Chapter 12 Stepwise Refinement and Structure Charts

Chapter 14 Structured Programming

Answers to coursebook questions and tasks

Syllabus sections covered: 2.1 (2.1.1 and 2.1.2), 2.3 (2.3.6)

It is suggested that Chapter 12 is worked through in parallel with Chapter 14.

Task 12.01

```
01       // Set up initial values
01.1     INPUT symbol
01.2     // Input Max Number Of Symbols
01.2.1   REPEAT
01.2.2      INPUT MaxNumberOfSymbols
01.2.3   UNTIL MaxNumberOfSymbols MOD 2 = 1
01.3     NumberOfLeadingSpaces ← (MaxNumberOfSymbols - 1) / 2
01.4     NumberOfSymbols ← 1
01.5     NumberOfMiddleSpaces ← -1
02       REPEAT
03          // Output number of leading spaces
03.1        FOR i ← 1 TO NumberOfLeadingSpaces
03.2           OUTPUT Space           // without moving to next
line
03.3        ENDFOR
04          // Output symbol, middle spaces, symbol
04.01       IF NumberOfSymbols = 1   // top of pyramid
04.02          THEN
04.03             OUTPUT Symbol
04.04          ELSE
04.05             IF NumberOfSymbols < MaxNumberOfSymbols
04.06                THEN
04.07                   OUTPUT Symbol
04.08                   FOR i ← 1 TO NumberOfMiddleSpaces
04.09                      OUTPUT Space       // don't move to
next line
04.10                   ENDFOR
04.11                   OUTPUT Symbol
04.12                ELSE    // output the final line
04.13                   FOR i ← 1 TO NumberOfSymbols
04.14                      OUTPUT Symbol      // don't move to
next line
04.15                   ENDFOR
04.16             ENDIF
04.17       ENDIF
04.18       OUTPUT Newline             // move to the next line
05          // Adjust values for next row
```

```
      NumberOfLeadingSpaces ← NumberOfLeadingSpaces - 1
      NumberOfMiddleSpaces ← NumberOfMiddleSpaces + 2
      NumberOfSymbols ← NumberOfSymbols + 2
   UNTIL NumberOfSymbols > MaxNumberOfSymbols
```

A better solution is to treat the tip and the base of the triangle separately and use the REPEAT loop for the other lines.

Task 12.02

```
CALL SetValues
CALL OutputTopRow
CALL AdjustValuesForNextRow
REPEAT
   CALL OutputLeadingSpaces
   CALL OutputRow
   CALL AdjustValuesForNextRow
UNTIL NumberOfSymbols = MaxNumberOfSymbols
CALL OutputBaseRow

PROCEDURE SetValues
   INPUT symbol
   // Input Max Number Of Symbols
   REPEAT
      INPUT MaxNumberOfSymbols
   UNTIL MaxNumberOfSymbols MOD 2 = 1
   NumberOfLeadingSpaces ← (MaxNumberOfSymbols - 1) / 2
   NumberOfSymbols ← 1
   NumberOfMiddleSpaces ← -1
ENDPROCEDURE

PROCEDURE OutputTopRow
   CALL OutputLeadingSpaces
   OUTPUT Symbol
   OUTPUT Newline
ENDPROCEDURE

PROCEDURE AdjustValuesForNextRow
   NumberOfLeadingSpaces ← NumberOfLeadingSpaces - 1
   NumberOfMiddleSpaces ← NumberOfMiddleSpaces + 2
   NumberOfSymbols ← NumberOfSymbols + 2
ENDPROCEDURE

PROCEDURE OutputLeadingSpaces
   FOR i ← 1 TO NumberOfLeadingSpaces
      OUTPUT Space            // without moving to next line
   ENDFOR
ENDPROCEDURE
```

```
PROCEDURE OutputRow
   OUTPUT Symbol
   FOR i ← 1 TO NumberOfMiddleSpaces
      OUTPUT Space       // don’t move to next line
   ENDFOR
   OUTPUT Symbol
   OUTPUT Newline             // move to the next line
ENDPROCEDURE

PROCEDURE OutputBaseRow
   FOR i ← 1 TO NumberOfSymbols
      OUTPUT Symbol
   ENDFOR
   OUTPUT Newline
ENDPROCEDURE
```

