Level 3 Technical Level IT

COMPUTER PROGRAMMING

Mark scheme

Unit Number: F/507/6465 and R/506/6118 (2015)
Series: June 2017

Version: 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

MARKING METHODS

In fairness to candidates, all examiners **must** use the same marking methods. The following advice may seem obvious, but all examiners **must** follow it as closely as possible.

1 If you have any doubt about how to allocate marks to an answer, consult your Team Leader.

2 Refer constantly to the mark scheme and standardising scripts throughout the marking period.

3 Use the full range of marks. Don't hesitate to give full marks when the answer merits them.

4 The key to good and fair marking is **consistency**.

INTRODUCTION

The information provided for each question is intended to be a guide to the kind of answers anticipated and is neither exhaustive nor prescriptive.

All appropriate responses should be given credit.

Where literary or linguistic terms appear in the Mark Scheme, they do so generally for the sake of brevity. Knowledge of such terms, other than those given in the specification, is not required. However, when determining the level of response for a particular answer, examiners should take into account any instances where the candidate uses these terms effectively to aid the clarity and precision of the argument.

DESCRIPTIONS OF LEVELS OF RESPONSE

The following procedure must be adopted in marking by levels of response:

- read the answer as a whole
- work up through the descriptors to find the one which best fits
- where there is more than one mark available in a level, determine the mark from the mark range judging whether the answer is nearer to the level above or to the one below.

Since answers will rarely match a descriptor in all respects, examiners must allow good performance in some aspects to compensate for shortcomings in other respects. Consequently, the level is determined by the 'best fit' rather than requiring every element of the descriptor to be matched. Examiners should aim to use the full range of levels and marks, taking into account the standard that can reasonably be expected of candidates.

Key

DNA = Do not allow

Where candidates have used diagrams to clarify their answers, examiners should consider diagrams and written statements equally in applying the mark scheme.

Section A

1. B

[AO1c, 1 mark]

2. D

[AO1a, 1 mark]

3. A

[AO1a, 1 mark]

4. A

[AO3a, 1 mark]

5. D

[AO3c, 1 mark]

6. Backtracking is a general algorithmic technique.

Explain how a backtracking algorithm works.

[AO3c, 3 marks]

1 mark for each point, eg

- Considers searching every possible combination to solve an optimization problem
- Inserts knowledge into problem **(1 mark)** so search tree can be pruned **(1 mark)**
- View picking a solution as a **sequence of choices**
- Incrementally builds candidates to the solutions
- For each choice, **consider every option recursively**
- Attempts to solve sub-problems **(1 mark)** values already found ignored **(1 mark)**
- Returns the best solution found
- Admits the concept of a "partial candidate solution"
- Technical explanations or examples
- **DNA**: Backtracking = debugging

Example of a technical explanation (max. **3 marks**):

Used to solve problems **(0 marks)**. Starts with a possible move and then tries to perform next move. **(1 mark)** If next move works then this continues recursively until ultimate success **(1 mark)**, but if it fails then the algorithm backtracks one step and tries a different move. **(1 mark)** In this way, all paths will be tried until the answer is found or all paths are exhausted. **(1 mark)**

7. **Table 1** identifies two types of programming language.

Using the list below, place each characteristic in its correct position in **Table 1**.

[AO1a, 3 marks]

Characteristic:

- Source code
- Object code
- Mnemonics
- Abstraction

Table 1

Type of language	**Characteristics**		
Low-level	Object code	**(1 mark)**	Mnemonics **(1 mark)**
High-level	Source code		Abstraction **(1 mark)**

1 mark for object/source code the correct way around
1 mark for mnemonics (low-level)
1 mark for abstraction (high-level)

Allow: **1 mark** for a table completed with **four** accurate characteristics (ie not using the list provided).

