
COMPUTER SCIENCE

9618/04

Paper 4 Practical

MOCK PAPER

2 hours 30 minutes

You will need: Candidate source files **evidence.doc**

INSTRUCTIONS

- Carry out every instruction in each task.
- Save your work using the file names given in the task as and when instructed.
- You must **not** have access to either the internet or any email system during this examination.
- You must save your work in the evidence document as stated in the tasks. If your work is not saved in the evidence document, you will **not** receive marks for that task.
- You must use a high-level programming language from this list:
 - Python (console mode)
 - Visual Basic (console mode)
- A mark of **zero** will be awarded if a programming language other than those listed here is used.
- Use white background on IDE, any screenshots having dark background will not be awarded marks.

INFORMATION

- The total mark for this paper is 75.
- The number of marks for each question or part question is shown in brackets [].

Open the document **evidence.doc**

Make sure that your name and related information will appear on every page of this document. This document must contain your answers to each question.

Save this evidence document in your work area as:

evidence_ followed by your name_ number, for example: evidence_zak_12345.

A class declaration can be used to declare a record.

If the programming language used does not support arrays, a list can be used instead.

Candidates will use a source file to answer Question 2.

The provided source text file is: `shipment_data.txt`

1. The factorial of a non-negative integer n , denoted $n!$, is the product of all positive integers less than or equal to n . For example, $5! = 5 * 4 * 3 * 2 * 1 = 120$. The factorial of 0 is defined as 1.

Below is an iterative pseudocode algorithm for calculating factorials:

```
FUNCTION IterativeFactorial(Number : INTEGER) RETURNS INTEGER
  IF Number = 0 THEN
    RETURN 1
  ELSE
    Result ← 1
    FOR Count ← 1 TO Number
      Result ← Result * Count
    NEXT Count
    RETURN Result
  ENDIF
ENDFUNCTION
```

(a)

(i) Write code for the function `IterativeFactorial()`.

[5 marks]

Save your program code in the given folder and name it as **question1**

Copy and paste the program code into **part 1(a)(i)** in the evidence document

(ii) Write code to call the function `IterativeFactorial()` with the parameter 5,0,3 and output the return values.

[3 marks]

Save your program code.

Copy and paste the screenshot(s) showing result into **part 1(a) (ii)** in the evidence doc.

(b)

(i) Rewrite the function `IterativeFactorial()` as a recursive function with the identifier `RecursiveFactorial()`.

[5 marks]

Save your program code.

Copy and paste the program code into **part 1(b)(i)** in the evidence document

(ii) Write code to call the function `RecursiveFactorial()` with different parameters 5,0 and 3 and output the return values.

[3 marks]

Save your program code.

Copy and paste the screenshot(s) showing result into **part 1(b) (ii)** in the evidence doc.

2.

A warehouse uses a stack-based system, implemented with a 1D array, `ProductStack`, for managing incoming product shipments.

Each product **record**, `ProductUDT`, includes:

1. a unique product ID (as an integer), and
2. a product name (as a string).

The `ProductUDT` ADT based records are managed through stack operations and random binary file interactions for efficient storage and access. A resource text file named `shipment_data.txt` is provided in the given folder, containing product records with unique IDs to ensure uniform outputs for evaluation and to facilitate direct access in the random file.

(a) Write program code to declare UDT, ADT and global variables as below:

`ProductUDT` — a user defined record data type as described above for product record.

`ProductStack` — a global array with space to store 50 product records.

`TopPointer` — a global variable indicating the last occupied position in the stack, initialized to -1 (indicating the stack is empty). **[4 marks]**

Save your program code in the given folder and name it as **question2**

Copy and paste the program code into **part 2(a)** in the evidence document

(b)

Using an external generic text editor like Notepad, open the resource text file `shipment_data.txt` in the provided folder and observe record structure used to save each record.

- (i)** Write program code for the procedure `push()`, that receives a parameter of type `ProductUDT` and add product data to the stack, `ProductStack`, ensuring no overflow.