Task 12.03

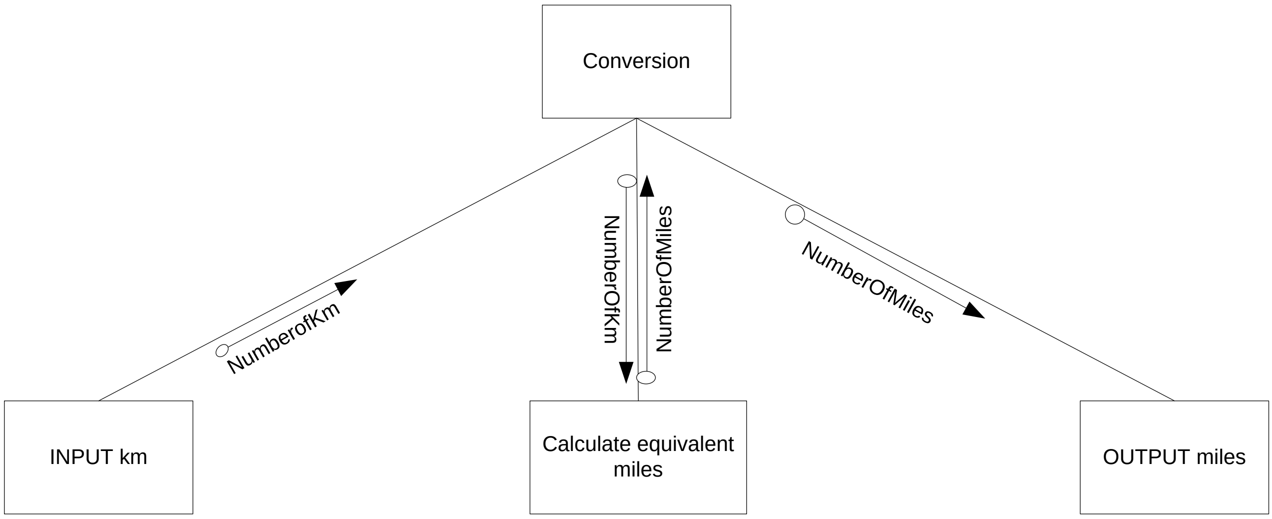


Figure 12.01

Task 12.04

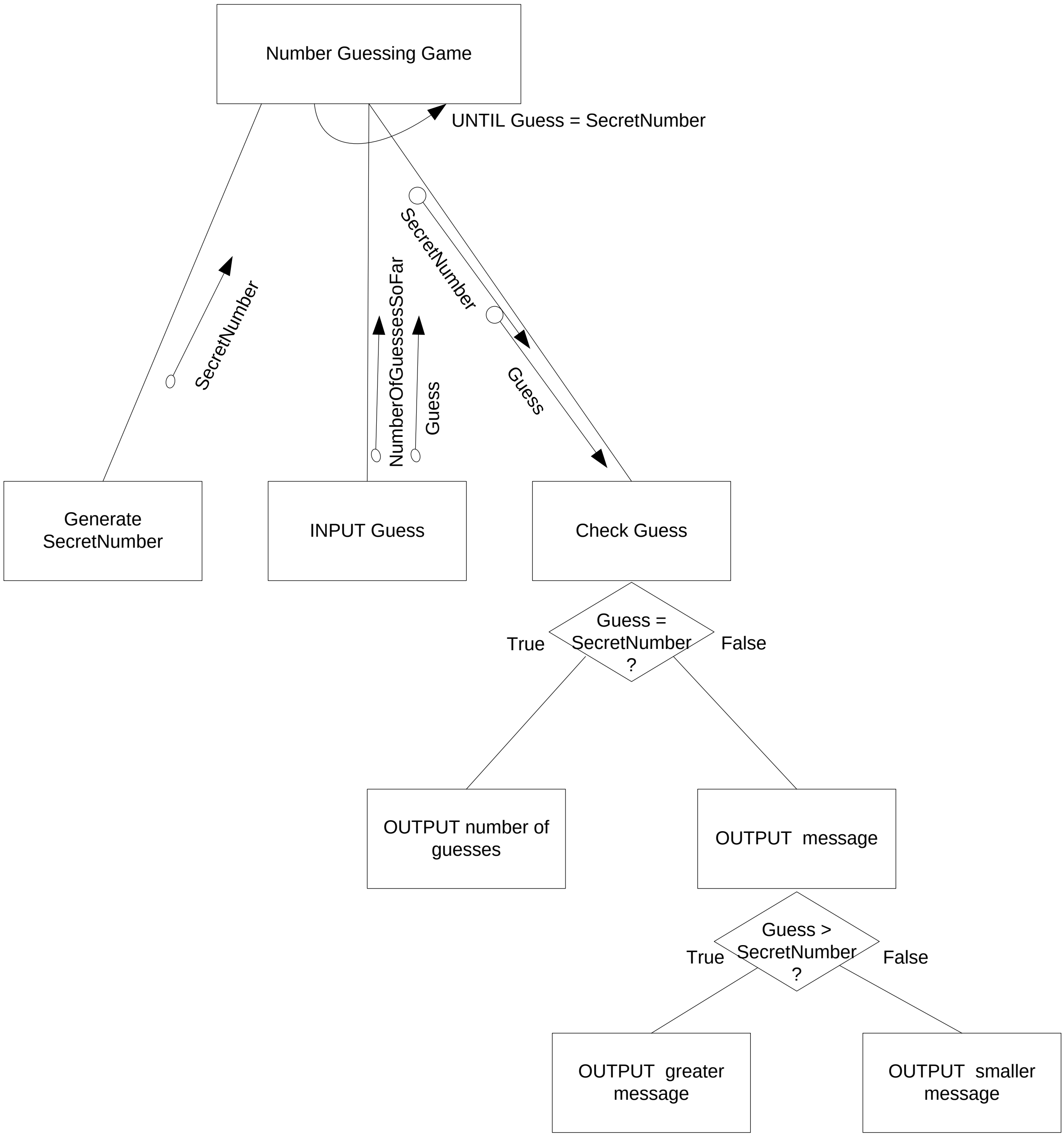


Figure 12.02

Task 12.05

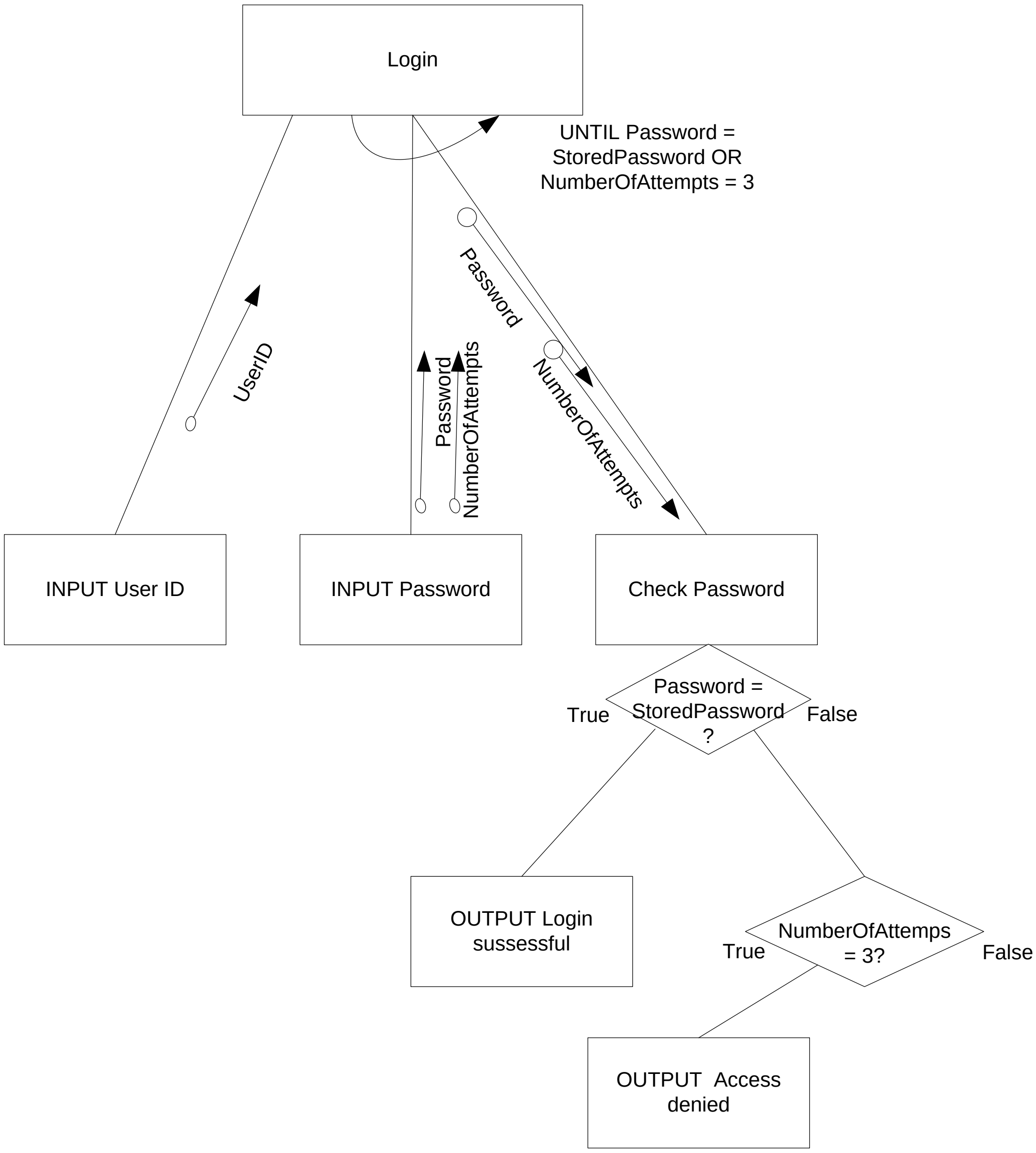


Figure 12.03

Exam-style questions in Chapter 12

1 a A: Initialise Tally

B: Generate random number

C: RandomNumber

D: Tally

E: Tally

b Pseudocode for random number tally

```
DECLARE Tally : ARRAY[1..20] OF INTEGER
CALL InitialiseTally(Tally)
FOR Count ← 1 TO NumberOfTests
```

```
   RandomNumber ← GenerateRandomNumber(20)
   CALL UpdateTally(RandomNumber, Tally)
ENDFOR
CALL OutputTally(Tally)
```

