OXFORD
INTERNATIONAL
AQA EXAMINATIONS

INTERNATIONAL GCSE COMPUTER SCIENCE

(9210)

Mark scheme

Paper 1: Programming

Specimen 2017

Copyright © 2017 Oxford International AQA Examinations and its licensors. All rights reserved.

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.

Copyright © 2017 Oxford International AQA Examinations and its licensors. All rights reserved.

International GCSE Computer Science mark scheme

How to mark

Aims

When you are marking your allocation of scripts your main aims should be to:

- recognise and identify the achievements of students
- where relevant, place students in the appropriate mark band and in the appropriate part of that mark band (high, low, middle) for **each** assessment objective
- record your judgements with brief notes, annotations and comments that are relevant to the mark scheme and make it clear to other associates how you have arrived at the numerical mark awarded for each assessment objective
- ensure comparability of assessment for all students, regardless of question or examiner.

Approach

It is important to be **open minded** and **positive** when marking scripts.

The specification recognises the variety of experiences and knowledge that students will have. It encourages them to study computer science in a way that is relevant to them. The questions have been designed to give them opportunities to discuss what they have found out about computer science. It is important to assess the quality of **what the student offers**.

Do not mark scripts based on the answer **you** would have written. The mark schemes have been composed to assess **quality of response** and not to identify expected items of knowledge.

Assessment Objectives

This component requires students to:

AO1: Demonstrate knowledge and understanding of the key concepts and principles of computer science.

AO2: Apply knowledge and understanding of the key concepts and principles of computer science.

AO3: Analyse problems in computational terms in order to develop and test programmed solutions and demonstrate an understanding of programming concepts.

Annotating scripts

You should write a summative comment at the end for each assessment objective. Indicate the marks for each assessment objective being tested at the end of the answer in the margin in sequence. It is vital that the way you arrive at a mark should be recorded on the script. This will help you with making accurate judgements and will assist any subsequent markers to identify how you are thinking. Please do not write negative comments about students' work or their alleged aptitudes.

COMPONENT NUMBER: **Paper 1**

COMPONENT NAME: **Programming**

STATUS:

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 student, if repeated, could result in the loss of more than one mark. The **DPT** label indicates that this mistake should result in a student losing only 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.

Section A

Question	Part	Marking guidance	Total marks
01	**1**	Integer variable stores whole numbers // stores numbers without a decimal/fractional part; Float/real variable stores numbers with a decimal/fractional part;	**2** **AO2=2**
01	**2**	Makes structure of program clearer // easier to understand; Subroutines can be easily re-used; If a subroutine is called more than once, it will reduce the memory required by a program; Subroutines can be tested independently; Subroutines can be developed by different members of a team; **Max 3**	**3** **AO2=3**
01	**3**	while; **A** while PlayerPositions[PlayerNum] < 99: **I** case	**1** **AO3=1**
01	**4**	2;	**1** **AO3=1**
01	**5**	Clear to programmer which values are being passed into/out of a subroutine // formal interface clear; Subroutine is more independent of rest of program; Easier to re-use subroutine in another program; **Max 2**	**2** **AO2=2**
01	**6**	Row // Column // Square;	**1** **AO3=1**
01	**7**	Local variables can be accessed/used only within subroutine they are declared in; Local variables use memory only when subroutine running; Global variables can be accessed anywhere in a program; Global variable use memory for the whole time the program is running; **Max 2**	**2** **AO2=2**

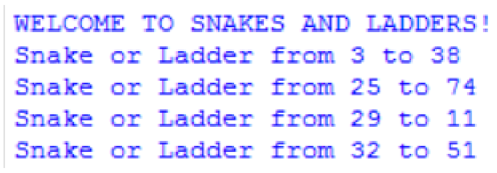
02	**1**	A ladder is created;
		from 72;
		to 92 // that moves the player forward 20 squares;
			3
			
