AQA

AS-LEVEL
COMPUTER SCIENCE

7516/1: Paper 1
Report on the Examination

7516
June 2019

Version: 1.0

Further copies of this Report are available from aqa.org.uk

General

Most students were well prepared for this exam and had made good use of the time available between the release of the Preliminary Material and the day of the exam, although Section B (the questions about the Skeleton Program code) was often poorly answered with only a limited understanding of the Skeleton Program shown.

Students will not receive marks for captures of screens that have not been produced by running their code. Students should also make sure, when pasting in screen captures, that they are readable: the height of a capital letter should be at least 2mm when looking at the printed Electronic Answer Document (EAD). This is perhaps a skill that could be practised before the exam by making use of an EAD and a past paper.

When the evidence requested is amended program source code for a subroutine, the entire edited subroutine should be added to the Electronic Answer Document as requested in the question paper. Responses only showing the new code but not where in the subroutine it was inserted may not secure all the marks available. The code should be copied out of the program editor and pasted as text into the EAD, not as a screen capture.

A copy of the Skeleton Program used by the school/college should be included alongside the scripts sent to the examiner whether or not the Skeleton Program was modified. A significant number of school/colleges did not to do this. A few centres attached a copy of the Skeleton Program to each student Electronic Answer Document and, sometimes, also the exam paper which is not required.

Question 1

Many vague answers were given for the difference between local and global variables. Regardless of how the student's chosen programming language requires local/global variables to be declared, students need to be aware that a global variable is accessible to all parts of the program, whereas a local variable is accessible only in the program block in which it was declared. Many students' answers were very vague as to the reasons for using local variables. Some were able to state that memory used to store local variables is released when the subroutine terminates, but very few mentioned that local variables make a subroutine self-contained and therefore it is easier to re-use such subroutines in other programs.

Question 2

Some students struggled to complete the trace table. Although the majority of students correctly stated that the pseudo-code represented a bubble sort, or that it rearranged the numbers in descending order, many students did not show this arrangement in the trace table.

Question 3

This question was completed well by students with the majority of students achieving a high mark. A lot of full mark answers were seen across all of the programming languages. A significant number of students were not able to demonstrate their ability to convert the FOR loop into the exact equivalent in their chosen programming language. A common mistake was to misinterpret the starting and ending values for the loop counter. The pseudo-code clearly stated that the loop counter started at 1. However, many responses were seen where the loop counter started at 0, or

did not use `(Count – 1)` as the last value. A few students also dropped marks by not using the exact messages and identifiers given in the pseudo-code.

Most students completed the testing section well and provided clear screenshots of their code working.

As stated above, it is important that students consider the readability of their screenshots when pasting them into the Electronic Answer Document.

The majority of students recognised that the algorithm produced binary but, to gain the mark, they needed to state that the conversion was from decimal to binary.

Question 4

The majority of students secured both of the marks for this question. A few students copied across more than just the identifier for each part and therefore did not gain the mark as it was not clear which part of the copied code they believed the identifier to be.

Question 5

The majority of students gained this mark.

Question 6

Many students' answers were very vague as to the reasons for using exception handling in the subroutine `SetUpBoard`. Answers that were given credit included stating that this would avoid a program crash in case of file errors.

Question 7

Less able students stated that it was the player's pieces that would be contained in the parameter. To earn this mark, a reference to the positions of player A's pieces was required.

Question 8

The majority of students secured both marks for completing the labels for the state transition diagram.

Question 9

The hierarchy chart was attempted well by most students.

Question 10

The majority of students secured some of the marks for this question. The more effective answers mentioned that row 0 column 0 was used to store the number of moves and row 0 column 1 was used to store the number of pieces promoted to dames. The majority of answers correctly stated that there were 12 rows after row 0 because each player could have a maximum of 12 pieces, and that each of these rows stored the row and column coordinates as well as the state of the piece (0 for normal piece and 1 for a dame).

Question 11

Many responses just described the code without showing any understanding of the purpose of using modular arithmetic to find odd and even numbers. The description of the programming statements assigning the constant values SPACE needed to show understanding that these were the squares that the game could use. Similarly, the squares being assigned the value UNUSED were those that the game couldn't use.

Question 12

Many responses correctly stated that the `NumberOfMoves` variable was used for counting the number of possible moves. The more effective responses also stated that this variable was also used as the index for the data structure `ListOfMoves`.

Question 13

Very few responses provided clear explanations of what happens in the `WHILE` loop in `SelectMove`. Many descriptions merely reworded the programming steps. An example of an explanation gaining high marks was:

> The user is asked to enter a Piece ID. The list of moves is searched sequentially for an occurrence of the Piece ID entered until either the Piece ID is found or the end of `ListOfMoves` is encountered. If the end of the list is encountered the user is asked again to enter the Piece ID.

Question 14

Task 1 required meaningful error messages to be displayed as well as the error code.

Most students used either a series of `IF` statements or a `CASE` statement. Some responses used a dictionary or a tuple. All these methods were given credit. However, meaningful error messages were required. In particular, error code 3 needed to inform the user that a number was required as input.

Question 15

Task 1 required students to change the subroutine `ValidJump` to jump over an opponent's piece.

The majority of students made the small change to the final `IF` statement from checking the middle piece did not belong to the opponent to checking that it was the opponent's. A common mistake was to change the assignment of `OppositePiecePlayer` to be the same as `Player`. This may result in jumping over the opponent's piece, but it is poor programming practice as it removes transparency of code. `Player` and `OppositePiecePlayer` should never be the same value.

Question 16

Task 1 required the writing of a new subroutine to count the number of pieces a player had on the board.

The majority of students produced sensible code for this task and credit was given for designing the subroutine so that it could be called either with `A` or with `B` without duplicating code. Some

responses counted the pieces on the board rather than in the data structures `A` and `B`. Both methods were given credit.

Task 2 required rewriting the `PrintResult` subroutine to output the winner and scores using the amended rules.

The formula was given in the question but many students did not make use of the values held in the `A` and `B` data structures for number of moves made and number of dames held. A common error was to assume that the highest score wins, contrary to the information given in the question. Some students did not output the scores or consider the possibility of a draw. Both requirements were stated in the question.

Question 17

This question changed the rules of the game and allowed the player to take an opponent's piece. Some students were able to analyse the new requirements and that the `OpponentsPieces` data structure now needed to be passed to the `MoveDame` subroutine, as changes to this data structure now needed to be made.

Task 1 required new code for the `MoveDame` subroutine.

Many students secured some marks on this question. A common omission was not checking that the piece to be removed was an opponent's existing piece. Another common error was to fail to update the opponent's data structure with `-1` values in place of the co-ordinates.

As stated previously, program code should be copied out of the program editor and pasted as text into the EAD and not as a screen capture. For testing evidence, it is important that students consider the readability of their screen shots when pasting them into the EAD.

Mark Ranges and Award of Grades

Grade boundaries and cumulative percentage grades are available on the Results Statistics page of the AQA Website.