LEVEL 3 TECHNICAL LEVEL
IT: Programming

F/507/6465 – Unit 2 Computer Programming

Mark scheme

June 2018

Version/Stage: 1.0 Final

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 students' 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 students' 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 students' 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

The following annotation is used in the mark scheme:

;	-	means a single mark
//	-	means alternative response
/	-	means an alternative word or sub-phrase
A	-	means acceptable creditworthy answer
R	-	means reject answer as not creditworthy
NE	-	means not enough
I	-	means ignore
DPT	-	in some questions a specific error made by a candidate, if repeated, could result in the candidate failing to gain more than one mark. The DPT label indicates that this mistake should only result in a candidate losing one mark on the first occasion that the error is made. Provided that the answer remains understandable, subsequent marks should be awarded as if the error was not being repeated.

Level of response marking instructions

Level of response mark schemes are broken down into levels, each of which has a descriptor. The descriptor for the level shows the average performance for the level. There are marks in each level.

Before you apply the mark scheme to a student's answer read through the answer and annotate it (as instructed) to show the qualities that are being looked for. You can then apply the mark scheme.

Step 1 Determine a level

Start at the lowest level of the mark scheme and use it as a ladder to see whether the answer meets the descriptor for that level. The descriptor for the level indicates the different qualities that might be seen in the student's answer for that level. If it meets the lowest level then go to the next one and decide if it meets this level, and so on, until you have a match between the level descriptor and the answer. With practice and familiarity you will find that for better answers you will be able to quickly skip through the lower levels of the mark scheme.

When assigning a level you should look at the overall quality of the answer and not look to pick holes in small and specific parts of the answer where the student has not performed quite as well as the rest. If the answer covers different aspects of different levels of the mark scheme you should use a best fit approach for defining the level and then use the variability of the response to help decide the mark within the level, ie if the response is predominantly level 3 with a small amount of level 4 material it would be placed in level 3 but be awarded a mark near the top of the level because of the level 4 content.

Step 2 Determine a mark

Once you have assigned a level you need to decide on the mark. The descriptors on how to allocate marks can help with this. The exemplar materials used during standardisation will help. There will be an answer in the standardising materials which will correspond with each level of the mark scheme. This answer will have been awarded a mark by the Lead Examiner. You can compare the student's answer with the example to determine if it is the same standard, better or worse than the example. You can then use this to allocate a mark for the answer based on the Lead Examiner's mark on the example.

You may well need to read back through the answer as you apply the mark scheme to clarify points and assure yourself that the level and the mark are appropriate.

Indicative content in the mark scheme is provided as a guide for examiners. It is not intended to be exhaustive and you must credit other valid points. Students do not have to cover all of the points mentioned in the Indicative content to reach the highest level of the mark scheme.

An answer which contains nothing of relevance to the question must be awarded no marks.

Question	Guidance	Mark
01	**C**	**1**
02	**C**	**1**
03	**D**	**1**
04	**B**	**1**
05	**C**	**1**

Question	Guidance	Mark
06.1	**1 mark** (max 2 marks) for each characteristic, eg: • ease of use, closest to human language/easier to write/read • needs to be compiled/translated before execution // source code • ease of error detection • machine independent • library functions • object-oriented // uses data types/structures, eg variables, arrays, classes, objects • more abstraction, machine's representation of information hidden • selection statements and repetition/iteration constructs • clear, well-defined syntax // some based on non-English languages • (often) multi-paradigm.	**2**
06.2	**1 mark** (max 1 mark) for an advantage, eg: • doesn't need to be translated • speed/efficient/optimal use of memory // execution time is faster • more control, with qualifying statement (eg over execution/what happens at level of CPU) • machine code is 1s and 0s which the processor understands • machine code is 1s and 0s which equates to electrical pulse on/off.	**1**
07.1	**1 mark** for: • Logical Composition [or inversion] with the Assistance of Computers.	**1**
07.2	**1 mark** for an example, eg: • those who work in creative fields • programmers working in artificial intelligence • specific example, eg users in design, music.	**1**
08.1	**1 mark** for organisation, eg: • a statement showing that quad=four people • an arrangement of four, eg one driver and three navigators; director, tester, two developers; • everyone sits around a computer which is projected onto a screen.	**1**
08.2	**1 mark** for usage, eg: • quad programming is good for team training/as a teaching tool • …particularly when introducing new architectural areas • in agile software development // extreme programming • for larger projects (eg due to development cost) // complex software • pooled expertise // description of one of the roles • fewer coding mistakes. **A.** Reasonable justification of answer to Question 08.1. **A.** Similar justifications to pair programming.	**1**