8. This question is about the **modular development approach**.

8.1 Identify **two** advantages of the **modular development approach**.

[AO2a, 2 mark]

1 mark (max. **1 mark**) for advantage, eg

- Reuse of code, eg can be called when needed or not at all
- Scoping of variables is easier
- Standardised approach, eg allows many programmers to collaborate on different parts of the project
- Team can focus on modules specific to their expertise
- Thoroughness/frequency of testing
- Can be used in other programs, eg libraries
- Organisation of code/tasks, eg easier to break down problem/test, add/amend features, shorter/tidier code, code separated into modules
- Errors are localised/can be easily identified

8.2 What should be considered to prevent problems occurring during development?

[AO2a, 2 marks]

1 mark (max. **2 marks**) for each consideration, eg

- Thorough documentation of modules/readability of code
- Agree approach across all teams, eg naming conventions, deadlines/durations, regular testing/debugging, appropriate number of people, efficient organisation of code, central point of maintenance, time scales
- Splitting into/importing individual files (ease of navigation, management of code, etc)
- Risk assessment/planning, eg difficulty of task/anticipate problems, final outcome
- Technical debt
- Client instructions from client, understanding of the requirements/future uses of modules

OR

1 mark for a consideration, with **1 mark** for an expansion point

9. Software design or development lifecycle is a process used to design high quality software.

9.1 Describe what is meant by 'closed beta' testing.

[AO3d, 1 mark]

1 mark (max. **1 mark**) for a definition, eg

- Closed beta tests are filtered / restricted / by invitation only, eg to developers, registered customer base, etc
- Testing which is limited for a purpose, eg to prevent reputational damage.

9.2 Name **two** phases of the software design lifecycle that come after implementation or coding.

[AO2a, 2 marks]

1 mark (max. **2 marks**) for each phase, ie

- Testing, allow: Pre-alpha, Alpha, Beta, RC, RTM, GA
- Deployment/installation, allow: selling, release
- Documenting
- Maintenance/service
- Review/evaluation
- Execution
- Conclusion
- Closure

10. A local garage has asked you to create a navigation structure for its web page.

Arrange the following pages so a user could navigate easily around the website.

- Home
- Order history
- Book a repair
- Book an MOT
- Change my booking
- About us
- Contact us
- Customer accounts
- Our services
- Our history
- Our awards

[AO2f, 1 marks; AO4a, 2 marks]

Indicative content:

- Home
- About us > Our history > Our awards
- Our services > Book a repair > Book an MOT
- Customer accounts > Change my booking > Booking history
- Contact us

- Home > Book a repair > Book an MOT
- About us > Our history > Our awards > Our services
- Customer accounts > Change my booking > Booking history
- Contact us

Q10 Descriptor	**Marks**
The pages are grouped with clear logic to the navigation structure that would be clear to the client; all pages are included	3
The pages are grouped with some logic to the navigation structure which would be reasonably clear to the client; maybe one or two pages are missing.	2
The pages are grouped but the navigation structure is limited either by logic or presentation	1
No creditworthy response	0

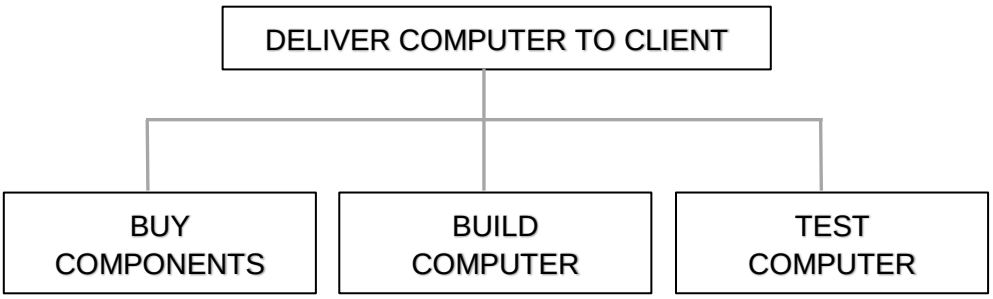
11. You buy components to build and test a new computer for delivery to a client.

Draw a **structure diagram** which illustrates this process.

[AO2d, 3 marks]

1 mark for a structure diagram with two levels (or other **simple** representation) which broadly reflects the process
1 mark for the outcome (eg deliver, sell, install)
1 mark for all three processes (or other implied, eg search for components, 'take delivery' instead of 'buy components')

Indicative content:



Allow other valid approaches/diagram, including flowcharts (consider whether the flowchart is a **simple** representation).

