AQA

# A-level
# COMPUTER SCIENCE

Paper 1

Monday 3 June 2019 Morning Time allowed: 2 hours 30 minutes

**Materials**
For this paper you must have:
- a computer
- a printer
- appropriate software
- the Electronic Answer Document
- an electronic version and a hard copy of the Skeleton Program
- an electronic version and a hard copy of the Preliminary Material
- an electronic version of the Data Files **flag1.gme** and **flag2.gme**.

You must **not** use a calculator.

**Instructions**
- Type the information required on the front of your 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 (also at the top of the front cover) of your Electronic Answer Document.
- Enter your answers into the Electronic Answer Document.
- Answer **all** questions.
- Save your work at regular intervals.

**Information**
- The marks for questions are shown in brackets.
- The maximum mark for this paper is 100.
- No extra time is allowed for printing and collating.
- The question paper is divided into **four** sections.

**Advice**
You are advised to allocate time to each section as follows:
**Section A** – 45 minutes; **Section B** – 20 minutes; **Section C** – 15 minutes; **Section D** – 70 minutes.

**At the end of the examination**
Tie together all your printed Electronic Answer Document pages and hand them to the Invigilator.

**Warning**
It may not be possible to issue a result for this paper if your details are not on every page of your Electronic Answer Document.

## Section A

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

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

You **must save** this document at regular intervals.

**0 1** **Figure 1** shows a bubble sort algorithm represented using pseudo-code. The algorithm sorts the data in a list `L`.

**Figure 1**

```
PROCEDURE BubbleSort(L)
  N ← LEN(L) - 2
  Count1 ← 0
  WHILE Count1 < LEN(L) - 1
    FOR Count2 ← 0 TO N
      IF L[Count2] > L[Count2 + 1] THEN
        Temp ← L[Count2]
        L[Count2] ← L[Count2 + 1]
        L[Count2 + 1] ← Temp
      ENDIF
    ENDFOR
    Count1 ← Count1 + 1
  ENDWHILE
ENDPROCEDURE
```

**0 1 . 1** Describe **two** changes that could be made to this bubble sort algorithm that would be likely to result in fewer comparisons being made when sorting the list `L`. The algorithm should still be a bubble sort algorithm if your suggested changes were made.

**[4 marks]**

**0 1 . 2** 'Sorting a list becomes an intractable problem when the size of the list is very large; it is a tractable problem when the size of the list is small.'

Explain why this statement is wrong.

**[2 marks]**

**0 1 . 3** Explain what approach(es) a programmer might take if asked to 'solve' an intractable problem.

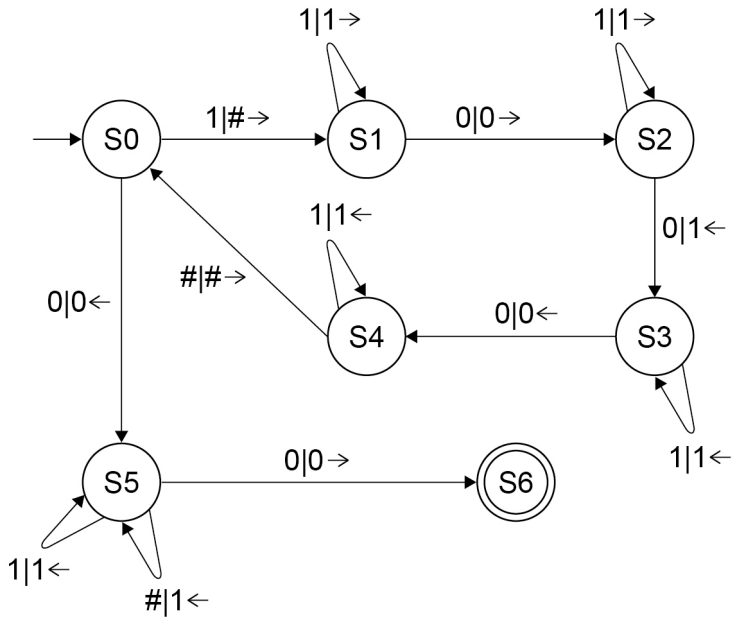
**[2 marks]**

**There are no questions printed on this page**

**Turn over for the next question**

**0 2** **Figure 2** shows the transition function (represented as a state transition diagram) and part of the tape for a Turing machine designed to complete a task. The current position of the read/write head is indicated by the * symbol.