			AO3=3

02	**2**	The square numbers run in different directions on odd and even numbered rows;	1 **AO3=1**
02	**3**	The game involves moving from one end of a row of a sequence of squares to another; Calculating which square/array index represents the end position of a move is easier (for this game) with a one-dimensional array.	**2** **AO3=2**
02	**4**	To switch between which player's turn it is; MOD is used so that when it is playerone's turn and 1 is added to the `PlayerNum` variable, the value 0 is returned instead of 2;	**2** **AO3=2**

Section B

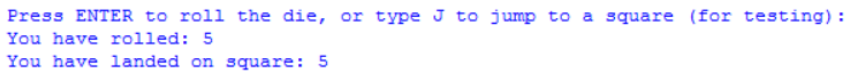
Question	Part	Marking guidance	Total marks

03	**1**	**1 mark**: Program displays correct message.
		A Minor typos and incorrect case.
		1 mark: Statement is at a position so that message is displayed at the correct time, ie it is before the call to ListSnakeAndLadders. The mark can be awarded even if the message displayed is incorrect.
		
		Example solution in Python 3
		
		`def PlayGame():`
		`  `**`print("WELCOME TO SNAKES AND LADDERS!")`**
		`  Board = CreateEmptyBoard()`
		`  Board = PlaceSnakesAndLadders(Board)`
		`  ListSnakesAndLadders(Board)`
		`  PlayerPositions = [0,0]`
		`  DisplayBoard(Board, PlayerPositions)`
		`  PlayerNum = 1`
		`  while PlayerPositions[PlayerNum] < 99:`
		`    PlayerNum = (PlayerNum + 1) % 2`
		`    ShowWhoseTurn(PlayerNum)`
		`    PlayerPositions[PlayerNum] = MakeMove(Board, PlayerPositions[PlayerNum])`
		`    if PlayerPositions[0]==PlayerPositions[1]:`
		`      print("Both players are on the same square")`
		`      PlayerToShift = random.randint(0, 1)`
		`      print("Player",PlayerToShift,"has been returned to square 0")`
		`      PlayerPositions[PlayerToShift] = 0`
		`    DisplayBoard(Board, PlayerPositions)`
		`  ShowWhoWon(PlayerNum)`
			2
			
			AO3=2

03	**2**	Screen capture(s) must match code from 03.1.
		
		1 mark: Correct message displayed before snakes and ladders are listed.
		A Incorrect message if it matches code in 03.1.
		
		
		`WELCOME TO SNAKES AND LADDERS!`
		`Snake or Ladder from 3 to 38`
		`Snake or Ladder from 25 to 74`
		`Snake or Ladder from 29 to 11`
		`Snake or Ladder from 32 to 51`
			1
			
			AO3=1

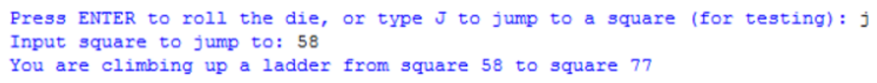
Question	Part	Marking guidance	Total marks

04	**1**	**1 mark**: Program displays correct text in message.
		1 mark: Program displays correct variable value in message.
		1 mark: Message displayed after existing message and before board redrawn.
		1 mark: If the move would have resulted in traversing a snake or ladder the program would display the final position after the traversal.
		A Minor typos and incorrect case.
		