2 a Step-wise refinement

b

```
PROCEDURE SetUpEmptyGrid
   FOR i ← 1 TO 8
      FOR j ← 1 TO 8
         Grid[i, j] ← 0
      ENDFOR
   ENDFOR
ENDPROCEDURE

PROCEDURE RandomlyDistributeCards
   FOR Number ← 1 TO 32
      CALL GetEmptyGridPosition
      Grid[x, y] ← Number
      CALL GetEmptyGridPosition
      Grid[x, y] ← Number
   ENDFOR
ENDPROCEDURE

PROCEDURE GetEmptyGridPosition
   REPEAT
      x ← RandomNumber(1,8)
      y ← RandomNumber(1,8)
   UNTIL Grid[x, y] = 0 // find a grid position without a card
ENDPROCEDURE

PROCEDURE SetUpPlayers
   Points[1] ← 0
   Points[2] ← 0
   ThisPlayer ← 1
ENDPROCEDURE

PROCEDURE GetPlayerCoordinates
   REPEAT
      INPUT x1, y1
   UNTIL Grid[x1, y1] > 0 // check grid position has a card
   CALL DisplayGrid
   REPEAT
      INPUT x2, y2
           // check grid position has a card and is not the
same as first card
   UNTIL (Grid[x2, y2] > 0) AND ((x1 <> x2) OR (y1 <> y2))
ENDPROCEDURE

PROCEDURE DisplayGrid
   FOR i ← 1 TO 8
      FOR j ← 1 TO 8
         IF (I = x1) AND (j = y1) // it is the chosen card
            THEN
               OUTPUT Grid[i, j]
            ELSE
               IF Grid[I, j] = 0 // the card in this position
has been removed
                  THEN
                     OUTPUT '   '
                  ELSE   // back of card to be shown as ' ? '
                     OUTPUT ' ? '
               ENDIF
         ENDIF
      ENDFOR
   ENDFOR
ENDPROCEDURE

PROCEDURE TestForMatch
   IF Grid[x1, y1] = Grid[x2, y2]
      THEN
         // match found, remove cards
         Grid[x1, y1] ← 0
         Grid[x2, y2] ← 0
         // increment points
         Points[ThisPlayer] ← Points[ThisPlayer] + 1
      ELSE
         CALL SwapPlayers
   ENDIF
ENDPROCEDURE

PROCEDURE SwapPlayers
   IF ThisPlayer = 1
      THEN
         ThisPlayer ← 2
      ELSE
         ThisPlayer ← 1
   ENDIF
ENDPROCEDURE

PROCEDURE TestForEndGame
   IF Points[1] + Points[2] = 32
      THEN
         GameEnd ← TRUE
   ENDIF
ENDPROCEDURE

PROCEDURE OutputResults
   OUTPUT Points[1]
   OUTPUT Points[2]
ENDPROCEDURE
```