12. Explain why developers might use the Concurrent Versions System (CVS).

[AO3g, 2 marks]

1 mark (max. **2 marks**) for each point, OR: **1 mark** + **1 mark** for an expansion, eg

- to save and retrieve/access previous versions of the source code
- developers share control in a common repository
- to keep track of each developer's work individually

- to reconstruct a version from recorded changes / maintain a single copy
- allow multiple developers to work on separate copies of source code in isolation
- as a production quality system
- if a new build does not function **(1 mark)** you can rollback to/reconstruct an older version **(1 mark)**

13. Pseudocode is an informal language which helps programmers develop algorithms.

Rewrite the following problem as pseudocode:

[AO2e, 3 marks; AO3a, 3 marks]

- Input 50 positive numbers.
- Add up the odd numbers and display the total.
- The program should terminate and display the current total if a **negative number** is input at any point.

Indicative content:

```
01  total=0
02  count=1
03  input number
04  while (count<=50) and (number>=0)
05        if number is odd, add number to total
06        increase count by 1
07        input number
08  endwhile
09  display total
```

Award up to **3 marks** for each of **two** components.

Allow: total occurrences of odd numbers rather than total of odd numbers.

Q13 Descriptor (first component)	**Marks**
Candidate has included all elements of the problem	3
Candidate has included most elements of the problem	2
Candidate has included some elements of the problem	1
No creditworthy response	0

Q13 Descriptor (second component)	Marks
Candidate has constructed a logical, working solution	3
Some of the solution is logical	2
The pseudocode is limited	1
No creditworthy response	0

14. Programming languages can be used to enhance web page interactivity.

14.1 Name **two** languages that could be executed directly on a web server.

[AO1c, 2 marks]

1 mark (max. **2 marks**) for each language, eg

- VBScript
- Java
- Python
- PHP
- ColdFusion
- Ruby
- C/C++
- (Server-side) JavaScript, HTML
- (Server-side generated) CSS

Allow:

- ASP (scripting environment)
- ASP.NET (framework that would use JScript/C# as the language)
- SQL, **DNA:** MySQL

14.2 Identify **one** language that could be used to enhance web page interactivity and, using examples, explain how this could be done.

[AO1c, 4 marks]

1 mark (max. **1 mark**) for a language, eg

- JavaScript
- ActionScript
- jQuery
- PHP

Allow:

- CSS

Award up to **3 marks** for the explanation.

Indicative content (two possible example/explanation combinations):

JavaScript **(1 mark)** can be used to add interactivity to web pages by adding code to HTML tags **(1 mark)** such as <button *onclick*="myFunction()"> **(1 mark)** which calls an event when the button is clicked **(1 mark)**.

JavaScript **(1 mark)** could be used to change the appearance of a page in reaction to the activity of the user **(1 mark)**, eg disabling submit buttons **(1 mark)** on a page until all of the form is completed correctly **(1 mark)**

Q14.2 Descriptor	**Marks**
Candidate has written an explanation which shows clear understanding	3
Candidate has written an explanation which shows some understanding	2
Candidate has written a limited explanation	1
No creditworthy response	0

15. A programmer using a high-level language should write code so that it can be maintained easily.

15.1 Explain the abstraction principle in terms of good programming practice.

[AO4a, 3 marks]

One definition of **abstraction principle**:

Each significant piece of functionality in a program should be implemented in just one place in the source code. Where similar functions are carried out by distinct pieces of code, it is generally beneficial to combine them into one by abstracting out the varying parts. **(3 marks)**

1 mark (max. **3 marks**) for each point, eg

- The use of abstractions or libraries reduces duplication **(1 mark)**
- Functionality should be implemented in just one place / similar functions should be combined into one **(1 mark)** using abstraction **(1 mark)**
- Abstraction reduces complexity/increases efficiency of the program / it is **more efficient** to combine functions into one **(1 mark)**
- Justification in terms of good programming practice **(1 mark)**

15.2 State **three** more principles you could follow to make code easier to maintain.

[AO4a, 3 marks]

1 mark (max. **2 marks**) for each principle, eg

- Document code, eg comment lines, 'readme.txt'
- Avoids unnecessary/duplicate code
- Indent/present code clearly, (allow: CSS, libraries, source files)
- Name variables/functions/objects etc consistently
- Name variable/functions/objects etc sensibly
- Separate code into different folders, eg public/public, include/source
- Specific characteristics, eg encapsulation, inheritance, polymorphism, decomposition

16 Iterative design is commonly used to develop human computer interfaces.

Explain the typical steps of iterative design when programming user interfaces.