		Example solution in Python 3
		
		def MakeMove(Board, CurrentPlayerPosition):
		  Option = input("Press ENTER to roll the die, or type J to jump to a square (for testing): ")
		  if Option in ['J', 'j']:
		    CurrentPlayerPosition = int(input("Input square to jump to: "))
		  else:
		    DieValue = random.randint(1, 6)
		    print("You have rolled:", DieValue)
		    CurrentPlayerPosition = CurrentPlayerPosition + DieValue
		  if CurrentPlayerPosition <= 99:
		    if Board[CurrentPlayerPosition] != 0:
		      print("You have landed on a snake or ladder")
		      CurrentPlayerPosition = CurrentPlayerPosition + Board[CurrentPlayerPosition]
		  print("You have landed on square:", CurrentPlayerPosition)
		  return CurrentPlayerPosition
			4
			
			AO3=4

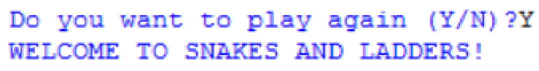
04	**2**	Screen capture(s) must match code from 04.1. **1 mark**: Message displayed that indicates position landed on. **I** If message is not the correct one or is displayed at the wrong point in the program.  Press ENTER to roll the die, or type J to jump to a square (for testing): You have rolled: 5 You have landed on square: 5	**1** **AO3=1**

Question	Part	Marking guidance	Total marks

05	**1**	**1 mark**: Correct logic to check if landed on ladder.
		1 mark: Message displayed to indicate that a ladder has been landed on, when appropriate.
		1 mark: Ladder message, displayed when appropriate, includes the start and end squares
		1 mark: Correct logic to check if landed on snake.
		1 mark: Message displayed to indicate that a snake has been landed on, when appropriate.
		1 mark: Snake message, displayed when appropriate, includes the start and end squares
		A "Correct logic" for one of the conditions could just be an else statement if the corresponding if statement has the correct condition
		I Message does not exactly match example in question paper, it is the content that is being marked in this part of the question.
		Example solution in Python 3
		
		def PlayGame():
		  print("WELCOME TO SNAKES AND LADDERS!")
		  Board = CreateEmptyBoard()
		  Board = PlaceSnakesAndLadders(Board)
		  ListSnakesAndLadders(Board)
		  PlayerPosition = 0
		  DisplayBoard(Board, PlayerPosition)
		  while PlayerPosition < 99:
		    PlayerPosition = MakeMove(PlayerPosition)
		    if PlayerPosition <= 99:
		      if Board[PlayerPosition] != 0:
		        if Board[PlayerPosition] > 0:
		          print("You are climbing up a ladder from square", PlayerPosition, "to square", PlayerPosition + Board[PlayerPosition])
		        else:
		          print("You are sliding down a snake from square", PlayerPosition, "to square", PlayerPosition + Board[PlayerPosition])
		        PlayerPosition = PlayerPosition + Board[PlayerPosition]
		        Pause = input()
		    print("You have landed on square:", PlayerPosition)
		    DisplayBoard(Board, PlayerPosition)
			6
			
			AO3=6

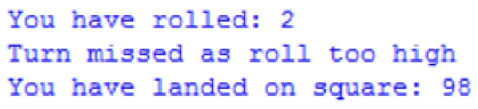
05	**2**	Screen captures must match code from 05.1. **1 mark**: Correct selections (J, 58) input. **1 mark**: Message displayed to indicate that player moving down snake from 58 to 19.  Press ENTER to roll the die, or type J to jump to a square (for testing): j Input square to jump to: 58 You are climbing up a ladder from square 58 to square 77	**2** **AO3=2**

Copyright © 2017 Oxford International AQA Examinations and its licensors. All rights reserved.

Question	Part	Marking guidance	Total marks
06	**1**	**1 mark**: Use of a loop type which allows indefinite iteration. **1 mark**: Condition that controls loop iteration will iterate under the correct circumstances. **1 mark**: Call to PlayGame is within loop. **1 mark**: Correct prompt displayed within loop. **A** Minor typos in prompt. **1 mark**: Variable with correct type and identifier created (if language does not require declaration then award mark if initial value assigned is of correct type and identifier correct). **1 mark**: User input stored into variable within loop. **1 mark**: No code inside loop that should not be. **Example solution in Python 3** def Main(): random.seed() **PlayAgain = "Y"** **while PlayAgain == "Y":** PlayGame() **PlayAgain = input("Do you want to play again (Y/N)**	**7** **AO3=7**
06	**2**	Screen capture(s) must match code from 06.1. **1 mark**: Screen capture shows game restarting after Y input at prompt. **A** Display of welcome message or start of list of snakes and ladders as evidence of game restarting.  Do you want to play again (Y/N)?Y WELCOME TO SNAKES AND LADDERS!	**1** **AO3=1**

