General Certificate of Education
Advanced Subsidiary Examination
June 2010

Computing COMP1/PM/PH

Unit 1 Problem Solving, Programming, Data Representation and Practical Exercise

Preliminary Material

A copy of this Preliminary Material will appear on the GCE Computing front page on the AQA Website on Monday 1 March 2010. It will include the Skeleton Program as issued by AQA.

A copy of the Preliminary Material, including any amendments to the Skeleton Program made by the Head of Computing and approved by AQA, must be given to candidates by their centre on or after Thursday 1 April 2010.

Information for Candidates

- This Preliminary Material comprises
  - Instructions to Candidates and a
  - Skeleton Program for **PHP** which your teacher will supply.

  You must **only** use the version of the Skeleton Program supplied by your teacher.
- Candidates are advised to familiarise themselves with the Preliminary Material before the examination.
- You must **not** take any copy of the Preliminary Material or any other material into the examination room. A second copy of the Preliminary Material will be given to you in the examination. Your teacher will also provide you with access electronically to the Skeleton Program at the start of the examination.

Instructions to Candidates

The question paper is divided into four sections and a recommendation is given to candidates as to how long to spend on each section. Below are the recommended timings for the 2010 examination.

Section A
You are advised to spend no more than **35 minutes** on this section.
Questions will examine the specification content **not** specific to the *Preliminary Material*.

Section B
You are advised to spend no more than **20 minutes** on this section.
You will be asked to create a new program **not** related to the *Preliminary Material* or Skeleton Program.

Section C
You are advised to spend no more than **15 minutes** on this section.
Questions will refer to the *Preliminary Material* and the Skeleton Program, but will not require programming.

Section D
You are advised to spend no more than **50 minutes** on this section.
Questions will use the Skeleton Program and the *Preliminary Material*.

Electronic Answer Document

Answers to all questions must be written into the word processed document made available to you at the start of the examination and referred to in the question paper as the Electronic Answer Document.

Preparation for the Examination

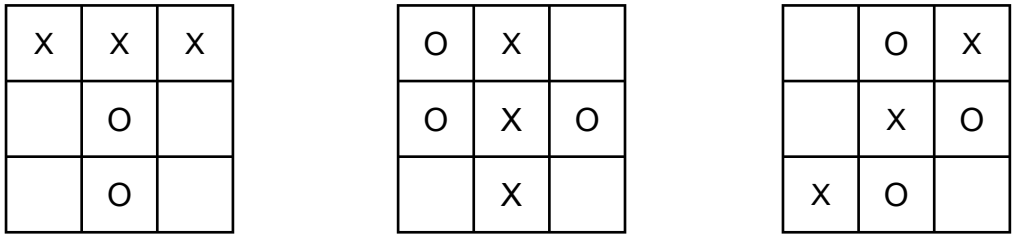
For your programming language you should ensure that you are familiar with this *Preliminary Material***,** including the Skeleton Program.

'Noughts and Crosses Game'

The **Skeleton Program** in this Preliminary Material is for a program based on the two-player game of 'Noughts and Crosses'.

When playing Noughts and Crosses players take turns to place a symbol on the board. The board is a 3x3 grid, which is initially empty. One player uses the symbol 'X', the other the symbol 'O'. Players are only allowed to place a symbol in an empty position on the board. The aim of the game is to be the first person to get three of their symbols in a line – a line of symbols can be either in a row, in a column or along a diagonal. Three winning board positions for the player using the symbol 'X' are shown in **Figure 1**.

Figure 1



In the **Skeleton Program** players use row and column coordinates to specify where they wish their 'X' or 'O' symbol to be placed. First, they specify the x coordinate – this indicates which column they want; then they specify the y coordinate – this indicates which row they want. **Figure 2** shows part of a game. The first board, (a), is the initial position, the second board, (b), shows what would happen if the 'X' player enters coordinates of (1,3).

Figure 2

(a)

	1	2	3
1			
2			
3			

(b)

	1	2	3
1			
2			
3	X		

The game finishes when a player gets three of their symbol in a line or there are no empty cells on the grid. If the grid is full but neither player has a line of three symbols then the game is drawn. After the last move the result of the game is displayed.

