OXFORD AQA

INTERNATIONAL QUALIFICATIONS

INTERNATIONAL GCSE
COMPUTER SCIENCE

Paper 1 Programming

Mock2 May 2025 Time allowed: 2 hours

Materials

For this paper you must have access to:

- a computer
- a printer
- appropriate software
- electronic versions of the **Preliminary Material** and the **Skeleton Program**
- a hard copy of the **Preliminary Material** and the **Skeleton Program**.

Information

- The marks for questions are shown in brackets.
- The maximum mark for this paper is 80.
- No extra time is allowed for printing and collating.
- The question paper is divided into **three** sections.
- You may use a bilingual dictionary.
- You must **not** use an English dictionary.
- You are **not** allowed to use a calculator.

Instructions

- Type the information required on the front of your **Electronic Answer Document**.
- Answer **all** questions.
- Enter your answers into the **Electronic Answer Document**.
- Before the start of the examination make sure your **centre number**, **candidate name** and **candidate number** are shown clearly in the footer of every page of your **Electronic Answer Document** (not the front cover).
- Include the evidence required for your answers to **Sections B** and **C** in your **Electronic Answer Document**.
- You **must save** your **Electronic Answer Document** at regular intervals.
- The questions in **Sections B** and **C** require you to make changes to the **Skeleton Program**.
- All of the programming questions in **Sections B** and **C** can be answered independently of each other. If you cannot answer one of the questions you can still attempt to solve later questions.
- You are advised to keep a backup copy of the original **Skeleton Program** so that you can go back to it if you accidentally make changes to the program which means it can no longer be compiled/executed while answering the questions in **Sections B** and **C**.

Secure all your printed Electronic Answer Document pages together and hand them to the invigilator.

Section A (Non-programming questions)

You are advised to spend no more than **30 minutes** on this section.

Type your answers to **Section A** in your **Electronic Answer Document**.

You **must save** your **Electronic Answer Document** at regular intervals.

The questions in this section are about programming and how the **Skeleton Program** works. Do **not** make any changes to the **Skeleton Program** when answering these questions.

0 1 . 1 State a name of a global variable in the **Skeleton Program** and explain its purpose.

[2 mark]

0 1 . 2 Describe how the subroutine `DrawBoard` uses iteration to display the board.

[2 mark]

0 1 . 3 What is the purpose of the `errorCode` variable in the Skeleton Program?

[1 marks]

0 1 . 4 In the `CreateBoard` subroutine, what is the purpose of the line

```
if column == 0 or column == WIDTH - 1 or row == 0 or row
== HEIGHT - 1:
```

[2 marks]

0 1 . 5 How does the `ProcessInput` subroutine handle user commands?

Explain the role of the `parts` list in this process.

[2 mark]

0 1 . 6 The **Skeleton Program** uses the data structure

```
directions = [UP, RIGHT, DOWN, LEFT]
```

Identify the type of this data structure, and list the type of each element it contains. **[2 marks]**

Question 1 continues on the next page

0 1 . 7 The subroutine **CheckForward** returns `True` or `False`.

State the advantage of having **CheckForward** return a Boolean rather than directly moving the pointer. **[2 mark]**

0 1 . 8 State one advantage of using **definite iteration** (for loops) in `CreateBoard` rather than **indefinite iteration** (while loops).

[1 marks]

0 1 . 9 Describe the difference between a **global variable** and a **local variable**.
Name one local variable in `ProcessInput(command)`.

[2 marks]

0 1 . 10 Explain how the `Turn` subroutine determines the new direction when turning left or right.

[2 marks]

0 2 . 1 The **Skeleton Program** `Turn` subroutine is to be modified to support eight directions instead of four. The new directions list is defined as:

```
directions = [UP, UP_RIGHT, RIGHT, RIGHT_DOWN, DOWN, LEFT_DOWN, LEFT, UP_LEFT]
```

Provide the missing code for the following section of the `Turn` subroutine: **[3 marks]**

```
if rotation == TURN_UP_RIGHT:
    newDirection =
elif rotation == TURN_RIGHT:
    newDirection =
elif rotation == TURN_RIGHT_DOWN:
    newDirection =
```

0 2 . 2

What would happen if the `Turn` subroutine used `directionIndex` instead of `(directionIndex + 1) % 8` for a right turn in the new eight-direction list directions = [UP, UP_RIGHT, RIGHT, RIGHT_DOWN, DOWN, LEFT_DOWN, LEFT, UP_LEFT]?

[1 mark]

Turn over for the next section

Section B (Short programming questions)

You are advised to spend no more than **45 minutes** on this section.

0 3 The **Skeleton program** is to display basic game instructions when started.

Change the `Main` subroutine so that:
- After the title, display `Navigate the pointer to reach @`
- On the next line: `Use FD to move, RT/LT to turn`
- Put a **blank** line after the instructions

Test your changes by running the **Skeleton Program**.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

0 3 . 1 **All** the PROGRAM SOURCE CODE for the amended subroutine `main`.

[3 marks]

0 3 . 2 SCREEN CAPTURE(S) showing the results of the requested test.

[1 mark]

0 4 The **Skeleton Program** is to be improved to include trap cells that reset the pointer to its starting position. Modify the program to:

- Define a new constant TRAP = '$' to represent trap cells.
- In option 3 (random board), add a 20% chance for any non-boundary, non-goal cell to become a trap (TRAP).
- When the pointer lands on a TRAP cell during a move (via MoveForward), reset the pointer to its starting position (row 1, column 1) and display the message "`Trap triggered! Pointer reset to (1,1)`".