**Figure 2**




| ... | 0 | 1* | 1 | 0 | 0 | 0 | 0 | ... |
|---|---|---|---|---|---|---|---|---|

The label 1|#→ on the transition from S0 to S1 means if the machine is in state S0 and a 1 is read from the tape then a # should be written to the tape and then the read/write head moved one cell to the right.

**0 2 . 1** After four steps of the computation have been completed, the current state, the tape contents and the position of the read/write head are:

**Tape**

| ... | 0 | # | 1 | 0* | 1 | 0 | 0 | ... |
|---|---|---|---|---|---|---|---|---|

**Current state**

| S3 |
|---|

Complete the unshaded cells of **Table 1** to show the result of tracing the computation of this Turing machine, from step five onwards. Show the contents of the tape, the current position of the read/write head and the current state as the input symbols are processed. Step four has been repeated at the start of the trace.

**Table 1**

| Tape | | | | | | | | | Current state |
|---|---|---|---|---|---|---|---|---|---|
| ... | 0 | # | 1 | 0* | 1 | 0 | 0 | ... | S3 |
| ... | | | | | | | | ... | |
| ... | | | | | | | | ... | |
| ... | | | | | | | | ... | |
| ... | | | | | | | | ... | |
| ... | | | | | | | | ... | |
| ... | | | | | | | | ... | |
| ... | | | | | | | | ... | |
| ... | | | | | | | | ... | |
| ... | | | | | | | | ... | |
| ... | | | | | | | | ... | |
| ... | | | | | | | | ... | |
| ... | | | | | | | | ... | |
| ... | | | | | | | | ... | |
| ... | | | | | | | | ... | |
| ... | | | | | | | | ... | |

Copy the contents of the unshaded cells in **Table 1** into the table in your Electronic Answer Document.

**[5 marks]**

**0 2 . 2** What is the purpose of the Turing machine shown in **Figure 2**?

**[1 mark]**

**0 2 . 3** What is the purpose of the transition from S5 to S6 in the Turing machine in **Figure 2**?

**[1 mark]**

**0 2 . 4** Explain what a Universal Turing machine is.

**[2 marks]**

**Turn over for the next question**

**0 3** **Figure 3** shows a partial solution to a logic puzzle. To complete the solution each of the letters A-I must appear exactly once in each row of nine cells, exactly once in each column of nine cells and exactly once in each of the collections of three-by-three cells shown with limits shown by a thicker border.

In **Figure 3** the logic puzzle has been completed except for the collection of three-by-three cells in the top-right corner.

**Figure 3**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| D | F | G | B | A | C | | | |
| I | E | H | F | G | D | | | |
| A | B | C | I | H | E | | | |
| B | G | D | E | C | H | A | F | I |
| F | H | I | D | B | A | E | G | C |
| C | A | E | G | I | F | B | H | D |
| H | D | B | C | E | G | F | I | A |
| G | C | F | A | D | I | H | B | E |
| E | I | A | H | F | B | D | C | G |

**0 3 . 1** Complete the solution for the puzzle shown in **Figure 3**.

Copy the contents of the unshaded cells in **Figure 3** into the table in your Electronic Answer Document.

**[1 mark]**

**Figure 4** shows a simpler example of this type of logic puzzle with fewer cells. In this simpler puzzle only the letters A-D are used.

**Figure 4**

| | | | |
|---|---|---|---|
| C | | | B |
| | | A | |
| | B | | |
| | | | A |

It is possible to represent this type of puzzle as a graph. To do this a unique number is given to each cell and a node containing this unique number is added to the graph. An edge between two nodes denotes a link between those two cells, meaning they cannot contain the same letter as each other.

**Figure 5** shows how unique numbers have been allocated to each cell in the puzzle in **Figure 4** and **Figure 6** shows an adjacency matrix that represents this puzzle.

**Figure 5**

| | | | |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 |

**Figure 6**

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| **1** | | | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| **2** | | | | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| **3** | | | | | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| **4** | | | | | | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| **5** | | | | | | | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| **6** | | | | | | | | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| **7** | | | | | | | | | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| **8** | | | | | | | | | | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| **9** | | | | | | | | | | | 1 | 1 | 1 | 1 | 0 | 0 |
| **10** | | | | | | | | | | | | 1 | 0 | 0 | 1 | 1 |
| **11** | | | | | | | | | | | | | 0 | 0 | 1 | 1 |
| **12** | | | | | | | | | | | | | | 1 | 1 | 1 |
| **13** | | | | | | | | | | | | | | | 1 | 1 |
| **14** | | | | | | | | | | | | | | | | 1 |
| **15** | | | | | | | | | | | | | | | | |

The graph in **Figure 6** can be considered to be both a representational abstraction and an abstraction by generalisation of the puzzle from **Figure 4**.

