khan"` | The name of the customer | | |
| `3` | The number of items in the order | | |
| `TRUE` | To indicate whether this is a new customer | | |
| `15.75` | The deposit paid by the customer | | |

[4]

**(b)** Other variables in the program have example values as shown:

| Variable | Example value |
|---|---|
| `Total` | `124.00` |
| `DepRate` | `2.00` |
| `Description` | `"AB12345:Cleaning Brush (small)"` |

Complete the table by evaluating each expression using the example values.

| Expression | Evaluates to |
|---|---|
| `(Total * DepRate) + 1.5` | |
| `RIGHT(Description, 7)` | |
| `(LENGTH(Description) - 8) > 16` | |
| `NUM_TO_STR(INT(DepRate * 10)) & '%'` | |

[4]

**(c)** The data that needs to be stored for each customer order in part **(a)** is not all of the same type.

Describe an effective way of storing this data for many customer orders while the program is running.

.....

.....

.....

.....

.....

..... [3]

**2** An algorithm will:

1. input a sequence of integer values, one at a time
2. ignore all values until the value 27 is input, then sum the remaining values in the sequence
3. stop summing values when the value 0 is input and then output the sum of the values.

**(a)** Draw a program flowchart to represent the algorithm.


[5]

**(b)** The solution to the algorithm includes iteration.

Give the name of a suitable loop structure that could be used.

Justify your answer.

Name .....

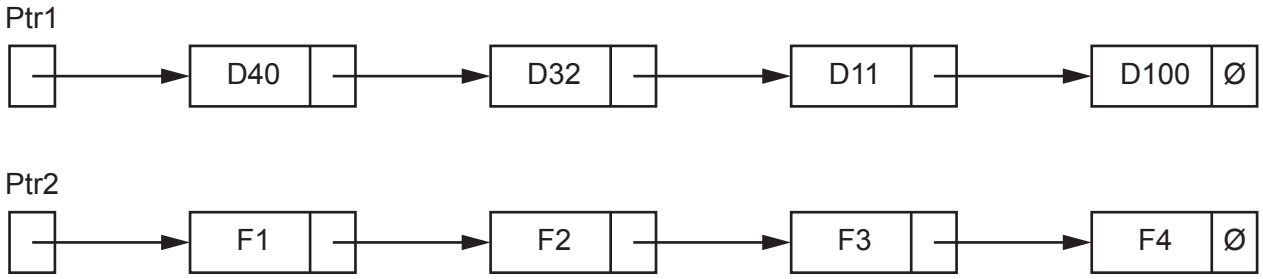
Justification .....

.....

[2]

**3** The diagram represents a linked list Abstract Data Type (ADT).

- Ptr1 is the start pointer. Ptr2 is the free list pointer.
- Labels D40, D32, D11 and D100 represent the data items of nodes in the list.
- Labels F1, F2, F3 and F4 represent the data items of nodes in the free list.
- The symbol Ø represents a null pointer.




**(a)** The linked list is implemented using two variables and two 1D arrays as shown.

The pointer variables and the elements of the Pointer array store the indices (index numbers) of elements in the Data array.

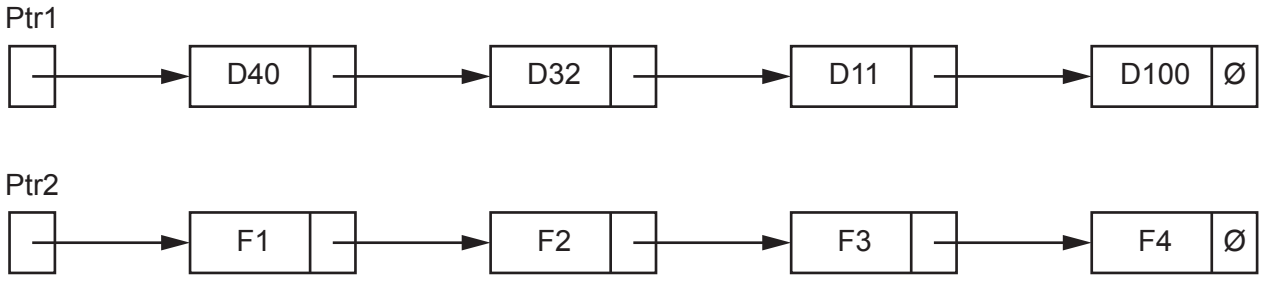
Complete the diagram to show how the linked list as shown above may be represented using the variables and arrays.