07	**1**	**1 mark**: Use of if statement.
		1 mark: If statement in correct place with condition that would correctly identify if move would go past square 99.
		1 mark: Player’s position is not changed if would go past square 99.
		1 mark: Correct message displayed if would go past square 99.
		A Minor typos in message.
		R Do not award marks for position not changing or message being displayed if this would also happen in incorrect circumstances.
		
		Example solution in Python 3
		
		def MakeMove(Board, CurrentPlayerPosition):
		  # print()
		  Option = input("Press ENTER to roll the die, or type J to jump to a square (for testing): ")
		  if Option in ['J', 'j']:
		    CurrentPlayerPosition = int(input("Input square to jump to: "))
		  else:
		    DieValue = random.randint(1, 6)
		    print("You have rolled:", DieValue)
		    CurrentPlayerPosition = CurrentPlayerPosition + DieValue
		  if CurrentPlayerPosition > 99:
		    CurrentPlayerPosition = CurrentPlayerPosition - DieValue
		    print("Turn missed as roll too high")
			4
			
			AO3=4

Question	Part	Marking guidance	Total marks

07	**2**	Screen capture(s) must match code from 07.1.
		
		1 mark: Screen capture shows message used in 07.1 displayed if roll goes past square 99.
		1 mark: Screen capture shows that player has not moved.
		
		
		`You have rolled: 2`
		`Turn missed as roll too high`
		`You have landed on square: 98`
			2
			
			AO3=2

Section C

Question	Part	Marking guidance	Total marks
08	**1**	**These four marks are available only if new subroutine created:** **1 mark**: New subroutine created and called from somewhere in `PlayGame`. **1 mark**: Subroutine has correct name (`MoveBackIfSameSquare`). **1 mark**: Player positions passed to subroutine as parameters. **1 mark**: If player positions are updated within subroutine they are returned to calling subroutine using the `return` command. **These eight marks are available whether or not new subroutine created:** **1 mark**: Call to subroutine is made after the call to `MakeMove` or if question tackled without creating new subroutine the code to carry out the task is after the call to `MakeMove`. **A** This mark can be awarded whether the code works or not. **I.** If the code is before or after the call to `DisplayMove`. **1 mark**: If statement correctly tests if both players are on same square. **1 mark**: Message explains that both players are on same square, if they are. **1 mark**: Player to move back to start is selected at random, under correct circumstances. **1 mark**: Equal probability of either player being moved back to start. **1 mark**: A player is moved back to the start (this being the correct player if selected at random, or any player if that aspect of the question not tackled), under correct circumstances. **1 mark**: A player remains on the same square – only award this mark if the other player has been moved back to square 0. **1 mark**: Message explains which player has been moved back to start. **Example solution in Python 3** **def MoveBackIfSameSquare(PlayerPositions):** **if PlayerPositions[0]==PlayerPositions[1]:**	**12**