**Question 3 continues on the next page**

**0 3 . 2** What is representational abstraction?

**[1 mark]**

**0 3 . 3** What is abstraction by generalisation?

**[1 mark]**

**0 3 . 4** Other than the contents of the cells, what information has been removed from the puzzle in **Figure 4** when it has been represented as a graph?

**[1 mark]**

A graph can be represented using an adjacency list or as an adjacency matrix.

**0 3 . 5** Explain the circumstances when it would be more appropriate to use an adjacency matrix instead of an adjacency list.

**[2 marks]**

**0 3 . 6** Only the top half of the matrix in **Figure 6** needed to be used to present the puzzle. For which type of graph would the bottom half of the matrix also need to be used?

**[1 mark]**

**0 4**

**0 4 . 1** Explain the functionality of the `*` metacharacter when it is used in a regular expression.

**[1 mark]**

**0 4 . 2** Explain the functionality of the `?` metacharacter when it is used in a regular expression.

**[1 mark]**

**0 4 . 3** Complete **Table 2** to show which of the strings belong to the language defined by the regular expression `1|01+`.

**Table 2**

| String | Belongs to language (Y/N)? |
|---|---|
| 1 | |
| 11 | |
| 01 | |
| 0111 | |
| 0101 | |
| 111 | |
| 0011 | |

Copy the contents of the unshaded cells in **Table 2** into the table in your Electronic Answer Document.

**[3 marks]**

## Section B

You are advised to spend no more than **20 minutes** on this section.

Enter your answers to **Section B** in your Electronic Answer Document.
You **must save** this document at regular intervals.

The question in this section asks you to write program code
**starting from a new program/project/file**.

You are advised to save your program at regular intervals.

**0 5** Write a program that gets **two** words from the user and then displays a message saying if the first word can be created using the letters from the second word or not.

For example:

- The word EAT can be formed from the word ATE as the first word uses one E, one A and one T and the second word also contains one of each of these letters.
- The word EAT can be formed from the word HEART as the second word contains one E, one A and one T which are the letters needed to form the first word.
- The word TO can be formed from the word POSITION as the second word contains one T and (at least) one O which are the letters needed to form the first word.
- The word MEET cannot be formed from the word MEAT as the second word only contains one E and two Es are needed to form the first word.

You may assume that the user will only enter words that consist of upper case letters.

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

**0 5 . 1** Your PROGRAM SOURCE CODE.

**[12 marks]**

**0 5 . 2** SCREEN CAPTURE(S) showing the result of testing the program by entering:

- the word NINE followed by the word ELEPHANTINE.
- the word NINE followed by the word ELEPHANT.

**[1 mark]**

**Turn over for Section C**

## Section C

You are advised to spend no more than **15 minutes** on this section.

Type your answers to **Section C** into your Electronic Answer Document.
You **must save** this document at regular intervals.

These questions refer to the **Preliminary Material** and the **Skeleton Program**, but do not require any additional programming.

Refer **either** to the **Preliminary Material** issued with this question paper **or** your electronic copy.

**0 6** The Skeleton Program loads the game data from a binary file.

State **two** reasons why a binary file might have been chosen for use with the Skeleton Program instead of a text file.

**[2 marks]**

**0 7** The `GetIndexOfItem` subroutine is used to find the position of an item in `Items`. The item to find can be specified using either its name or its ID.

**0 7 . 1** Explain what it means if a value of -1 is returned by `GetIndexOfItem`.

**[1 mark]**

**0 7 . 2** How does the subroutine `GetIndexOfItem` know if it should try to find the item using the name or the ID passed to it as parameters?

**[1 mark]**

**0 7 . 3** What is the time complexity of the algorithm used to find an item in the `GetIndexOfItem` subroutine?

**[1 mark]**

**0 7 . 4** There is an error in the code for the Skeleton Program that does not cause a problem when using the data in the game files provided but could cause a problem with alternative game files.

