AQA

AS Computer Science

Paper 1
Report on the Examination

7516
June 2018

Version: 1.0

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 **when printed**. 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 whole of the edited subroutine should be added to the Electronic Answer Document. 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

The majority of students gained four marks from this question.

Question 2

Some students struggled to complete the trace tables. The question clearly stated that the pseudo-code described a method to test whether an integer is prime or not, yet some answers to 2.1 left the trace table showing that `5` was not prime, and some answers to 2.2 left the trace table showing that `25` was prime. Using a trace table to show how values change while iterating through a loop needs practice. Many answers indicated little understanding of repeatedly applying code until the condition(s) terminate the loop. Pseudo-code clearly shows what code is part of a loop by ending with `ENDWHILE` or `ENDFOR`. Similarly, `IF` statements terminate with `ENDIF`. Indentation is also used to help students see the logical flow of statements.

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 common mistake was to use real/float division instead of integer division. Some students did not implement the `IF ... ELSE IF ...` statement, instead using two separate `IF` statements, this type of solution was not given full credit. A few students also failed to gain marks by not using the exact messages and identifiers given in the pseudo-code. 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.

Most students completed the testing section well and provided clear screen shots of their code working. A few students did not get the mark because they tested using different values to those provided in the question. The screenshot needed to show clearly that the output was the result of a single program run, with three consecutive input values.

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

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 four marks, a common error being the switching of the identifiers `Signal` and `Symbol`.

Question 6

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 a subroutine in another program.

Question 7

Weaker students merely said that the variable `i` was a counter for the loop, which was not enough to gain credit. The better answers stated that the variable `i` was used as the index for the current character in the transmission string.

Many students missed out on some marks because they did not explain what happens in the specific case of a file of just spaces being read. This meant that `StripLeadingSpaces` will generate the empty string and therefore call the `ReportError` subroutine with the message `"No Signal received"`. Many students misinterpreted the code and thought the error message displayed would be `"No transmission found"`. The better answers stated that the empty string is passed back to `GetTransmission`, which in turn passes back the empty string to `ReceiveMorseCode`. This means that `LastChar` is set to `-1` and the loop is not entered.

Question 8

Many students did not realise that the significant point here was the length of the sequence of consecutive non-space characters. It is immaterial which symbol is used in the transmission string. So `"@@@"` would still be an acceptable string as it would be translated into a dash ('`-`'). Suitable examples of transmission strings generating the error message `"Non-standard symbol received"` would be `"xx"` or `"===="`.

The only acceptable symbols are a single non-space character or three consecutive non-space characters.

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. Many realised that the digits 0 to 9 need to be added to the `Letter` data structure and the equivalent Morse codes need to be added into the `MorseCode` data structure. Usually the `Dot` and `Dash` data structures were also mentioned as needing changes. The better answers mentioned that the tree structure in the Preliminary Material needed extending and therefore the pointers needed adding to the `Dot` and `Dash` data structures so that the digits could be decoded correctly. A few students gave answers that showed an insight beyond what was expected and gave answers including the issue that there would need to be empty locations in the `Letter` and `MorseCode` data structures to accommodate the fact that some Morse codes for digits did not immediately follow on from the letters. The better answers also pointed to the fact that the `SendMorseCode` subroutine needed to test for digits.

Question 11

This question was about validation of the input string. However, many students seemed to be under the misguided impression that checking whether a string was uppercase would be sufficient. In fact, this method merely checks that the alphabetic characters are uppercase; non-alphabetical characters would not be found using this method. Surprisingly, some students mentioned that they could not get the `ReportError` subroutine to work. The purpose of pre-releasing the Skeleton Program is for students to familiarise themselves with the subroutines and how they work. Students should be encouraged to experiment as much as possible with the Skeleton Program before the examination.

Some excellent solutions were seen. Each `PlainTextLetter`, that was not a space, could either be checked that it was in the range of 'A' to 'Z' or that its ASCII code was in the range of `65` to `90` (inclusive). Other creditworthy answers checked that every `PlainTextLetter` existed in the `Letter` data structure. Some students realised that the index calculated for any `PlainTextLetter` would be out of bounds of the `MorseCode` data structure if it was not a space and not an uppercase letter. They therefore used exception handling in order to catch the error of an invalid character. This was a creditworthy method of solution.

Question 12

A variety of creditworthy answers were seen in response to this question. The standard method was to generate the output string by appending the relevant signals while checking each symbol to detect whether it is a dot, a dash or a space: using `IF` statements, nested `IF` statements or CASE statements were all acceptable. Some students used `Replace/replace` and realised that they needed to replace the single spaces for double spaces before attempting to change a dot for a = followed by a space, and a dash for === followed by a space.

Question 13

Most students were able to code a loop that output each letter in turn. Many solutions copied the declarations of the `Letter` and `MorseCode` data structures into the new subroutine. This is not good programming practice and was not given credit. Students were expected to pass these data structures into the subroutine as parameters. The better answers also addressed the requirement that four letters and their Morse codes should be output per line. Some used a separate counter to check when four letters were processed. More elegant solutions used the modulus operator to check when the control variable of the loop was exactly divisible by 4.

The solutions that managed to display the alphabet in exactly the required table format used a variety of methods to pad the varying lengths of Morse code. Very few realised that the `ReportError` subroutine used a suitable formatting method that could be replicated in this subroutine code.

The question required students to include the source code of the subroutines `DisplayMenu` and `SendReceiveMessages` to show that the amendments requested in Task 2 had been made. Some students only showed the lines they added to the code. This is not sufficient: all the code of the amended subroutines should be pasted into the EAD.

Question 14

Task 1 required code to enter three values that had to be integers, but very few students included validation in their code for this. Some added the code to the subroutine `SendMorseCode` rather than `SendReceiveMessages` as the question asked. Students must be aware that they need to adhere to program specifications to be awarded the relevant marks.

Most students were able to encrypt a plain text character by adding a key to the index, although some did not encrypt spaces. The better answers included a method to alternate the three different keys. Many solutions included an adjustment to the changed index, but few checked that the resulting index was always within range for any integer key. Some solutions encrypted the complete plain text string before entering the original loop to calculate the letter index to convert plain text letters into Morse code, while other solutions produced the encryption as the letter index was calculated: either method is acceptable.

Mark Ranges and Award of Grades

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

Converting Marks into UMS marks

Convert raw marks into Uniform Mark Scale (UMS) marks by using the link below.

UMS conversion calculator