Outline Design

The Structured English description of an algorithm for playing one game is as follows.

```
SET number of moves TO 0
Clear Board
Display Board
IF starting symbol IS EQUAL TO player one's symbol
  THEN OUTPUT message saying player one is going to make a move first
  ELSE OUTPUT message saying player two is going to make a move first
ENDIF
SET current symbol TO starting symbol
DO
  DO
    Get move coordinates
    Check if move is valid
    IF NOT valid move
      THEN OUTPUT invalid move message
    ENDIF
  UNTIL valid move
  Make move
  Display Board
  Check if game has been won
  INCREMENT number of moves BY 1
  IF NOT game has been won
    THEN
      IF number of moves IS EQUAL TO 9
        THEN game has been drawn
        ELSE change current symbol
      ENDIF
  ENDIF
UNTIL game has been won OR game has been drawn
OUTPUT result of game
```

Variables

The main variables used are as follows.

Identifier	Data Type	Purpose
`NoOfMoves`	`Integer`	Used to keep track of how many moves have been made in the game so far ie how many positions in the board have a symbol in them
`CurrentSymbol`	`Char`	Used to indicate the symbol being used by the player whose turn it currently is - '`X`' or '`O`'
`Board`	`Array[1..3,1..3] Of Char`	A two-dimensional array used to indicate which symbol, if any, is currently in each position on the 3x3 grid.

Notes

The programming language used to code the game will determine the letter case for each identifier, and so may not match exactly the identifiers shown in the table above.

Your chosen programming language may use arrays with a lower bound value of `0`. If so, array cells with indices of `0` are not used.

