AQA Level 3 Technical Level IT
Computer Programming

UNIT NUMBER: F/507/6465

Mark scheme

Mark schemes are prepared by the Lead Assessment Writer and considered, together with the relevant questions, by a panel of subject teachers. This mark scheme includes any amendments made at the standardisation events which all associates participate in and is the scheme which was used by them in this examination. The standardisation process ensures that the mark scheme covers the learners' responses to questions and that every associate understands and applies it in the same correct way. As preparation for standardisation, each associate analyses a number of learners' scripts: alternative answers not already covered by the mark scheme are discussed and legislated for. If, after the standardisation process, associates encounter unusual answers which have not been raised they are required to refer these to the Lead Assessment Writer.

It must be stressed that a mark scheme is a working document, in many cases further developed and expanded on the basis of learners' reactions to a particular paper. Assumptions about future mark schemes on the basis of one year's document should be avoided; whilst the guiding principles of assessment remain constant, details will change, depending on the content of a particular examination paper.

Further copies of this mark scheme are available from: aqa.org.uk/tech-levels

The following list indicates the correct answers used in marking learners' responses to the multiple-choice questions:

KEY LIST

1	D
2	C
3	B
4	D
5	B

06 State three principles of good programming practice.

[3 marks]

Best principles are recommended methods of writing code.

1 mark for each principle

Examples:

- Avoid repetition of code
- Avoid adding functionality until you need it
- Make code as easy to read and understandable as possible
- Write code for person who is going to maintain it
- Follow standard conventions and practice
- Minimise dependencies in other areas of the program
- Place code that has similar functionality within same component
- Avoid optimising until the code is working
- Any other principle that is reasonable

07 Explain how you could demonstrate that user documentation meets the client's requirements.

[3 marks]

Learners may also consider user or audience requirements. Documentation could be in any format, including paper and online.

1 mark for testing:
- o with user base/audience
- o against client requirements

1 mark for research:
- o prior to completing document
- o qualitative research on draft

1 mark for citing structure or presentation methods, eg:
- o clear, concise, jargon-free writing
- o illustrative material

- only necessary technical terms

1 mark for any other valid explanation that assesses standard of documentation against requirements.

1 mark may be given for expansion points, eg

- testing with user base (1 mark) by giving documentation to the user and asking them to work through a scripted problem (1 mark for expansion)

08 a) Describe the difference between a global and local variable**.**

[2 marks]

1 mark for global scope (declared at start), ie can be used in any procedure or subroutine.
1 mark for local scope (declared within subroutine or programming blocks), ie variable not recognised outside part of program it was declared.

b) State one way in which the limitations of a local variable could be overcome without changing its type.

[1 mark]

1 mark for overcoming limitations, eg:

- local variable could be passed as a parameter to function/subroutine

09 State two characteristics of patch software and give one risk of using it.

[3 marks]

Characteristics (up to 2 marks)
1 mark for its function, to update/fix/improve a computer program, its data or libraries.
1 mark for recognising it is not a full version of software, eg addresses interim changes that need to be made, such as security patch.
1 mark for distributed as executable files or modifies the binary.

Risk (1 mark):

- may introduce new bugs
- not tested as exhaustively as a full release
- example of patch that has been problematic
- might break existing functionality

10 a) In the software lifecycle, describe one difference between an open and closed beta.
[2 marks]

1 mark for one difference, eg:
- o closed beta tests are filtered, eg to developers, registered customer base, while open beta are generally not (or less filtered)
- o open beta allows more representative testing of problems identified in closed beta
- o open beta more traditionally used with applications, closed beta often used with gaming
- o any other reasonable difference

b) Give one benefit of a closed beta.
[1 mark]

1 mark for a benefit, eg:
- o more filtered testing panel, eg:
  - selecting (especially gaming) matured and detached people who know they are testing over enthusiasts who only want to play (esp. gaming)
- o more discreet user base, limits reputational damage if testing throws up fundamental problems
- o any other reasonable benefit

11 a) Explain how logging expected and actual results against a bug report might help an IT Support desk identify a fault.
[2 marks]

Unexpected behaviour might not mean a genuine bug, ie the user is mistaken. By including expected behaviour, IT Support can assess whether misdiagnosing the expected behaviour is causing the user to report a bug.