Overflow Check: Before adding a new product, check if `TopPointer` equals 49. If so, print "Stack overflow! Unable to add more products." and do not add the product to the stack. **[3 marks]**

Save your program code.

Copy and paste the program code into **part 2(b)(i)** in the evidence document

- (ii)** Write program code for the procedure `EncodeShipment()` to read records from source file and store them in stack:

1. read all records one by one from the text file `shipment_data.txt`, and
2. store in a variable `thisRecord` of type `ProductUDT`, and
3. push `thisRecord` to the stack `ProductStack` by calling procedure `push()` and passing as argument.

Text File Format: Each line in `shipment_data.txt` is formatted as
"ProductID<space>ProductName". **[4 marks]**

Save your program code.

Copy and paste the program code into **part 2(b)(ii)** in the evidence document

- (iii) Write program code for the function `pop()` to retrieve and remove the top element from stack `ProductStack` and return it, managing underflow.

Underflow Check: If `TopPointer` is `-1` when `pop()` is called, print "Stack underflow! No products to remove." and return a null or equivalent in your programming language to indicate that no product was popped. [3 marks]

Save your program code.

Copy and paste the program code into **part 2(b)(iii)** in the evidence document

- (iv) Write program code for the procedure `DecodeShipment()` to:
1. pop elements from stack `ProductStack`, and
 2. save them in a random (binary) file `shipment_data.dat` in the provided folder as a fixed sized length record.




Random File Structure: In `shipment_data.dat`, each product record will be 50 bytes, where 4 bytes are saved for `ProductID` (integer) and 46 bytes for `ProductName` (fixed-length string). [4 marks]

Save your program code.

Copy and paste the program code into **part 2(b)(iv)** in the evidence document

(c) Direct Product Lookup in Binary File

Write a function `RetrieveProductName(product_id)`, that:

-  directly accesses `shipment_data.dat` that was made in **b(iv)** above,
-  reaches for a product record using `product_id`, and
-  returns the associated product name.

Direct Access Implementation: Function `RetrieveProductName(product_id)` uses fixed-size record length to direct access the random file, calculate the byte offset using:

$$\text{offset} = (\text{product_id} - 1) * 50.$$




Use this offset to seek to the correct position in the binary file before reading the product record. If the `product_id` is not found, returns "Product not found". [6 marks]

Save your program code.

Copy and paste the program code into **part 2(c)** in the evidence document

(d) Integration and Testing

(i) Write a main program part that integrates the subroutines by calling:

-  EncodeShipment () to process and encode product records from shipment_data.txt,
-  DecodeShipment () to read the Stack in LIFO order and save as a fixed sized record in shipment_data.dat.
-  RetrieveProductName () to direct access binary random file for the product id 7 from file shipment_data.dat.

[4 marks]

Save your program code.

Copy and paste the program code into **part 2(d)(i)** in the evidence document

(ii) Test run your program and use product ID 7 for lookup when calling RetrieveProductName (). Take a screenshot of the output showing the decoded product records and the result of the direct product lookup.

[2 mark]

Save your program code.

Copy and paste the screenshot(s) showing result into **part 2(d) (ii)** in the evidence doc.

3. In a modern, automated home, smart devices like lights and thermostats adjust settings throughout the day to optimize comfort and energy efficiency. Your task is to simulate this smart home environment using object-oriented programming (OOP).

(a) Class Design and Implementation

(i) SmartDevice Class: Define a class SmartDevice with following attributes. Include a constructor to initialize these attributes.

name: STRING	stores the name of the device like "Living Room Light", "Main Thermostat"
type: STRING	stores the type of the device like light, thermostat
serial_number: STRING	stores the serial number of the device, like LR001, MT001, etc.
setup_year: STRING	stores the year the device was set up, like 2024, 2025, etc.
is_on: BOOLEAN	stores the power status of the device, like TRUE or FALSE for on or off respectively

[2 Marks]

Save your program code in the given folder and name it as **question3**

Copy and paste the program code into **part 3(a)(i)** in the evidence document

- (ii) Accessors and Mutators: Implement public getter and setter methods for all attributes, including a method `is_device_on()` to check the device's power status. [3 Marks]

Save your program code.