Under what circumstances will the `GetIndexOfItem` subroutine return the index of an item that was not the item being searched for?

**[2 marks]**

If the Skeleton Program had been designed to use a hash table to store the items it is likely that it would be able to find the position of an item, when given its ID, in less time than the algorithm currently used in the subroutine `GetIndexOfItem`.

**0 7 . 5** Explain the steps involved in finding an item in the hash table. You should assume that all the items available in the game have already been successfully added into the hash table.

**[4 marks]**

**0 7 . 6** Give one reason why a hash table is **not** a suitable choice to store the items in the Skeleton Program.

**[1 mark]**

**0 8** A set is an unordered collection of values in which each value occurs at most once. The items in the game can be described using sets.

The set **A** contains all the items in the game.

The set **B** contains all the items that have "use" in their `Commands`.

The set **C** contains all the items that have "gettable" in their `Status`.

The set **D** contains all the items in the player's inventory.

The set **E** contains all the items which have a `Location` equal to the `CurrentLocation` of the player and that have "usable" in their `Status`.

The set **F** contains all the items which have a `Location` equal to the `CurrentLocation`.

Four operations that can be performed on sets are union, difference, intersection and membership.

**0 8 . 1** Explain how the operations can be used with some of the sets **A–F** to produce the set of items that the player can use in the current game state.

**[3 marks]**

**0 8 . 2** The set described in question **08.1** is a proper subset of some of the sets **A**–**F**.

List all of the sets (**A–F**) that it is a proper subset of.

**[1 mark]**

**0 8 . 3** Explain why the intersection of sets **F** and **C** does not contain all the items that the player can currently get.

**[1 mark]**

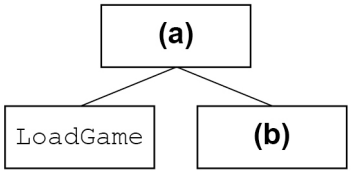
**0 8 . 4** Explain the difference between a subset and a proper subset.

**[1 mark]**

**Turn over for the next question**

**0 9** **Figure 7** shows an incomplete hierarchy chart for part of the Skeleton Program.

**Figure 7**




**0 9 . 1** What should be written in the box labelled **(a)**?

**[1 mark]**

**0 9 . 2** What should be written in the box labelled **(b)**?

**[1 mark]**

**0 9 . 3** State the name of an identifier for a user-defined subroutine in the Skeleton Program that uses exception handling.

**[1 mark]**

**0 9 . 4** Explain the difference between a local variable and a global variable.

**[1 mark]**

**0 9 . 5** State **one** reason why it is considered to be good practice to use local variables.

**[1 mark]**

## Section D

You are advised to spend no more than **70 minutes** on this section.

Type your answers to **Section D** in your Electronic Answer Document.
You **must save** this document at regular intervals.

These questions require you to load the **Skeleton Program** and to make programming changes to it.

**1 0** This question refers to the subroutine `PlayGame`.

The program is to be changed so that it does not always display the same message when the user enters a command that is not recognised.

Adapt the subroutine `PlayGame` so that sometimes the existing message `"Sorry, you don't know how to ***."` is displayed and sometimes the new message `"Sorry, I don't know what *** means."` is displayed. `***` denotes the command entered by the user.

The message to display should be selected randomly by the program and the probability of each of the messages being displayed should be equal. The new message should be laid out exactly as shown – all on one line with all the text, punctuation and spaces in the indicated places.

Test that the changes you have made work:

- run the **Skeleton Program**
- load the file **flag1.gme**
- enter the command `eat` a few times to show that your changes have worked and that the message displayed is randomly selected.

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

**1 0 . 1** Your PROGRAM SOURCE CODE for the amended subroutine `PlayGame`.

**[4 marks]**

**1 0 . 2** SCREEN CAPTURE(S) showing the results of the requested test.

**[1 mark]**

**Turn over for the next question**

**1 1** This question refers to the subroutine `GetItem`.

Currently the player can carry any number of items. The game is to be changed so that the player cannot carry more than five items.

Change the subroutine `GetItem` so that if the player is already carrying five items and they try to get an item that they would normally be able to get then the item is not added to their inventory. An appropriate message should be displayed saying that they can't get that item because they are carrying too much. If they are carrying fewer than five items then the subroutine should add the item to the player's inventory.