| Variable | Value |
|---|---|
| `Start_Pointer` | |
| `Free_List_Pointer` | 5 |

| Index | Data array | Pointer array |
|---|---|---|
| **1** | D32 | 2 |
| **2** | | 3 |
| **3** | | |
| **4** | D40 | |
| **5** | | |
| **6** | F2 | 7 |
| **7** | | |
| **8** | | |

[5]

**(b)** The original linked list is to be modified. A new node D6 is inserted between nodes D32 and D11.




The algorithm required is expressed in four steps as shown.

Complete the steps.

1. Assign the data item ..... to ..... .
2. Set the ..... of this node to point to ..... .
3. Set Ptr2 to point to ..... .
4. Set pointer of ..... to point to ..... .

[4]

**4** A procedure `Count()` will:

1. input a value (all values will be positive integers)
2. count the number of odd values and count the number of even values
3. repeat from step 1 until the value input is 99
4. output the two count values, with a suitable message.

The value 99 must not be counted.

**(a)** Write pseudocode for the procedure `Count()`.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [6]

**(b)** The procedure `Count()` is to be tested.

Typical test data would consist of odd and even values, for example:

23, 5, 64, 100, 2002, 1, 8, 900, 99

The purpose of this test would be to test a typical mix of even and odd values and check the totals.

Give **three** test data sequences that may be used to test **different** aspects of the procedure.

Do **not** include invalid data.

**Sequence 1:**

Test data .....

Purpose of test. .....

.....

**Sequence 2:**

Test data .....

Purpose of test. .....

.....

**Sequence 3:**

Test data .....

Purpose of test. .....

.....

[3]

**5** A global 1D array of integers contains four elements, which are assigned values as shown:

```
Mix[1] ← 1
Mix[2] ← 3
Mix[3] ← 4
Mix[4] ← 2
```

A procedure `Process()` manipulates the values in the array.

The procedure is written in pseudocode:

```
PROCEDURE Process(Start : INTEGER)
   DECLARE Value, Index, Count : INTEGER

   Index ← Start
   Count ← 0

   REPEAT
      Value ← Mix[Index]
      Mix[Index] ← Mix[Index] - 1
      Index ← Value
      Count ← Count + 1
   UNTIL Count = 5

   Mix[4] ← Count * Index

ENDPROCEDURE
```

Complete the trace table on the opposite page by dry running the procedure when it is called as follows:

```
CALL Process(3)
```

| Index | Value | Count | Mix[1] | Mix[2] | Mix[3] | Mix[4] |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

[6]

**6** **(a)** A procedure `CreateFiles()` will take two parameters:

- a string representing a file name
- an integer representing the number of files to be created.

The procedure will create the number of text files specified.

Each file is given a different name. Each file name is formed by concatenating the file name with a suffix based on the file number. The suffix is always three characters.

For example, the call `CreateFiles("TestData", 3)` would result in the creation of the three files, TestData.001, TestData.002 and TestData.003.

Each file will contain a single line. For example, file TestData.002 would contain the string:

```
This is File TestData.002
```

Write pseudocode for `CreateFiles()`.

Assume both parameters are valid and that the integer value is between 1 and 999, inclusive.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [6]

**(b)** A module `CheckFiles()` will count the number of files produced by `CreateFiles()` in part **(a)**.

`CheckFiles()` will take a string representing a file name and return the number of files found.

**(i)** Identify the type of module that should be used for `CheckFiles()`.

..... [1]

**(ii)** Write the module header for `CheckFiles()`.

.....