1 mark for user may be mistaken.
1 mark for developer can compare bug report against with expected behaviour.

b) Give one other type of information that could be important if an accurate diagnosis is to be made.
[1 mark]

1 mark of other type of information, eg:
- o browser, hardware, software versions
- o memory dump
- o any other reasonable type

12 A user can trigger an event in event-driven programming by interacting with the program, such as through use of the mouse or keyboard. State three more ways an event can be triggered.
[3 marks]

For each of the following:

1 mark for:
- o objects can trigger their own events, eg upon status or value changes

- operating system can trigger events, eg from a timer on a webpage
- by calling the function module in the program
- when an error occurs
- any other acceptable method

13 a) Polymorphism is a characteristic of an object-orientated paradigm. Describe one other characteristic.

[2 marks]

1 mark for a characteristic, eg:
- abstraction
- encapsulation
- inheritance

1 mark for a description of the characteristic, eg:
- a characteristic of abstraction is
  - focusing on the essential and discarding the irrelevant
  - using keywords virtual (C++) or abstract and interface (Java)
  - puts responsibility on the programmer to implement a class/object
- a characteristic of encapsulation is
  - it protects the system from malicious attack
  - it is an information hiding mechanism
- a characteristic of inheritance is
  - when a class of objects is defined, any subclass can inherit the definitions
  - that subclass need not carry its own definition of data

b) Describe two main benefits of object-orientated programming.

[4 marks]

1 mark for each benefit, for a maximum of 2 marks.

1 mark for a benefit, eg:
- makes code more **maintainable**
- objects are **reusable**
- applications are more **scalable**
- OOP provides a clear modular structure
- OOP provides a good framework for code libraries

1 mark for a description of the benefit
- a benefit of 'maintainable' is that
  - objects are self-contained
  - identifying the source of errors becomes easier
- a benefit of 'reusable' is that
  - as objects are self-contained, this makes it easy to reuse code in other systems
- a benefit of 'scalable' is that
  - the interface provides a roadmap for reusing an object
  - the object can be replaced without affecting others
  - it is easy to replace inefficient code
  - new behaviours can be built from existing objects
- a benefit of 'modular' is that it makes it good for defining abstract data types
- a benefit of 'framework' is that it is
  - a good basis for developing GUIs

14 a) Explain the principle of familiarity when designing a user interface.

[2 marks]

1 mark for understanding principle of familiarity, eg:

- the user's model is based primarily on experience
- user interface components should reflect the user's model
- UI design must take account of the experience of the users

1 mark for one point expanding on the principle of familiarity, eg:

- a benefit of familiarity is that user of operating system automatically knows how to use standard elements, regardless of context
- as well as mirroring the design and concepts of an OS, the principle of familiarity also suggests that the terminology should be utilised
- a poorly designed interface is the reason why many systems are never used

b) Describe how familiarity could be achieved when designing a user interface in application for an operating system.

[4 marks]

1 mark for each point or expansion listed, eg:

- Use familiar OS user interface components to offer standard functionality
  - search and help techniques
  - playback controls
  - use of symbols/icons
  - reflect style of GUI
  - expected menu systems and dialog boxes
- take account of the needs, experience and capabilities of the system users
  - this might include considering physical and mental limitations such as through accessibility features
- there should be consistency in the way that operations are activated
  - behaviours should not surprise users
- guidance could be provided that links about to previous behaviour, eg:
  - rollover description of an unfamiliar icon

15 a) Describe the technique of pair programming.

[2 marks]

Description of technique (up to **2 marks**)

Pair Programming is when two programmers sit side by side next to a single computer to code an engineering task. The driver/developer writes the code, the observer/navigator reviews each line of code and the strategic direction of the work.

1 mark for two people working together to code
1 mark for roles of developer/navigator

b) Justify two possible pairings for a pair-programming task.

[4 marks]

Accept other pairings that can be justified (or the weaknesses acknowledged).

1 mark for each pairing, **1 mark** for each justification, eg:

- Expert Navigator, Expert Developer
  - Most productive pairing
- Expert Navigator, Novice Developer
  - Good way to train staff

16 a) Describe the purpose of workflow testing**.**

[2 marks]

Workflow testing is scripted testing which replicates specific workflows expected to be the working style of the user. It's purpose is to find the bugs most likely to negatively impact on user workflows.

1 mark for duplicates workflows of the user
1 mark to find bugs that impact user workflows

b) State two advantages and two disadvantages of black box testing.