Test that the changes you have made work:

- run the **Skeleton Program**
- load the file **flag1.gme**
- enter the command `get torch`
- then enter the command `get red die`
- then enter the command `open cupboard door`
- then enter the command `go east`
- and then enter the command `get book`.

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

**1 1 . 1** Your PROGRAM SOURCE CODE for the amended subroutine `GetItem`.

**[7 marks]**

**1 1 . 2** SCREEN CAPTURE(S) showing the requested test. You must make sure that evidence for all parts of the requested test by the user is provided in the SCREEN CAPTURE(S).

**[1 mark]**

**1 2** This question extends the functionality of the game by modifying the subroutine `PlayGame` and adding a new subroutine called `DropItem`.

An additional command, `drop`, is to be added to the game. When a player uses the `drop` command the item specified after `drop` should be moved from the player's inventory to the location the player is currently in unless the item has a status of fragile, in which case it should be removed from the game as it breaks when dropped.

A message should be displayed stating either that the item has now been dropped, that the item broke when it was dropped or that the item is not in their inventory.

**What you need to do**

**Task 1**
Create a new subroutine called `DropItem` that implements all the functionality required for the `drop` command.

**Task 2**
Modify the `PlayGame` subroutine so that a call is made to the `DropItem` subroutine when the `drop` command has been entered by the user.

**Task 3**
Test that the changes you have made work:

- run the **Skeleton Program**
- load the file **flag1.gme**
- enter the command `drop apple`
- then enter the command `drop jar`
- then enter the command `examine inventory`
- then enter the command `open green door`
- then enter the command `go north`
- then enter the command `go south`.

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

**1 2 . 1** Your PROGRAM SOURCE CODE for the amended subroutine `PlayGame` and the new subroutine `DropItem`.

**[12 marks]**

**1 2 . 2** SCREEN CAPTURE(S) showing the requested test.

**[1 mark]**

**Turn over for the next question**

**1 3** This question refers to the subroutine `PlayDiceGame`.

The dice game in the **Skeleton Program** is very simple – the player and the other character both roll a die and whoever gets the highest number wins.

The `PlayDiceGame` subroutine is to be changed to a more complex dice game. All other parts of the subroutine are to remain the same – no changes should be made to checking that it is possible to play the dice game or to what happens when the player wins, loses or draws the game.

In the new game the player rolls their die three times. The largest of the numbers they roll is multiplied by 100 and the second largest of the numbers they roll is multiplied by 10. The results of these two multiplications are added together along with the smallest of the numbers they rolled to give the player's total score.

The other character's die is also rolled three times. The second number they roll is multiplied by 10 and the third number they roll is multiplied by 100. The results of these two multiplications are added together along with the first number they rolled to give their total score.

As with the original dice game, whoever has the highest total score wins the game and if the scores are the same the result is a draw.

The program should display the result of each die roll for each player and the total scores obtained by both players.

**New dice game examples**

- The player rolls a 4, then a 2 and then a 3. Their total score is 4×100 + 3×10 + 2 = 432. The other character rolls a 5, then a 2, then a 3. Their total score is 3×100 + 2×10 + 5 = 325. The player's score is higher so they win the game.

- The player rolls a 2, then a 3 and then a 4. Their total score is 4×100 + 3×10 + 2 = 432. The other character rolls a 6, then a 1, then a 4. Their total score is 4×100 + 1×10 + 6 = 416. The player's score is higher so they win the game.

**What you need to do**

**Task 1**
Modify the subroutine `PlayDiceGame`.

**Task 2**
Test that the changes you have made work by playing the new dice game with the guard twice:

- run the **Skeleton Program**
- load the file **flag2.gme**
- enter the command `playdice guard`
- choose a minimum value of 1 and a maximum value of 6 for the player's die each time the die is rolled
- if the player wins the game take any item from the guard; if the guard wins the game and the randomly selected item to take from the player is the player's die then you will need to re-run the **Skeleton Program** and load the file **flag2.gme** again before completing the next part of the test
- enter the command `playdice guard`
- choose a minimum value of 1 and a maximum value of 6 for the player's die each time the die is rolled.

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

**1 3 . 1** Your PROGRAM SOURCE CODE for the amended subroutine `PlayDiceGame`.

**[8 marks]**

**1 3 . 2** SCREEN CAPTURE(S) showing the tests described in **Task 2**.

**[1 mark]**

**END OF QUESTIONS**

**There are no questions printed on this page**

**There are no questions printed on this page**

JUN197517 1