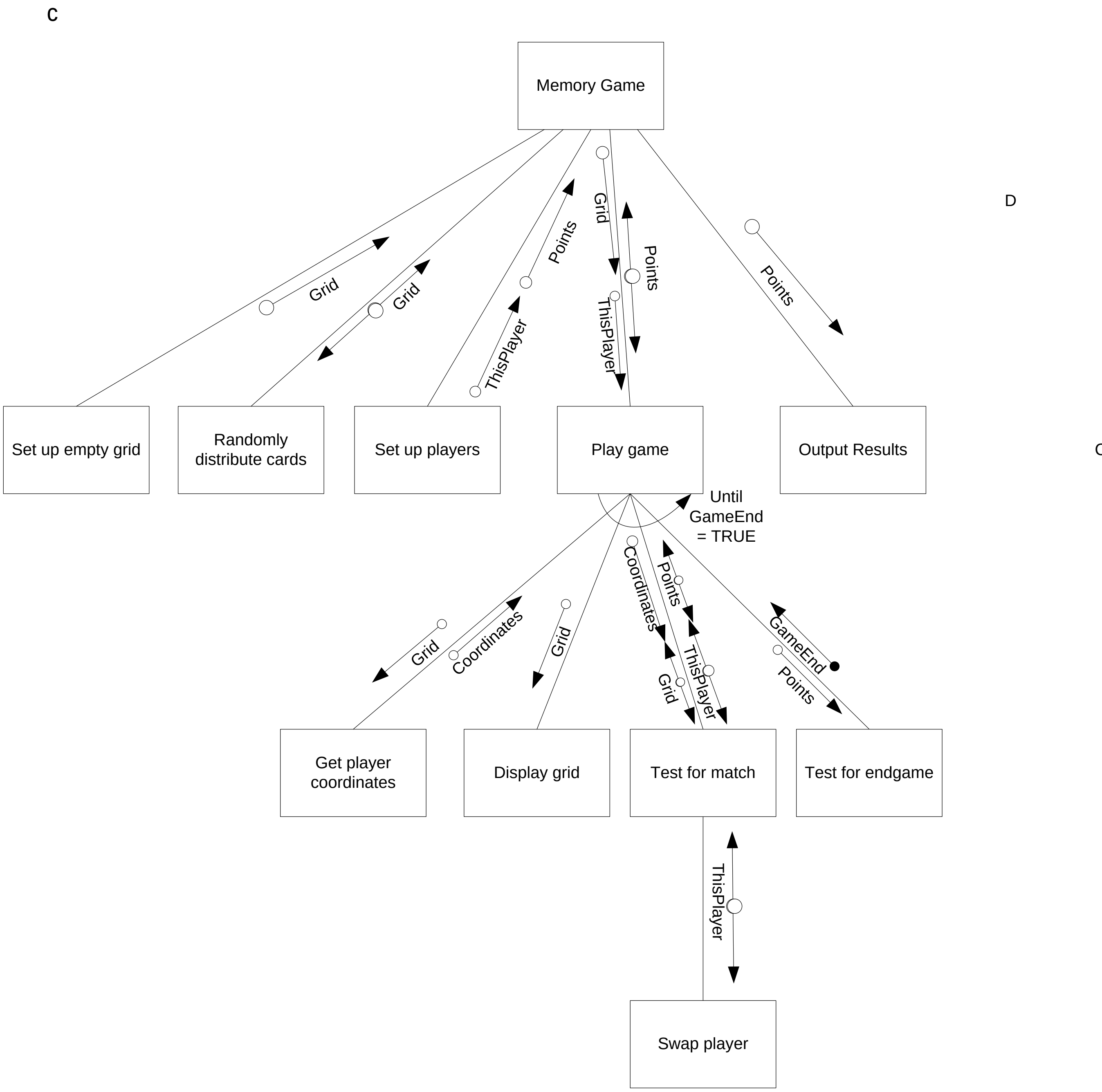


Figure 12.04

Task 14.01

```
PROCEDURE SetValues
   INPUT Symbol
   CALL InputMaxNumberOfSymbols // need to ensure it is an odd
number
   NumberOfSpaces ← (MaxNumberOfSymbols -1) / 2
   NumberOfSymbols ← 1
ENDPROCEDURE

PROCEDURE InputMaxNumberOfSymbols
```

```
   REPEAT
      INPUT MaxNumberOfSymbols
   UNTIL MaxNumberOfSymbols MOD 2 = 1
ENDPROCEDURE

PROCEDURE OutputSpaces
   FOR Count ← 1 TO NumberOfSpaces
      OUTPUT Space           // without moving to next line
   ENDFOR
ENDPROCEDURE

PROCEDURE OutputSymbols
   FOR Count ← 1 TO NumberOfSymbols
      OUTPUT Symbol           // without moving to next line
   ENDFOR
   OUTPUT Newline             // move to the next line
ENDPROCEDURE

PROCEDURE AdjustValuesForNextRow
   NumberOfSpaces ← NumberOfSpaces - 1
   NumberOfSymbols ← NumberOfSymbols + 2
ENDPROCEDURE

CALL SetValues
REPEAT
   CALL OutputSpaces
   CALL OutputSymbols
   CALL AdjustValuesForNextRow
UNTIL NumberOfSymbols > MaxNumberOfSymbols
```

Python

```
SPACE = " "

def InputMaxNumberOfSymbols() :
   global MaxNumberOfSymbols
   MaxNumberOfSymbols = int(input('Input how many
symbols on base: '))
   while MaxNumberOfSymbols % 2 != 1 :
      MaxNumberOfSymbols = int(input('Input how many
symbols on base: '))

def SetValues() :
   global Symbol # str[1]
   global NumberOfSpaces # int
   global NumberOfSymbols # int
   Symbol = input('Input symbol: ')
   InputMaxNumberOfSymbols()  # need to ensure it is
an odd number
   NumberOfSpaces = (MaxNumberOfSymbols - 1) // 2
   NumberOfSymbols = 1
```

```
def OutputSpaces() :
   for Count in range(NumberOfSpaces) :
      print(SPACE, end='')               # without
moving to next line

def OutputSymbols() :
   for Count in range(NumberOfSymbols) :
      print(Symbol, end='')              # without
moving to next line
   print()                               # move to the
next line

def AdjustValuesForNextRow() :
   global NumberOfSpaces # int
   global NumberOfSymbols # int
   NumberOfSpaces -= 1
   NumberOfSymbols += 2

SetValues()
while NumberOfSymbols <= MaxNumberOfSymbols :
   OutputSpaces()
   OutputSymbols()
   AdjustValuesForNextRow()
```