..... [1]

**(iii)** State the file mode that should be used in `CheckFiles()`.

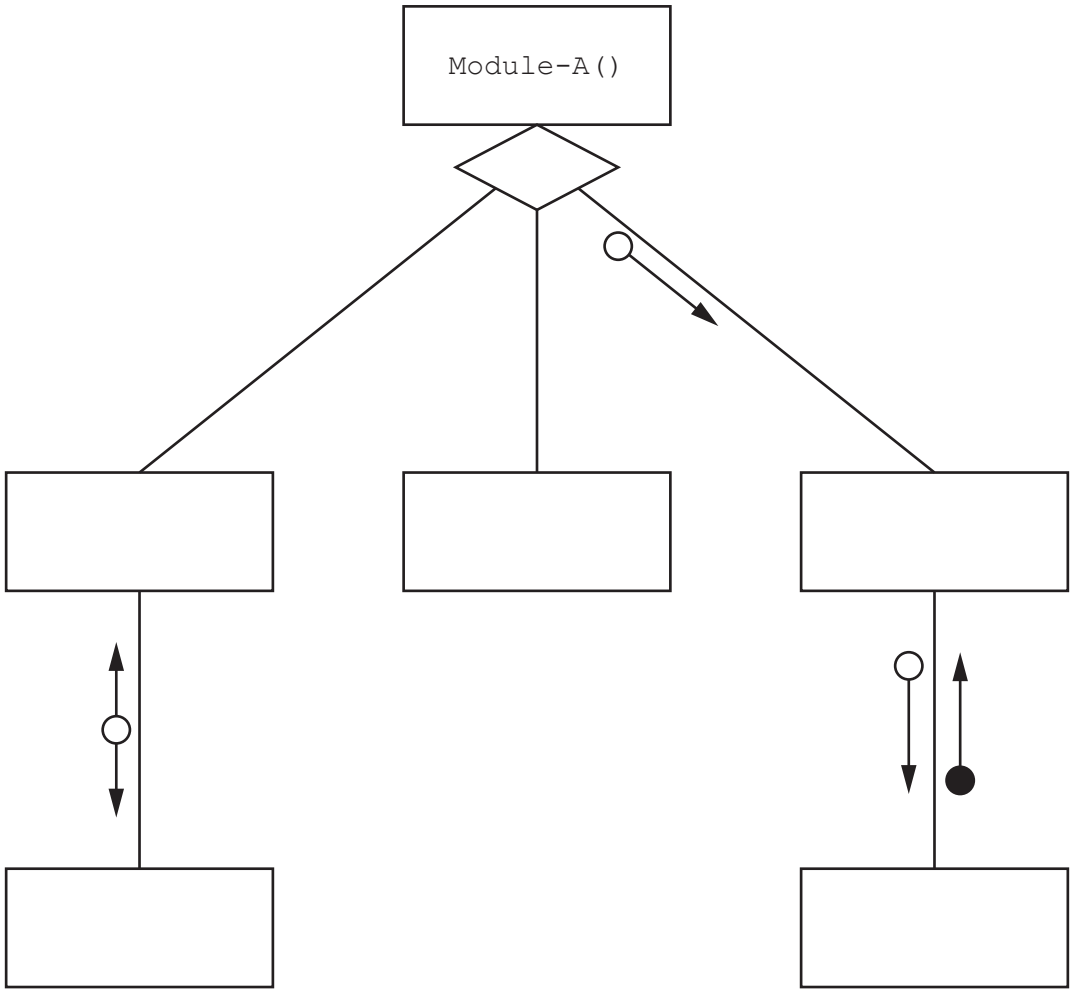
..... [1]

**7** A program contains six modules:

| Pseudocode module header |
|---|
| `PROCEDURE Module-A()` |
| `PROCEDURE Module-X(T1 : INTEGER, S2 : REAL)` |
| `PROCEDURE Reset(BYREF Code : INTEGER)` |
| `FUNCTION Restore(OldCode : INTEGER) RETURNS BOOLEAN` |
| `FUNCTION Module-Y(RA : INTEGER, RB : BOOLEAN) RETURNS BOOLEAN` |
| `FUNCTION Module-Z(SA : INTEGER) RETURNS INTEGER` |

`Module-X()` calls `Reset()`
`Module-Y()` calls `Restore()`

**(a)** Complete the structure chart for these modules.




[4]

**(b)** Explain the meaning of the diamond symbol as used in the diagram in part **(a)**.

.....

