A-level
COMPUTER SCIENCE

Paper 1

June 2019

Preliminary Material

To be opened and issued to candidates on or after 1 September 2018, subject to the instructions given in the Teachers' Notes (7517/1/TN).

Note

- The Preliminary Material and Skeleton Program are to be seen by candidates and their teachers **only**, for use during preparation for the examination on **Monday 3 June 2019**. It **cannot** be used by anyone else for any other purpose, other than that stated in the instructions issued, until after the examination date has passed. It must **not** be provided to third parties.

Information

- A Skeleton Program is provided separately by your teacher and must be read in conjunction with this Preliminary Material.
- You are advised to familiarise yourselves with the Preliminary Material and Skeleton Program before the examination.
- A copy of this Preliminary Material and the Skeleton Program will be made available to you in hard copy and electronically at the start of the examination.
- You must **not** take any copy of the Preliminary Material, Skeleton Program or any other material into the examination room.

There is no Preliminary Material printed on this page

INSTRUCTIONS FOR CANDIDATES

Electronic Answer Document (EAD)

Answers for all questions, for all sections, must be entered into the word processed document made available to you at the start of the examination and referred to in the question paper rubrics as the **Electronic Answer Document**.

Preparation for the Examination

You should ensure that you are familiar with the **Preliminary Material** and the **Skeleton Program** for your programming language.

AQA TEXT ADVENTURES

AQA TEXT ADVENTURES is a framework for playing text adventure games where the user gives commands by entering text, and the current state of the game is given to the user via text displayed on the screen. Text adventure games are one of the oldest types of computer games.

AQA TEXT ADVENTURES works in conjunction with a data file that provides the data to be used in the game. The data file contains details about the characters, locations and items for a scenario. This separation of the scenario from the program code means that the program can be used to play many different text adventure games as different data files containing different game data can be used.

In text adventure games the user inputs a simple sentence, e.g. `go north`. The sentence is then parsed. If it is a valid sentence that the program can understand, the action indicated by the command given in the sentence is completed by the character the user is controlling (called the player) and the current state of the game is changed. The user is informed of changes to the game state via text displayed on the screen. If the sentence is not valid then an error message is displayed.

The commands that are available to use are described in **Table 1**.

Table 1

Command	Description	Example
`go`	Moves the player from one location to another. The `go` command needs to be followed by one of the six valid directions: `north, south, east, west, up, down.`	`go north` Will take the player to the location north of the current location, if there is such a location and it is possible to go in that direction.
`get`	Allows the player to pick up an object that is in the same location as they are. The `get` command needs to be followed by the name of the object that the player wants to get. If the object can be got then it is added to the player's inventory and is no longer in its original location.	`get torch` Will add the torch object to the player's inventory if it is in the same location as the player and there is no other reason the torch cannot be got e.g. it is currently carried by another character.
`use`	Allows the player to use an object that is in their location or their inventory. The `use` command needs to be followed by the name of the object that the player wants to use. The effect of using an object depends on which object is being used, e.g. using a key in a room with a locked door will unlock the door (if it is the right key for that door).	`use silver key` Will try to use the silver key object.
`examine`	Provides the player with a more detailed description of an object or character in the same location as they are or an object in their inventory. The `examine` command needs to be followed by the name of the object or character that the player wants to examine. `Examine inventory` will display a list of the items the player is currently carrying.	`examine red die` If the red die is in the player's inventory or in the same location as the player a description of the red die will be displayed.
`say`	The player will speak, saying the text that follows the `say` command.	`say hello`
`quit`	Ends the game, quitting the program.	`quit`
`read`	Will display the text of objects that can be read e.g. books, notes, signs.	`read book` Will display the text written in the book object, if the book is in the same location as the player or in the player's inventory.

`move`	Allows the player to move an object that is in the same location as they are. The `move` command needs to be followed by the name of the object that the player wants to move. If the object can be moved then the result of moving the object will be displayed on the screen.	`move bed` Will move the bed object if the bed is in the same location as the player.
`open`	Allows the player to open an openable item that is in the same location as they are. The `open` command needs to be followed by the name of the item to open. This command is often used to open a door that is in-between two locations.	`open silver door` Opens the silver door if the player is in the same location as the silver door and the silver door can be opened.
`close`	Allows the player to close a closable item that is in the same location as they are. The `close` command needs to be followed by the name of the item to close.	`close silver door` Closes the silver door if the player is in the same location as the silver door and the silver door can be closed.
`playdice`	The player will play a dice game with another character. If the player wins then a list of the items that the losing character is carrying is displayed and the user chooses an item to take; if the player loses, the other character takes a random item from the player's inventory. The `playdice` command needs to be followed by the name of a character to play the dice game with. The character and the player need to be in the same location and both need to have a die.	`playdice guard` Plays a dice game with the guard character if the player is in the same location as the guard and both the player and the guard have a die.

The details that are stored about the characters, locations and items are detailed below.

Every character has a:

- unique ID
- name
- description
- current location.

The player's ID is 1001. All character IDs are less than 2000.

Every item has a:

- unique ID
- name
- description
- ID of its current location (if the location is greater than 2000 then the location is an item that is a container, in other words the item is inside another item)
- current status (e.g. a door could have a status of open; other status values are: locked, close, tiny, small, medium, large, edible, fragile, usable, gettable, water, liquid, container, horrible, occupied, off, powered, lightable, weapon)
- list of commands that can be used with the item (which will be one or more of: get, use, move, open, close, read)
- list of the results of using the listed commands (the results will be in the same order as the commands were listed in, though if nothing other than moving an item into the player's inventory happens when using the get command then no result will be specified as this is the default result).

All item IDs are greater than 2000.

Some items (doors) are actually two items – one in each of the locations separated by the item. When this is the case the difference between the IDs of the two items is 10000.

Any item that has a current location between 1001 and 1999 is being carried by a character (it is in a character's inventory).

A usable status means that the item can be used by the player even if it is not in their inventory as long as it is in the same location as the player. A gettable status means that the item is one that can be in the player's inventory (or that of another character in the game). An item can have more than one status; each status value for the item is stored in the same variable with commas separating the values. Some of these status values are not currently used in the Skeleton Program.

Every location has a:

- unique ID
- description
- ID of the location that can be found:
    - north
    - east
    - south
    - west
    - up
    - down

- from that location. If there is no location in a particular direction then this is indicated with the value 0. All location IDs are less than 2000.

Data Files

Two **Data Files** named **flag1.gme** and **flag2.gme** are supplied with the **Skeleton Program**. The **flag1.gme** data file provides the characters, locations and items for a text adventure game where the user needs to find a flag to win the game. The **flag2.gme** data file is the same text adventure game as in the **flag1.gme** file but with the player in a different starting location and some of the items are in different locations and have a different status.

END OF PRELIMINARY MATERIAL