VB.NET

```
Module Module1

    Const Space = " "
    Dim Symbol As Char
    Dim MaxNumberOfSymbols, NumberOfSpaces, NumberOfSymbols As
Integer

    Sub InputMaxNumberOfSymbols()
        Do
            Console.Write("Input how many symbols on base: ")
            MaxNumberOfSymbols = Console.ReadLine()
        Loop Until (MaxNumberOfSymbols Mod 2 = 1)
    End Sub

    Sub SetValues()
        Console.Write("Input symbol: ")
        Symbol = Console.ReadLine()
        InputMaxNumberOfSymbols()            ' need to ensure it is
an odd number
        NumberOfSpaces = (MaxNumberOfSymbols - 1) \ 2
        NumberOfSymbols = 1
    End Sub

    Sub OutputSpaces()
        Dim Count As Integer
        For Count = 1 To NumberOfSpaces
            Console.Write(Space)             ' without moving to next
line
        Next
    End Sub

    Sub OutputSymbols()
        Dim Count As Integer
        For Count = 1 To NumberOfSymbols
```

```
            Console.Write(Symbol)           ' without moving to next
line
        Next
        Console.WriteLine()                 ' move to the next line
    End Sub

    Sub AdjustValuesForNextRow()
        NumberOfSpaces -= 1
        NumberOfSymbols += 2
    End Sub
    Sub Main()
        SetValues()
        Do
            OutputSpaces()
            OutputSymbols()
            AdjustValuesForNextRow()
        Loop Until (NumberOfSymbols > MaxNumberOfSymbols)

        Console.ReadLine()
    End Sub

End Module
```

Pascal

```
program Project2;

{$APPTYPE CONSOLE}

uses
  SysUtils;

  const Space = ' ';
var Symbol : char;
    MaxNumberOfSymbols, NumberOfSpaces,
NumberOfSymbols : integer;

procedure InputMaxNumberOfSymbols;
begin
   repeat
      Write('Input how many symbols on base: ');
      ReadLn(MaxNumberOfSymbols);
   until MaxNumberOfSymbols MOD 2 = 1
end;

procedure SetValues;
begin
   Write('Input symbol: ');
   ReadLn(Symbol);
   InputMaxNumberOfSymbols; // need to ensure it is an
odd number
   NumberOfSpaces := (MaxNumberOfSymbols - 1) DIV 2 ;
   NumberOfSymbols := 1
end;

procedure OutputSpaces;
var Count : integer;
begin
   for Count := 1 to NumberOfSpaces do
      Write(Space);           // without moving to
```

```
next line
end;

procedure OutputSymbols;
var Count : integer;
begin
   for Count := 1 to NumberOfSymbols do
      Write(Symbol);           // without moving to
next line
   WriteLn;              // move to the next line
end;

procedure AdjustValuesForNextRow;
begin
   NumberOfSpaces := NumberOfSpaces - 1;
   NumberOfSymbols := NumberOfSymbols + 2;
end;

begin
SetValues;
repeat
   OutputSpaces;
   OutputSymbols;
   AdjustValuesForNextRow;
until NumberOfSymbols > MaxNumberOfSymbols;

ReadLn
end.
```