[AO3f, 4 marks; AO4a, 2 marks]

Indicative content:

- Iteration: steady refinement of the design based on testing/evaluation
- A repeated circle of events
- Instead of delivering a final version in a single delivery it would be delivered first in a rough form, then subject to user feedback and testing
- Part of each iteration would involve studying how intuitive and efficient the interface is. This cycle would then repeat to refine the previous delivery until a final accepted and tested design
- Testing/fixing (at each stage instead of at the end)
- Refining based on user testing/feedback to improve usability
- Acquiring quantitative and qualitative feedback, eg to increase productivity while using the interface
- Assessing interface for ease of learning/use/familiarity
- Debugging the interface
- Steps, eg complete, present/test, note any problems; refine/repeat
- Diagrams which illustrate any of the above

Q16 Descriptor	**Marks**
Candidate has clearly explained iterative design in terms of user interfaces	5 - 6
Candidate has shown some understanding of iterative design, and how the steps relate to user interfaces	3 - 4
Candidate has shown limited understanding of iterative design, but attempted to link this to user interfaces	1 - 2
No creditworthy response	0

Section B

17. You are designing a character for a game. The character must make decisions about what to do in the 30 minutes from waking up to leaving the house.

Draw a flowchart which:

- Shows waking up to an alarm until leaving the house
- Handles a 5-minute snooze button
- Handles three other decisions required
- Checks important decisions before leaving

[AO2d, 8 marks; AO3a, 4 marks]

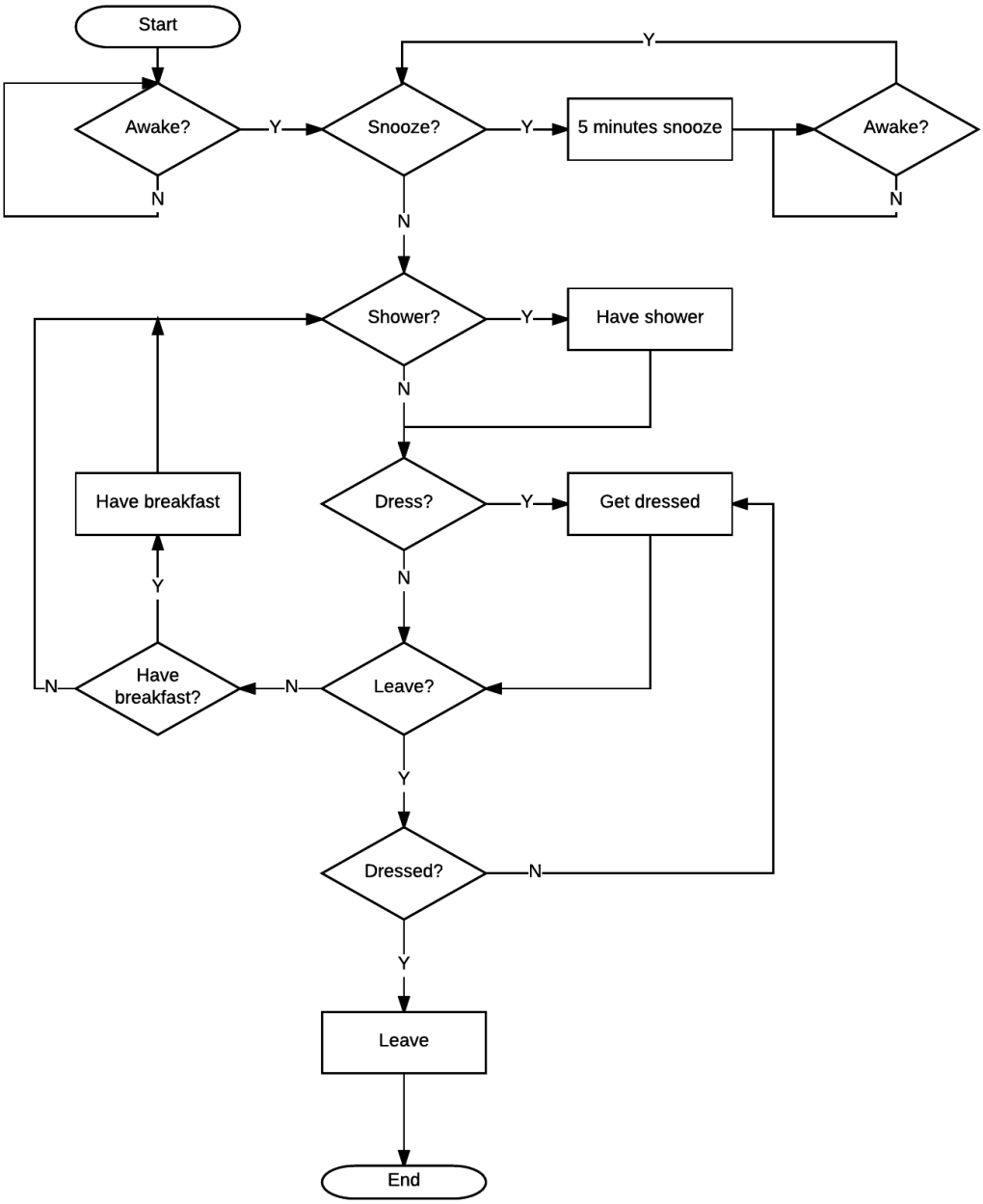
Flowchart Symbol	Name
	Start/end
	Input/output
	Process
	Decision