END OF INSTRUCTIONS TO CANDIDATES

```php
<?php

/* Skeleton program code for the AQA COMP1 Summer 2010 examination
 * This code should be used in conjunction with the Preliminary
 * materials. Written by the AQA COMP1 Programmer Team developed in
 * the PHP version 5.2.8 programming environment
*/

function DisplayBoard($Board){
      fwrite(STDOUT, "  | 1 2 3 \n");
      fwrite(STDOUT, "--|-------\n");
      for ($Row = 1; $Row <= 3; $Row++) {
            fwrite(STDOUT, $Row . " | ");
            for ($Column = 1; $Column <= 3; $Column++) {
                  fwrite(STDOUT, $Board[$Column][$Row] . " ");
            }
            fwrite(STDOUT, "\n");
      }
      fwrite(STDOUT, "\n");
} // end of DisplayBoard

function ClearBoard(&$Board) {
      for ($Row = 1; $Row <= 3; $Row++) {
            for ($Column = 1; $Column <= 3; $Column++)
                  $Board[$Column][$Row] = ' ';
      }
} // end of ClearBoard

function GetMoveCoordinates(&$XCoordinate, &$YCoordinate){
      fwrite(STDOUT, "Enter x coordinate: ");
      $XCoordinate = intval(trim(fgets(STDIN)));
      fwrite(STDOUT, "Enter y coordinate: ");
      $YCoordinate = intval(trim(fgets(STDIN)));
```

```
} // end of GetMoveCoordinates

function CheckValidMove($XCoordinate, $YCoordinate, $Board){
      // check X coordinate is valid
      $ValidMove = true;
      if ($XCoordinate < 1 || $XCoordinate > 3) {
            $ValidMove = false;
      }
      return $ValidMove;
} // end of CheckValidMove

function CheckXOrOHasWon($Board){
      $XOrOHasWon = false;
      for ($Column = 1; $Column <= 3; $Column++) {
            if ($Board[$Column][1] == $Board[$Column][2]
                && $Board[$Column][2] == $Board[$Column][3]
                  && $Board[$Column][2] != ' ') {
                  $XOrOHasWon = true;
            }
      }
      for ($Row = 1; $Row <= 3; $Row++) {
            if ($Board[1][$Row] == $Board[2][$Row]
                  && $Board[2][$Row] == $Board[3][$Row]
                  && $Board[2][$Row] != ' ') {
                  $XOrOHasWon = true;
            }
      }
      return $XOrOHasWon;
} //end of CheckXOrOHasWon

function GetWhoStarts(){
      $RandomNumber = rand(1, 100);
      if ($RandomNumber % 2 == 0)
```

```
            return 'X';
      else
            return 'O';
} // end of GetWhoStarts

// main program block starts here
$Board = array(array());
$PlayerOneName = "";
$PlayerTwoName = "";
$PlayerOneScore = 0.0;
$PlayerTwoScore = 0.0;
$XCoord = 0;
$YCoord = 0;
$ValidMove = false;
$NoOfMoves = 0;
$GameHasBeenWon = false;
$GameHasBeenDrawn = false;
$CurrentSymbol = ' ';
$StartSymbol = ' ';
$PlayerOneSymbol = ' ';
$PlayerTwoSymbol = ' ';
$Answer = ' ';

fwrite(STDOUT, "What is the name of player one? ");
$PlayerOneName = trim(fgets(STDIN));
fwrite(STDOUT, "What is the name of player two? ");
$PlayerTwoName = trim(fgets(STDIN));
fwrite(STDOUT, "\n");
// Choose player one's symbol
do {
      fwrite(STDOUT, $PlayerOneName . " what symbol do you wish to use, X or O? ");
      $PlayerOneSymbol = trim(fgets(STDIN));
      if ($PlayerOneSymbol != 'X' && $PlayerOneSymbol != 'O') {
```

```
            fwrite(STDOUT, "Symbol to play must be uppercase X or O\n");
            fwrite(STDOUT, "\n");
      }
} while($PlayerOneSymbol != 'X' && $PlayerOneSymbol != 'O');
if ($PlayerOneSymbol == 'X')
      $PlayerTwoSymbol = 'O';
else
      $PlayerTwoSymbol = 'X';
$StartSymbol = GetWhoStarts();
do {        // Play a game
      $NoOfMoves = 0;
      $GameHasBeenDrawn = false;
      $GameHasBeenWon = false;
      ClearBoard($Board);
      fwrite(STDOUT, "\n");
      DisplayBoard($Board);
      if ($StartSymbol == $PlayerOneSymbol)
            fwrite(STDOUT, $PlayerOneName . " starts playing " . $StartSymbol . "\n");
      else
            fwrite(STDOUT, $PlayerTwoName . " starts playing " . $StartSymbol . "\n");
      fwrite(STDOUT, "\n");
      $CurrentSymbol = $StartSymbol;
      do {        // Play until a player wins or the game is drawn
            do {        // Get a valid move
                  GetMoveCoordinates($XCoord, $YCoord);
                  $ValidMove = CheckValidMove($XCoord, $YCoord, $Board);
                  if (!$ValidMove)
                        fwrite(STDOUT, "Coordinates invalid - please try again!\n");
            } while(!$ValidMove);
            $Board[$XCoord][$YCoord] = $CurrentSymbol;
            DisplayBoard($Board);
            $GameHasBeenWon = CheckXOrOHasWon($Board);
            $NoOfMoves++;
```

```
      if (!$GameHasBeenWon) {
            if ($NoOfMoves == 9)  // Check if maximum number of
                                  // allowed moves has been reached
                  $GameHasBeenDrawn = true;
            else {
                  if ($CurrentSymbol == 'X')
                        $CurrentSymbol = 'O';
                  else
                        $CurrentSymbol = 'X';
            }
      }
} while(!$GameHasBeenWon && !$GameHasBeenDrawn);
if ($GameHasBeenWon) {   // Update scores and display result
      if ($PlayerOneSymbol == $CurrentSymbol) {
            fwrite(STDOUT, $PlayerOneName . " congratulations, you win!");
            $PlayerOneScore = $PlayerOneScore + 1;
      }
      else {
            fwrite(STDOUT, $PlayerTwoName . " congratulations, you win!");
            $PlayerTwoScore = $PlayerTwoScore + 1;
      }
}
else
      fwrite(STDOUT, "A draw this time!");
fwrite(STDOUT, "\n");
fwrite(STDOUT, $PlayerOneName . " your score is " . $PlayerOneScore . "\n");
fwrite(STDOUT, $PlayerTwoName . " your score is " . $PlayerTwoScore . "\n");
fwrite(STDOUT, "\n");
if ($StartSymbol == $PlayerOneSymbol)
      $StartSymbol = $PlayerTwoSymbol;
else
      $StartSymbol = $PlayerOneSymbol;
fwrite(STDOUT, "Another game Y/N? ");
```

```
      $Answer = trim(fgets(STDIN));
} while($Answer != 'N' && $Answer != 'n');

?>
```