Question	Part	Marking guidance	Total marks

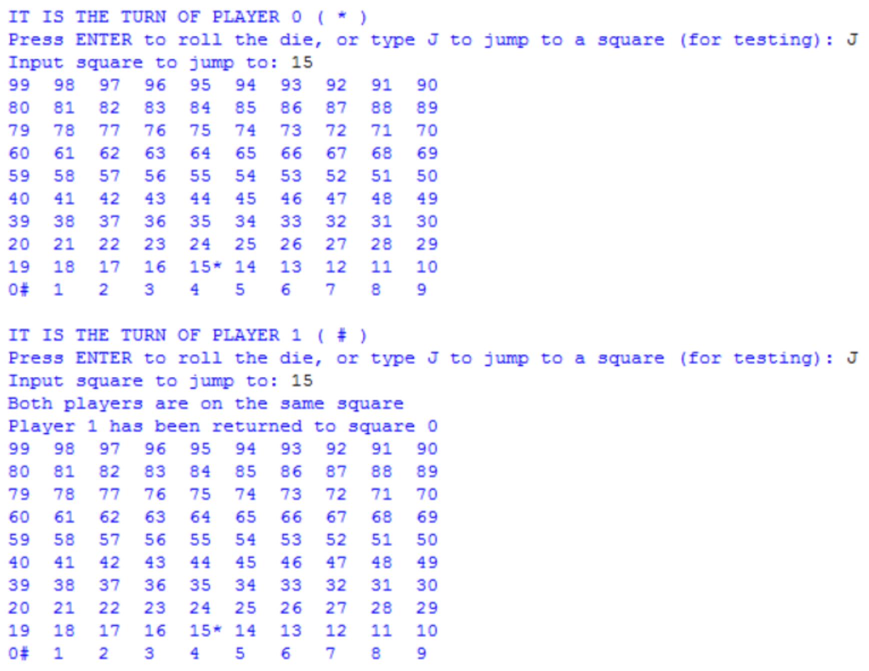
```
    print("Both players are on the same square")
    PlayerToShift = random.randint(0, 1)
    print("Player",PlayerToShift,"has been returned to square 0")
    PlayerPositions[PlayerToShift] = 0
  return PlayerPositions

def PlayGame():
  print("WELCOME TO SNAKES AND LADDERS!")
  Board = CreateEmptyBoard()
  Board = PlaceSnakesAndLadders(Board)
  ListSnakesAndLadders(Board)
  PlayerPositions = [0,0]
  DisplayBoard(Board, PlayerPositions)
  PlayerNum = 1
  while PlayerPositions[PlayerNum] < 99:
    PlayerNum = (PlayerNum + 1) % 2
    ShowWhoseTurn(PlayerNum)
    PlayerPositions[PlayerNum] = MakeMove(Board,
PlayerPositions[PlayerNum])
    PlayerPositions=MoveBackIfSameSquare(PlayerPositions)
    DisplayBoard(Board, PlayerPositions)
  ShowWhoWon(PlayerNum)
```

AO3=12

Question	Part	Marking guidance	Total marks
08	**2**	Screen capture(s) must match code from 08.1.	**2**

1 mark: Screen capture shows both players have landed on the same square.
2 marks: Screen capture shows that one player has been returned to square 0 and the other has remained where they landed.
A If a square other than 15 has been used for testing.



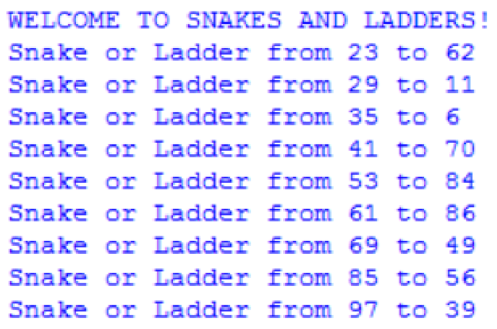
```
IT IS THE TURN OF PLAYER 0 ( * )
Press ENTER to roll the die, or type J to jump to a square (for testing): J
Input square to jump to: 15
99  98  97  96  95  94  93  92  91  90
80  81  82  83  84  85  86  87  88  89
79  78  77  76  75  74  73  72  71  70
60  61  62  63  64  65  66  67  68  69
59  58  57  56  55  54  53  52  51  50
40  41  42  43  44  45  46  47  48  49
39  38  37  36  35  34  33  32  31  30
20  21  22  23  24  25  26  27  28  29
19  18  17  16  15* 14  13  12  11  10
0#  1   2   3   4   5   6   7   8   9

IT IS THE TURN OF PLAYER 1 ( # )
Press ENTER to roll the die, or type J to jump to a square (for testing): J
Input square to jump to: 15
Both players are on the same square
Player 1 has been returned to square 0
99  98  97  96  95  94  93  92  91  90
80  81  82  83  84  85  86  87  88  89
79  78  77  76  75  74  73  72  71  70
60  61  62  63  64  65  66  67  68  69
59  58  57  56  55  54  53  52  51  50
40  41  42  43  44  45  46  47  48  49
39  38  37  36  35  34  33  32  31  30
20  21  22  23  24  25  26  27  28  29
19  18  17  16  15* 14  13  12  11  10
0#  1   2   3   4   5   6   7   8   9
```