Indicative flowchart and descriptors on following page. Accept reasonable alternatives.

Choose the most appropriate band/row on a **best fit** basis.

Award up to **4 marks** in Column 2 (Logical) and up to **8 marks** in Column 2 (Accurate).

Q17 Descriptor	**Implemented**	**Logic and Accuracy**
3 or 4 bullets carefully implemented; eg alarm and snooze button sensible decisions identified (eg wash, dress, have breakfast) important decision identified (eg am I dressed?) the processes are logical, eg alarm and snooze processes work 5-minute timer; 30-minute count waking tasks are not performed while asleep producing accurate results: eg the character gets up the character does not leave house without dressing the character leaves the house	3-4	6-8
3 or 4 bullets implemented; eg alarm and snooze button some sensible decisions some of the processes are logical, eg a snooze button attempted and processes mostly logical there is no 5-minute timer most decisions are in a logical order producing mostly accurate results.	2-3	3-5
1 or 2 bullets implemented; some processes attempted, though with limited results.	1-2	0-2
No creditworthy response.	0	0


Start
Awake?
Y
N
Snooze?
Y
N
5 minutes snooze
Y
Awake?
N
Shower?
Y
N
Have shower
Have breakfast
Dress?
Y
N
Get dressed
Y
N
Have breakfast?
N
Leave?
Y
Dressed?
N
Y
Leave
End

18. Examine the following code.

```
01  pageref=new Array()
02  var count=0
03
04  pageref=['home','default.asp','services','services.asp','price
05  s','prices.asp','contact','contact.asp','about
06  us','about.asp']
07
08  for (var i=0; i<pageref.length; i+=2) {
09    displaypage(i)
10  }
11
12  window.alert(count)
13  temp+=1
14
15  function displaypage(num) {
16    var x = filename(num)
17    window.alert('the '+pageref[num]+' page has filename '+x)
18    switch(num) {
19      case 4:
20        window.alert('add price list')
21        break;
22      case 6:
23        window.alert('add contact database')
24        break;
25    }
26  }
27
28  function filename(num) {
29    var temp=0
30    count+=1
31      return pageref[num+1]
32  }
```

18.1 The code does not have any annotations to show how it works. The programmer has used `window.alert()` to output messages to the screen to help with debugging.

Write technical comments which **explain** how the code and program work.

Select **five** different aspects to comment on. Marks are awarded for:

- selecting different aspects carefully (eg, don't comment on two `window.alert` commands)
- the technical understanding shown
- using clear and concise language.

[AO3a, 8 marks; AO3h, 3 marks]

Choose the most appropriate band/row on a **best fit** basis.

Award up to **3 marks** in Column 2 (Selection) and up to **5 marks** in Column 2 (Explanation/Understanding).