0 4 . 1 The PROGRAM SOURCE CODE that creates a new constant to represent a TRAP.

[1 mark]

0 4 . 2 **All** the PROGRAM SOURCE CODE for the amended subroutine `CreateBoard`.

[3 marks]

0 4 . 3 All the PROGRAM SOURCE CODE for the amended subroutine `MoveForward`.

[3 mark]

0 5 The **Skeleton Program** needs improvement to validate the user's input when selecting a board type in the `Main` subroutine. Currently, if the user enters an invalid choice (not 1, 2, 3, or 9), the program proceeds without any error message.

Modify the `Main` subroutine so that:

- If the user enters an invalid choice, the message `Invalid choice. Please try again.` is displayed.

- The program repeats the menu and prompt until a valid choice is entered.

Test your changes by running the **Skeleton Program** and then:

Running the program and entering **6**, then **2**.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

0 5 . 1 **All** the PROGRAM SOURCE CODE for the amended subroutine `Main`.

[4 marks]

0 5 . 2 SCREEN CAPTURE(S) showing the results of the requested test.

[1 mark]

Turn over for the next question

0 6 The **Skeleton program** is to be improved with **backward** movement capability in the `ProcessInput` subroutine.
Change the `ProcessInput` subroutine so that:

- Add a new command "**BK**" that moves the pointer backward in the opposite direction
- Check for obstacles in the backward direction
- Maintain all boundary checks
- End the game immediately
- For "**BK**" with no number, move 1 step backward
- For "**BK n**", move n steps backward
- Set errorCode to 1 if movement is blocked

Test your changes by running the **Skeleton Program** and then:

- Select the example board (option 2).
- Trying "**BK**" and commands
- Observing movement when path is clear/blocked.

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

0 6 . 1 **All** the PROGRAM SOURCE CODE for the amended subroutine `ProcessInput`.

[5 marks]

0 6 . 2 SCREEN CAPTURE(S) showing the results of the requested test.

[1 mark]

0 7 The **Skeleton program** is to be improved with diagonal movement capability in the `SpecialMove` subroutine.

Change the `SpecialMove` subroutine so that:

Add four new special moves:
- "SPC then UL" (Up-Left)
- "SPC then UR" (Up-Right)
- "SPC then DR" (Down-Right)
- "SPC then DL" (Down-Left)

Each move should check axis obstacles
Display "Moved diagonally [direction]" after execution
The pointer should not move if either path is blocked

Test your changes by running the **Skeleton Program** and then:

Trying all four diagonal commands
Observing movement when paths are clear/blocked

Evidence that you need to provide
Include the following evidence in your Electronic Answer Document.

0 7 . 1 **All** the PROGRAM SOURCE CODE for the amended subroutine `SpecialMove`.

[7 marks]

0 7 . 2 SCREEN CAPTURE(S) showing the results of the requested test.

[1 mark]

Turn over for the next question

Section C (Longer programming questions)

You are advised to spend no more than **45 minutes** on this section.

0 8 The **Skeleton Program** is to be improved to add a bonus cell that clears surrounding obstacles. Modify the `CreateExampleBoard` and `MoveForward` subroutines to:

Task 1
Place a bonus cell represented by ***** at the fixed position (2, 5) (row 2, column 5) in the example board.

Task 2

When the pointer moves to the bonus cell, remove all obstacles (#) in the eight surrounding cells (up, down, left, right, and the four diagonals), replacing them with empty spaces .

Task 3

Display the message "`Bonus activated! Surrounding obstacles cleared!`" when the bonus is triggered.

Test your changes by:

The bonus cell must be placed in `CreateExampleBoard` at (2, 5).

Only obstacles (#) in the eight surrounding cells are cleared, not other symbols (e.g., @, *).

0 8 . 1 **All** the PROGRAM SOURCE CODE for the new subroutine `CreateExampleBoard`.

[3 marks]

0 8 . 2 **All** the PROGRAM SOURCE CODE for the amended subroutine `MoveForward`.

[12 mark]

0 8 . 3 SCREEN CAPTURE(S) showing the results of the requested test.

[1 mark]

Turn over for the next question

0 9 The **Skeleton Program** is to be modified so Implement a new `Jump` subroutine that allows the pointer to jump over a single obstacle in the current direction when specific conditions are met.

Task 1
Create a new subroutine called `Jump` that:

- Check if there is an obstacle in the immediate next cell and an empty cell two spaces ahead in the current direction.
- If both conditions are met, move the pointer two spaces forward, effectively jumping over the obstacle.
- If not possible (either no obstacle to jump or no empty landing space), set errorCode = 1.
- Display appropriate success or failure messages.

Task 2
Modify the `ProcessInput` subroutine to handle a new JMP command that calls the Jump subroutine.

Task 3:Test your implementation using an example board with:

- An obstacle at (1,4)

- An empty cell at (1,5)

- Pointer starting at (1,3) facing right

- Verify the pointer jumps from (1,3) to (1,5) when JMP is executed.

0 9 . 1 **All** the PROGRAM SOURCE CODE for new subroutine `Jump`.

[8 marks]

0 9 . 2 **All** the PROGRAM SOURCE CODE for the amended subroutine `ProcessInput`.

[3 mark]

0 9 . 3 SCREEN CAPTURE(S) showing the results of the requested test.

[1 mark]

END OF QUESTIONS

There are no questions printed on this page