Question	Guidance	Mark
09.1	**1 mark** for: • index of arrays starts at zero // 31 is assigned to (index) 0 not 1	**1**
09.2	**1 mark** for: • month.	**1**
09.3	**1 mark** for: • 11. **A.** `daysinmonths(11)`	**1**
09.4	**1 mark** for reason, eg: • there is nothing returned from the function **(1 mark)** in `Lines 03 and 04` and the program prints the output from the function (**1 mark**) • the output is printed within the function **(1 mark)** but no value is returned (**1 mark**) • in Python, a function always returns '`None`' if no return statement has been reached (**1 mark**) • the function returns no value (**1 mark**) the code attempts to print the returned value (**1 mark**) • the word 'print' before the function call is causing it to print its value (**1 mark)** which is None at that point in the code (**1 mark**). Possible answer (max 3 marks): • when the function is called, an argument is passed to it via a parameter **(1 mark)**. An output is printed within the function but no value is returned **(1 mark)** to the print statement that calls it. Therefore, the calling print statement has no value from the list/array to print **(1 mark)**.	**3**
09.5	**1 mark** (max 1 mark) for: • Change `Line 04` to return, ie: `return(months[month])` • Remove the command print from `Line 06` and `Line 07`. **A.** Equivalent syntax in other languages. **A.** Remove [month] from `Line 04`	**1**

Question	Guidance	Mark
10.1	**1 mark** for: • (double) array • lists/tuples (Python) • hash table • dictionary • record • struct / structure • DataTable (class). **A.** String (eg CSV file could be stored in a string).	**1**
10.2	**1 mark** (max 2 marks) for each example, eg: • the storage of data as objects • object-oriented programming such class for storing information about a property in an entity in a table • databases • spreadsheets • specific examples of tabular data, eg CSV file, SQL server tables • metadata tables • repository tables • action tables • practical example, eg programming scenario requiring data tables, storing number of days in each month of the year, multiple user/customer inputs.	**2**

Question	Guidance	Mark
11	**1 mark** (max 2 marks) for each point, eg: • a pointer is a programming language **object** • whose value refers to/references another value // stores the memory address of another value • stored elsewhere in computer memory • using its memory address • usually used to allow the data stored at that memory location to be accessed • stores the location of another value • in a linked list, a pointer points to the next member of the list • The ReadLine method reads each line of text, and increments the file pointer to the next line as it reads.	**2**
12.1	**1 mark** for: • `num` **A.** `n2`	**1**
12.2	**1 mark** for: • `Lines 01`, `02`, `03` • `Lines 06`, `08`, `10`. **A**. **1 mark** for any 3 of the above. **A.** Lines written out as code. **A.** `06, 08` without the `10` (variable is not defined as per Question 12.3)	**2**
12.3	**1 mark** for: • `Line 10` **1 mark** for: • `num = n3` **R.** `n3` // change to `n3`	**2**
12.4	**1 mark** for: • `8`.	**1**
12.5	**1 mark** for: • `Line 07`. **A.** `elif (n2 >= n1) and (n2 <= n3)` **A.** `Line 10` // `num= n4` if this has not been given as the answer to Question 12.3.	**1**
12.6	**1 mark** for changing sign, ie: • `n2 >= n3` **A.** num = n3 if `Line 10` has been awarded the mark in Question 12.5.	**1**