AO3=2

Question	Part	Marking guidance	Total marks

09	**1**	**1 mark**: A loop is created that will execute five times.
		1 mark: All commands to place a ladder are within a loop.
		1 mark: Random number generated for ladder start position and stored into a variable.
		1 mark: Correct range (1..89) for ladder start random number.
		1 mark: Check made to see if ladder start position already in use.
		1 mark: If start position already in use a new position is picked.
		1 mark: Re-picking of position will repeat until an empty position is found.
		1 mark: Random number generated for ladder length.
		1 mark: Correct range (10..40) for ladder length random number.
		1 mark: Check made to see if ladder end position is off board.
		1 mark: If end position already in use a new length is picked or a new start position is picked.
		1 mark: Re-picking of length or start position will repeat until ladder ends on board.
		1 mark: Value at correct index in array is updated.
		1 mark: Value stored at correct index is the length of the ladder.
		
		Example solution in Python 3
		
		def PlaceSnakesAndLadders(Board):
		  for Ladder in range(5):
		    LadderStart = random.randint(1, 89)
		    while Board[LadderStart] != 0:
		      LadderStart = random.randint(1, 89)
		    LadderLength = random.randint(10, 40)
		    while LadderStart + LadderLength > 99:
		      LadderLength = random.randint(10, 40)
		    Board[LadderStart] = LadderLength
		  Board[29] = -18
		  Board[35] = -32
		  Board[69] = -20
		  Board[85] = -29
		  Board[97] = -38
		  return Board
			14
			
			AO3=14

09	**2**	Screen capture(s) must match code from 09.1. **1 mark**: Screen capture shows at least one legal ladder has been generated **OR 2 marks**: Screen capture shows exactly five legal ladder positions have been generated.  WELCOME TO SNAKES AND LADDERS! Snake or Ladder from 23 to 62 Snake or Ladder from 29 to 11 Snake or Ladder from 35 to 6 Snake or Ladder from 41 to 70 Snake or Ladder from 53 to 84 Snake or Ladder from 61 to 86 Snake or Ladder from 69 to 49 Snake or Ladder from 85 to 56 Snake or Ladder from 97 to 39	**2** **AO3=2**

GET HELP AND SUPPORT

Visit our website for information, guidance, support and resources at **oxfordaqaexams.org.uk**

You can contact the computer science subject team directly;

E: **computerscience@oxfordaqaexams.org.uk**

OXFORD

INTERNATIONAL
AQA EXAMINATIONS

OXFORD INTERNATIONAL AQA EXAMINATIONS
LINACRE HOUSE, JORDAN HILL, OXFORD, OX2 8TA
UNITED KINGDOM
enquiries@oxfordaqaexams.org.uk
oxfordaqaexams.org.uk

Permission to reproduce all copyright material has been applied for. In some cases, efforts to contact copyright-holders may have been unsuccessful and Oxford International AQA Examinations will be happy to rectify any omissions of acknowledgements. If you have any queries please contact the Copyright Team, AQA, Stag Hill House, Guildford, GU2 7XJ.

Copyright © 2017 Oxford International AQA Examinations and its licensors. All rights reserved.