..... [2]

**BLANK PAGE**

**8** A class of students are developing a program to send data between computers. Many computers are connected together to form a wired network. Serial ports are used to connect one computer to another.

Each computer:

- is assigned a unique three-digit ID
- has three ports, each identified by an integer value
- is connected to between one and three other computers.

Data is sent as individual message strings.
Each string contains the destination ID (the ID of the computer that is to receive the message) followed by the data:

```
<DestinationID><Data>
```

Messages may pass through several computers on the way to their destination.
When a message arrives at a computer, that is **not** the destination, the program needs to forward it on to another computer using one of its serial ports.

The port to use is obtained from information that is stored in an array `RouteTable`.

`RouteTable` is a global 2D array of integers. It is declared in pseudocode as follows:

```
DECLARE RouteTable : ARRAY[1:6,1:3] OF INTEGER
```

The values in the first two columns of `RouteTable` define a range of ID values.
Column 3 gives the corresponding port number to use when forwarding the message to a computer with an ID within this range.

For example, the contents of `RouteTable` could be:

| | Column 1 | Column 2 | Column 3 |
|---|---|---|---|
| **Row 1** | 100 | 199 | 1 |
| **Row 2** | 200 | 259 | 2 |
| **Row 3** | −1 | <undefined> | <undefined> |
| **Row 4** | 260 | 399 | 2 |
| **Row 5** | 400 | 599 | 3 |
| **Row 6** | 600 | 999 | 1 |

In this example, a message that arrives with a `DestinationID` of "283" will be forwarded using port 2.

Row 3 in the example shows an unused row. These may occur anywhere. Unused rows have the column 1 element set to −1. The value of unused elements in the other two columns is undefined.

The programmer has defined the first program module as follows:

| Module | Description |
|---|---|
| `GetPort()` | • takes a `DestinationID` as a parameter of type string<br>• searches for the range corresponding to the `DestinationID` in the array<br>• returns the port number, or returns −1 if no corresponding range is found |

**(a)** Write pseudocode for module `GetPort()`.

Assume `DestinationID` contains a valid three-digit string.

..... [7]

**(b)** Copies of the same program will run on each computer. The program contains a global variable `MyID` of type string, which contains the unique ID of the computer in which the program is running.

When messages are received, they are placed on one of two stacks. Stack 1 is used for messages that have reached their destination and stack 2 is used for messages that will be forwarded on to another computer.

Additional modules are defined:

| Module | Description |
|---|---|
| `StackMsg()`<br>(already written) | • takes two parameters:<br>  ○ a string representing a message<br>  ○ an integer representing the stack number<br>• adds the message to the required stack<br>• returns `TRUE` if the message is added to the required stack, otherwise returns `FALSE` |
| `ProcessMsg()` | • takes a message as a parameter of type string<br>• ignores any message with a zero-length data field<br>• extract the `DestinationID` from the message<br>• checks whether the `DestinationID` is this computer or whether the message is to be forwarded<br>• uses `StackMsg()` to add the message to the appropriate stack<br>• outputs an error if the message could not be added to the stack |

Write pseudocode for module `ProcessMsg()`.

Module `StackMsg()` must be used.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [7]

**(c)** The program contains a module `GetFile()` which receives text files sent from another computer.

Lines from the file are sent one at a time. Each message contains one line and `ProcessMsg()` from part **(b)** adds each message as it is received onto stack 1.

Module `GetFile()` removes messages from stack 1 and writes the data to a text file.

There is a problem. Under certain circumstances, the received file does not appear as expected.

Assume that while a file is being received `ProcessMsg()` receives only messages containing lines from the file.

**(i)** Describe the circumstances and explain the problem.

Circumstances .....

.....

Explanation .....

.....

.....

.....

[3]

**(ii)** Suggest a more appropriate Abstract Data Type that could be used to store the messages that would not have the same problem.

..... [1]