Question	Guidance	Mark
13.1	**1 mark** (max 3 marks) for each point or expansion, eg: • technical processes not understood by client • client cannot visualise interface design • flowchart too complicated • difficult for client to feedback • may be difficult/costly/time-consuming to make/modify • a counterpoint about storyboarding.	**3**
13.2	**1 mark** (max 3 marks) for each point, eg: • to break down/functionally decompose a problem into more manageable components/key processes // simple representation helps visualisation // parts can be worked on one at a time • to present the structure of a concept that has multiple pages or components • explain architecture concepts to clients/developers • to get a better understanding of a program/data structure • a blueprint or template when developing // to have a clear idea of the order to code parts of a project • as a resource when debugging • as part of Unified Modelling Language/UML • this can be useful for large/complex projects • to create smaller tasks for groups/individuals // separate divisions within development team • class hierarchy, eg in object-oriented programming to show relationship between parent/class etc • for describing program behaviour, eg sequence, selection, iteration • appropriate drawing (explain points and/or example) • worked examples. **A.** representing the hierarchy of employees within a company. **R.** diagram if nothing added to written explanation (or vice versa).	**3**

Question	Guidance	Mark
14	Mark using the indicative content and levels of response table below. **Indicative content:**	6

Model	Advantages	Disadvantages
Waterfall • A linear model / step-by-step design • Each phase must be completed (or repeated) before the next can begin • **Cannot return to a previous phase unlike spiral** • **Used for small/medium projects with clearly defined requirements from outset, spiral can handle large projects** • Loops "Requirements > Analysis > Design > Coding > Integration > Testing > Acceptance > Maintenance" • **Both waterfall and spiral have planning in the early stages** • Resource planning done before each phase • Testing after coding phase	• Simple to understand and use • Easy to manage – each phase has specific deliverables and review process • Phases processed/ completed one at a time/do not overlap • Works well for smaller projects where requirements are well understood • **Easier to maintain than with spiral**	• Scope needs to be clear and detailed from the outset • Working software not given to client until late in the SDLC • Difficulty of managing team resources effectively • **Difficult /expensive to change client requirements later/or in testing phase, whereas spiral is more flexible to change** • **High level of risk and uncertainty because lack of risk management compared to spiral**

Spiral • Linear/iterative model • Combination of waterfall and prototype models • Medium - high-risk projects • **Realistic model compared to waterfall as changes can be made at any point** • **Client control / kept up-to-date ∴ more likely to have needs addressed; waterfall: client only involved at the beginning** • **Easier to change the design/revisit different phases than waterfall** • Determine objectives/constraints; evaluate/identify and resolve risks; develop and test; plan the next iteration • Multiple prototypes • Iteration over steps until customer satisfied with refined prototype • Testing after engineering phase • Repeats/spirals "Planning > Risk analysis > Engineering > Coding and implementation > Evaluation"	• High amount of risk analysis • Good for higher risk projects • Good for development of large and mission-critical projects • Additional functionality can be added later • Useful when significant changes are expected (research and exploration), eg new product line • **Working software produced early in SDLC at the end of each iteration, compared to end of lifecycle with waterfall**	• Requirements are complex • **Can be expensive compared to waterfall** • Completed phases cannot be revisited easily • Users can be unsure of needs • Little scope for backtracking/ revision • Needs expert for close risk assessment • Client may have to spend a lot of time with development team • **Waterfall has clear documentation of whole process, spiral less and more difficult to track** • Difficult to fix the start/end of phase • Difficult to commit long-term, eg potential changes to economic priorities.

Level	Descriptor	Marks
3	Clear comparison of both models.	5-6
2	Some understanding/comparison of both models or clear understanding of one model.	3-4
1	General understanding of one or both models is shown.	1-2
	No creditworthy response	0

Question	Guidance	Mark
15	**1 mark** (max 3 marks) for each way, **1 mark** for each expansion point, eg: • clear communications and training of staff/end-users • clear technical documentation • appropriate user documentation • clear requirements from the outset • availability of developers in case of need to issue bug fixes • regular consultation during development • operational handover/training • testing prior to deployment • tested/prepared for environment it is to be installed in (capacity, etc) • software installed and configured properly to begin with • follow good coding practice in coding phase, eg commenting • choosing an appropriate release window to allow for issues • regular updates • maintaining safe environment, eg virus checking • understanding/agreement of servicing requirements • logs of issues / feedback / automated error reporting • agreement to extend the lifecycle of the software product. **A. 2 marks** for a clear way/description, **1 mark** for a partial way/description or one which lacks clarity. **A.** If candidate has identified more than three ways, credit the best three.	**6**