[4 marks]

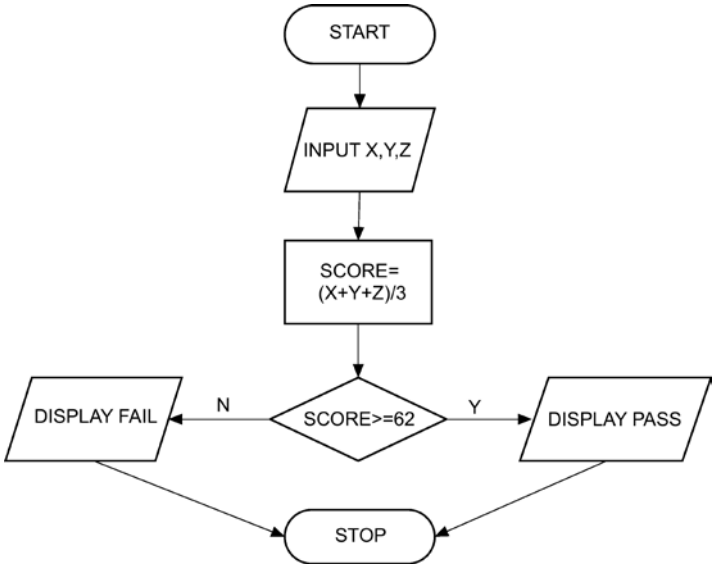
Black box (behavioural/acceptance) testing examines functionality of an application based on its specifications. This method of test can be applied to each and every level of software testing such as unit, integration, system and acceptance testing.

1 mark for each advantage (max 2 marks), **1 mark** for each disadvantage (max 2 marks).

Advantages	**Disadvantages**
Efficient for large code segments	Structure/design/implementation not known to tester
Environment also tested	Limited coverage/scenarios
Less skilled testers can test application with no knowledge of implementation, programming language or operating systems.	Tester cannot target specific code segments or error prone areas, many paths untested
Separate user/developer roles.	Reason for failure not always found
Test reproducible	Difficult to write test cases/identify enough inputs
User point-of-view helps identify issues	May repeat tests already done by programmer

17 a) Using appropriate symbols and flow lines draw a flowchart to represent the following algorithm.

[3 marks]



Maximum of **3 marks**:

- **1 mark** for appropriate I/O
  accept assigned variables instead of input
- **1 mark** for appropriate symbols
- **1 mark** for appropriate flow lines
  - all arrows must be pointing in right direction and not be omited
- **1 mark** for correct logic

b) Describe two benefits of using flowcharts to represent algorithms or processes.

[4 marks]

1 mark for a benefit (for a maximum of two benefits)
1 mark for each description of benefit, eg:

- it allows a complex problem to be functionally decomposed to a number of levels
- visual representation of a concept, clear to all parties without programming knowledge, easy to explain
- problem can be analysed in more effective way, eg conditional statements
- good program documentation, simple way to store a complex record
- blueprint during the systems analysis and development phase, effective synthesis
- helps in debugging and troubleshooting process, reduces time spent
- more efficient program maintenance

c) Compare and contrast the advantages and disadvantages of the Waterfall and Spiral development approaches.

[8 marks]

There are two approaches, Waterfall and Spiral

	Advantages	Disadvantages
Waterfall	• Clear documentation • Minimum planning overhead • One phase at a time • Simple to implement • Easy identification of milestones	• Less IT literate customers may not always know what they want • Can complicate teamwork if a stage has to be completed before moving on • Customer not involved, so not always developed for their needs • Can't go back to previous stage
Spiral	• Simplistic to implement and execute for projects • Limited resource allocation • Iterative framework • Revisit phases easily • Realistic model • Changes to software can be made at any point • Customer involved	• Completed phases cannot be revisited easily, no scope for backtracking/revising • Needs close risk assessment • Client may have to spend a lot of time with development team • Often no documentation • Difficult to fix the start/end of phase • Over-involvement of customer

Band	Descriptor	Marks
4	Candidate has fully compared and contrasted the advantages and disadvantages of both approaches	7-8 marks
3	Candidate has made some comparisons between the advantages and disadvantages of both approaches	5-6 marks
2	Candidate has listed some advantages and disadvantages of both approaches	3-4 marks
1	Candidate has listed some advantages or disadvantages	1-2 marks

18 a) State two possible efficiencies gained by moving blocks of code that are being repeated, to functions or subroutines.

[2 marks]

1 mark for each efficiency, eg:

- Makes code easier to read/debug
- More flexibility to pass parameters
- You should always try to reuse code where possible
- Mistakes only have to be corrected once, not for each recurrence
- Share code as libraries to plugin to different projects

The following pseudocode has been written to outline a program to input football results and output a league table and top scorers.