Descriptor	**Selection**	**Explanation/ Understanding**
5 aspects of code judiciously selected, with appropriate explanation	3	3–5
3–4 aspects of code carefully selected; there is some appropriate explanation of the choices made, **OR** 5 or more aspects of code selected; there are some appropriate observations about the choices made	2	2–4
1–2 aspects selected with some understanding of the code, **OR** 3 or more aspects selected; there is limited understanding of the code	1	0-2
No creditworthy response	0	0

Further, award up to **3 marks** for the appropriateness of the language:

Descriptor	**Language**
Most of the language is clear	3
Some of the language is clear.	2
Little thought has been given to the clarity of the language	1
No creditworthy response	0

18.2 There is an exception error in the code. Give the number of the line where it occurs and explain why it happens.

[AO3e, 3 marks]

1 mark for
- Line 13

1 mark (max. **2 marks**) for reasons, eg variable temp
- has not been defined
- has not been initialised **(1 mark)** so it cannot be incremented **(1 mark)**
- has not been assigned a value
- is local when it needs to be global

- should be defined outside the function / has been defined inside the function

Allow: **1 mark** (max. **1 mark**) if candidate does not identify exception error but identifies another 'error' (even if non-exception), eg a potential difference in syntax or outcomes which might be applicable in some situations/languages (optional semi-colons, differences in command verbs/or absence of, by calling a value outside of the dimensions of the array, etc).

18.3 Apart from `window.alert()`, describe **two** other techniques you could use to test and debug the code.

[AO3d, 4 marks]

1 mark (max. **2 marks**) for each technique, eg

- write output to a log file
- breakpoints
- watching expressions
- trace tables
- use a debugger and step through code
- reference to specific solutions in relation to code

DNA

- actual vs expected results (without `window.alert` then there would be no results in this example to compare)

1 mark (max. **2 marks**), for either of

- an expansion point on each of the above techniques
- two expansion points on one of the above techniques

Question	Assessment Outcomes (2015)				TOTAL
	1	2	3	4	
SECTION A					
1	1c (1)				1
2	1a (1)				1
3	1a (1)				1
4			3a (1)		1
5			3c (1)		1
6			3c (3)		3
7	1a (3)				3
8.1		2a (2)			2
8.2		2a (2)			2
9.1			3d (1)		1
9.2		2a (2)			2
10		2f (1)		4a (2)	3
11		2d (3)			3
12			3g (2)		2
13		2e (3)	3a (3)		6
14.1	1c (2)				2
14.2	1c (4)				4
15.1				4a (3)	3
15.2				4a (3)	3
16			3f (4)	4a (2)	6
Total A	**12**	**13**	**15**	**10**	**50**
SECTION B					
17		2d (8)	3a (4)		12
18			3a (8), 3h (3)		11
18.2			3e (3)		3
18.3			3d (4)		4
Total B	**0**	**8**	**22**	**0**	**30**
Total A+B	**12**	**21**	**37**	**10**	**80**

Question	Assessment Outcomes (2016)				TOTAL
	1	2	3	4	
SECTION A					
1	1c (1)				1
2	1a (1)				1
3	1a (1)				1
4			3a (1)		1
5		2c (1)			1
6			3c (3)		3
7	1a (3)				3
8.1		2a (2)			2
8.2		2a (2)			2
9.1			3d (1)		1
9.2		2a (2)			2
10		2f (1)		4a (2)	3
11		2d (3)			3
12			3g (2)		2
13		2e (3)	3a (3)		6
14.1	1c (2)				2
14.2	1c (4)				4
15.1				4a (3)	3
15.2				4a (3)	3
16			3f (4)	4a (2)	6
Total A	**12**	**13**	**15**	**10**	**50**
SECTION B					
17		2d (8)	3a (4)		12
18			3a (8), 3h (3)		11
18.2			3e (3)		3
18.3			3d (4)		4
Total B	**0**	**8**	**22**	**0**	**30**
Total A+B	**12**	**21**	**37**	**10**	**80**

Assessment Outcomes	Marks available in section A	Marks available in section B	Total Marks
AO1: Understand the different types of computer programming, languages and the common uses	8–12 marks 10–15%	2 × 15 37.5%	12
AO2: Analyse the tools and techniques for planning, design and development	12–15 marks 15–19%		21
AO3: Evaluate the key features and techniques used in computer programming	12–15 marks 15–19%		37
AO4: Demonstrate the principles of good program practice and user interface design	8–12 marks 10–15%		10
Total marks	**50 marks**	**30 marks**	**80**