Question	Guidance	Mark
16.1	**1 mark** (max 2 marks) for each point or expansion point, eg: • to collect/document requirements of a system from client, users and other stakeholders • to fully understand what a project will deliver • so developers know what the client wants from the product • problems of scope/understanding/volatility.	**2**
16.2	**1 mark** for each way, **1 mark** for an expansion point, eg: • carry out research • visualisation • clarity of scope • consistency of language • following organisational guidelines • use of templates • documenting dependencies • analysis of changes • user involvement/sign-off of documented requirements (**1 mark**) to ensure the product will deliver what users expect • client sign-off • make use of staff (eg business analyst) (**1 mark**) skilled in obtaining/documenting requirements (**1 mark**) • an example of poor quality (**1 mark**) and a way to improve this (**1 mark**).	**4**

Question	Guidance	Mark
17	**Indicative content:** • responsive web design, eg media queries, platform, screen sizes/resolutions, font sizes • ensure UI elements appropriate/usable on touch UI, eg consider mouse-specific elements like mouseover • consistent features between versions / familiarity • don't hide features/content on mobile • consider areas/elements in site that might need to be more front and centre on phone version, eg restaurant site might prioritise mobile payment/menu/booking • use of libraries, eg CSS, bootstrap • compatibility issues • design for mobile then scale things up, eg size of links • don't clutter the design/consider speed of loading • user testing • test the mobile site, and test changes haven't affected original site • research, storyboarding, etc • remove advertisements, reduce size of images etc (mobile data/speed) • stop videos auto-playing • general principles of good programming practice from specification, eg familiarity.	9

Level	Descriptor	Marks
3	A range discussed with clear understanding, focused on the scenario and relevant to the user, including things to avoid doing.	7-9
2	A range discussed with some understanding, mostly focused on the scenario and relevant to the user.	4-6
1	Makes general points, for example lists principles of good programming practice.	1-3
	No creditworthy response	0

Question	Guidance	Mark
18.1	**Indicative content:** • logic implied by The Rules and layout of the maze • variables or object types, declarations/assignment. • reading/verifying movement inputs and outputs to screen, eg player position, numbers on grid • sequence of actions • checking ahead, eg for black squares, edges of grid and resultant effect on player position • decisions/processes, eg keeping score, removing from grid (eg manipulating arrays/variables) • constructs • functions • loops/termination, eg keeping the game running/checking for input until all the numbers are picked.	**9**

Level	Descriptor	Marks
3	Explains a range of features and techniques required; examples are relevant and show understanding of the logic needed to program the game.	7-9
2	Describes some features and techniques required; most of the examples are relevant and show some understanding of the logic required to program the game.	4-6
1	Lists some features and techniques or derives some logical statements from the problem.	1-3
	No creditworthy response	0

Question	Guidance	Mark
18.2	**1 mark** for each point or expansion point, eg: • expand arrays to accommodate a second player, points, lives, etc • the logic/AI of the computer player, eg random moves, reactions, does it learn, does it attack? • whether player/computer can occupy the same space/cross over and consequences • whether player/computer play at the same time • logic to compare player and computer scores at the end of the game and declare winner.	**6**

Assessment Outcomes					
Question	**AO1**	**AO2**	**AO3**	**AO4**	**Question Total**
Section A					
01			3a (1)		1
02	1b (1)				1
03			3a (1)		1
04			3d (1)		1
05			3c (1)		1
06.1	1a (2)				2
06.2	1a (1)				1
07.1	1c (1)				1
07.2	1c (1)				1
08.1		2f (1)			1
08.2		2f (1)			1
09.1			3a (1)		1
09.2			3a (1)		1
09.3			3a (1)		1
09.4			3e (3)		3
09.5			3e (1)		1
10.1		2g (1)			1
10.2		2g (2)			2
11			3a (2)		2
12.1			3c (1)		1
12.2			3a (2)		2
12.3			3d (2)		2
12.4			3e (1)		1

12.5			3e (1)		1
12.6			3e (1)		1
13.1		2f (1)		4a (2)	3
13.2		2d (3)			3
14		2b (6)			6
15		2a (6)			6
Section B					
16.1		2a (2)			2
16.2		2a (4)			4
17				4a (9)	9
18.1		2d (9)			9
18.2			3e (6)		6
Totals	**6**	**36**	**27**	**11**	**80**