Some areas have been copied and pasted to repeat the code, and already the structure is beginning to look inefficient.

b) Rewrite the pseudocode to move the duplicate code into separate modules, such as functions and subroutines.

[10 marks]

The exemplar pseudocode on the following page is not intended as a model answer but merely to illustrate some of the possible approaches that may be taken. It is not necessary for the candidate to define variables in the pseudocode, nor for the solution to work in its entirely.

Band	Descriptor	Marks
4	Candidate has relocated duplicate code efficiently into subroutines/functions and passed parameters to them in a way which adds functionality	7-8 marks
3	Candidate has relocated most duplicate code into subroutines/functions and passed parameters to them in a way which adds some functionality	5-6 marks
2	Candidate has relocated some duplicate code into subroutines/functions and passed parameters to them	3-4 marks
1	Candidate has relocated some duplicate code into subroutines/functions	1-2 marks

In addition there are **2 marks** available for including the following:

1 mark for appropriate mechanism for automatically going back to the start when the end is reached

- eg a WEND/WHILE or DO/LOOP statement containing the pseudocode, or English language equivalent

1 mark for appropriate means of selecting an alternative in response to user input

- eg INPUT followed by IF or CASE statement. There are two aspects to the code (input and display) and an opportunity to ask the user which path to take and act accordingly.

Examples of possible approaches are given for the examiner but are not exhaustive; also see exemplar code.

c) Insert comments to explain the elements or structures.

[3 marks]

1 mark for each comment that explains how lines or blocks of code function

```
// let's put this in a continuous loop
WHILE (TRUE)

        OUTPUT "WHICH LEAGUE? [1-4]"
        INPUT NumLeague

        // let's create a menu to choose between input or display
        choose INPUT or DISPLAY from a menu

        IF INPUT is chosen THEN

                // let's collect this round of game data

                OUTPUT "HOW MANY GAMES TO INPUT?"
                INPUT NumGames

                CALL Input_Results (NumLeague, NumGames)

                Set Method to 'hi to low on points'

        ELSE

                // choose a method to sort league

                OUTPUT "Which sort method?"
                INPUT Method

        END IF

        CALL Print_League (NumLeague, Method)

        CALL Display_TopScorers (5, Num League)

// let's complete the loop
WEND

SUB Input_Results (NumLeague, NumGames)
Initialise counter to 1
        WHILE counter is less than or equal to NumGames
                INPUT GameDetails
                CALL Process_League (GameDetails)
        WEND
END SUB

SUB Process_League(GameDetails)
        Process GameDetails into TEAM/SCORE/SCORERS
        Calculate points for game
        Add the points to the team's total
```

```
        Increase team's games played by 1
        Add the goals to the scorer's total
END SUB

SUB Print_League (League, Method)
        CALL Sort_League (League, Method)
        display league table (League)
END SUB

SUB Display_TopScorers (NumScorers, League)
Initialise counter to 1
        WHILE counter is less than or equal to NumScorers
                display player number (counter)
        WEND
END SUB

SUB Sort_League (League, Method)
        Sort league table (League) by (Method)
END SUB
```

Assessment Outcomes coverage

Assessment Outcomes	Marks and % of marks available in section A	Marks and % of marks available in section B	Total marks
AO1: Understand the different types of computer programming, languages and the common uses	10 marks 12.5%	0 marks 0%	**10**
AO2: Analyse the tools and techniques for planning, design and development	11 Marks 13.75%	15 marks 18.75%	**26**
AO3: Evaluate the key features and techniques used in computer programming	19 Marks 23.75%	8 marks 10%	**27**
AO4: Demonstrate the principles of good program practice and user interface design	10 Marks 12.5%	7 marks 8.75%	**17**
Total Marks	**50**	**30**	**80**

Question	Assessment Outcome 1	Assessment Outcome 2	Assessment Outcome 3	Assessment Outcome 4	Question Total
1	1a (1)				1
2		2e (1)			1
3		2a (1)			1
4			3f (1)		1
5				4b(1)	1
6				4b(3)	3
7			3h(3)		3
8			3a(3)		3
9			3d(3)		3
10		2a(3)			3
11			3d(3)		3
12	1a(3)				3
13	1b(6)				6
14				4a(6)	6
15		2d(6)			6
16			3d(6)		6
17		2d(7) 2b(8)			15
18			3a(8)	4a(2) 4b(5)	15
Totals	**10**	**26**	**27**	**17**	**80**