Copy and paste the program code into **part 3(a)(ii)** in the evidence document

- (iii) Age Calculation Method: Add a method `get_age_in_years()` to calculate and return the device's age. [2 Marks]

Save your program code.

Copy and paste the program code into **part 3(a)(iii)** in the evidence document

- (iv) Power Control Methods: Include methods `turn_on()` and `turn_off()` to control the device's power status. [3 Marks]

Save your program code.

Copy and paste the program code into **part 3(a)(iv)** in the evidence document

(b) Inheritance

- (i) SmartLight Class: Create a subclass `SmartLight` inheriting from `SmartDevice`, adding an attribute `brightness` and a method `set_brightness(brightness)` to adjust it. [2 Marks]

Save your program code.

Copy and paste the program code into **part 3(b)(i)** in the evidence document

- (ii) SmartThermostat Class: Develop a subclass `SmartThermostat` with an attribute `temperature` and methods `set_temperature(temperature)` and `suggest_temperature(season)` to suggest and set temperatures based on the season as:

- If the season is winter, it sets the temperature to 20 degrees.
- If the season is summer, it changes the temperature to 25 degrees.
- For any other season that isn't explicitly winter or summer, it sets the temperature to 22 degrees.

[2 Marks]

Save your program code.

Copy and paste the program code into **part 3(b)(ii)** in the evidence document

(c) System Simulation

(i) SmartLight Object:

- Create a SmartLight object with the following initial values: name as "Living Room Light", type as "light", serial_number as "LR001", setup_year as the current year, and is_on as FALSE. The brightness should be initially set to 0.
- Turn on the SmartLight object and set its brightness to 75 to simulate a bright environment.
- Change the brightness to 50 to simulate a dimmer environment.
- Finally, turn off the SmartLight object.

[4 marks]

Save your program code.

Copy and paste the program code into **part 3(c)(i)** in the evidence document

(ii) SmartThermostat Object:

- Create a SmartThermostat object named "Main Thermostat" with type as "thermostat", serial_number as "MT001", setup_year as the current year, is_on as True, and initially set the temperature to 68°F.
- Adjust the temperature to 70°F to simulate warming the room.
- Use the suggest_temperature() method with "winter" as the parameter to suggest and adjust the temperature to the suggested value.

[4 marks]

Save your program code.

Copy and paste the program code into **part 3(c)(ii)** in the evidence document

(iii) Smart Home Integration:

- Initialize:
 - a SmartLight object for the living room ("Living Room Light", "light", "LR002", current year, False) with an initial brightness of 0, and
 - a SmartThermostat object for the home ("Home Thermostat", "thermostat", "HT001", current year, True) with an initial temperature of 68°F.
- Simulate the day cycle:
 - Zak Morning (7:00 AM): Turn on the living room light to 60% brightness and adjust the thermostat to 70°F to simulate a warm morning.
 - Zak Day (9:00 AM to 5:00 PM): Turn off the living room light assuming natural light suffices and adjust the thermostat to a comfortable day setting of 68°F.
 - Zak Evening (6:00 PM): Turn on the living room light to 70% brightness for adequate lighting and adjust the thermostat to 72°F for evening comfort.
 - Zak Night (10:00 PM): Turn off the living room light and lower the thermostat to 65°F for energy efficiency during sleep.
- For each time of day, output the status of each device, including whether the light is on and its current brightness, as well as the current thermostat setting.

[8 marks]

Save your program code.

Copy and paste the program code into **part 3(c)(iii)** in the evidence document

Instructions:

- Ensure your program is well-commented, explaining key operations and choices.
- Save your work frequently, using given file name.
- Marks will be awarded for correctness, adherence to OOP principles, and clarity of your code.