Task 14.02

```
CALL InitialiseBoard
CALL SetUpGame
CALL OutputBoard
WHILE GameFinished = FALSE
   CALL ThisPlayerMakesMove
   CALL OutputBoard
   CALL CheckIfThisPlayerHasWon
   IF GameFinished = FALSE
      THEN
         CALL SwapThisPlayer
   ENDIF
ENDWHILE
```

```
PROCEDURE InitialiseBoard
   FOR Row ← 1 TO 6
      FOR Column ← 1 TO 7
         Board[Row, Column] ← BLANK // use a suitable value
for blank
```

```
      ENDFOR
   ENDFOR
ENDPROCEDURE

PROCEDURE SetUpGame
   ThisPlayer ← 'O' // Player O always starts
   GameFinished ← FALSE
ENDPROCEDURE

PROCEDDURE OutputBoard
   FOR Row ← 1 TO6
      FOR Column ← 1 TO 7
         OUTPUT Board[Row, Column] // don’t move to next line
     ENDFOR
      OUTPUT Newline               // move to next line
   ENDFOR
ENDPROCEDURE

PROCEDURE ThisPlayerMakesMove
   ValidColumn ← ThisPlayerChoosesColumn // use a module to
return valid

// column number
   ValidRow ← FindNextFreePositionInColumn // use a module to
return row number
   Board[ValidRow, ValidColumn] ← ThisPlayer
ENDPROCEDURE

FUNCTION ThisPlayerChoosesColumn // returns a valid column
number
   OUTPUT "Player " ThisPlayer " 's turn."
   REPEAT
      OUTPUT "Enter a valid column number:"
      INPUT ColumnNumber
   UNTIL ColumnNumberValid = TRUE // check whether the column
number is valid
   RETURN ColumnNumber
ENDFUNCTION

FUNCTION ColumnNumberValid // returns whether or not the
column number is valid
   Valid ← FALSE
   IF ColumnNumber >= 1 AND ColumnNumber <= 7
      THEN
         IF Board[6, ColumnNumber] = BLANK // at least 1 empty
space in column
            THEN
               Valid ← TRUE
         ENDIF
   ENDIF
   RETURN Valid
ENDFUNCTION

FUNCTION FindNextFreePositionInColumn // returns the next free
position
   ThisRow ← 1
   WHILE  Board[ThisRow, ValidColumn] <> BLANK // find first
empty cell
      ThisRow ← ThisRow + 1
   ENDWHILE
   RETURN ThisRow
ENDFUNCTION

PROCEDURE CheckIfThisPlayerHasWon
   WinnerFound ← FALSE
   CALL CheckHorizontalLineInValidRow
   IF WinnerFound = FALSE
      THEN
         CALL CheckVerticalLineInValidColumn
   ENDIF
   IF WinnerFound = TRUE
      THEN
         GameFinished ← TRUE
        OUTPUT ThisPlayer " is the winner"
      ELSE
        CALL CheckForFullBoard
   ENDIF
ENDPROCEDURE

PROCEDURE CheckHorizontalLineInValidRow
   FOR i ← 1 TO 4
      IF Board[ValidRow, i] = ThisPlayer AND
         Board[ValidRow, i + 1] = ThisPlayer AND
         Board[ValidRow, i + 2] = ThisPlayer AND
         Board[ValidRow, i + 3] = ThisPlayer
         THEN
            WinnerFound ← TRUE
      ENDIF
   ENDFOR
ENDPROCEDURE

PROCEDURE CheckVerticalLineInValidColumn
   IF ValidRow = 4 OR ValidRow = 5 OR ValidRow = 6
      THEN
         IF Board[ValidRow, ValidColumn] = ThisPlayer AND
            Board[ValidRow - 1, ValidColumn] = ThisPlayer AND
            Board[ValidRow - 2, ValidColumn] = ThisPlayer AND
            Board[ValidRow - 3, ValidColumn] = ThisPlayer
            THEN
               WinnerFound ← TRUE
         ENDIF
   ENDIF
ENDPROCEDURE

PROCEDURE CheckForFullBoard
   BlankFound ← FALSE
   ThisRow ← 0
   REPEAT
      ThisColumn ← 0
      ThisRow ← ThisRow + 1
      REPEAT
         ThisColumn ← ThisColumn + 1
         IF Board[ThisRow, ThisColumn] = BLANK
            THEN
               BlankFound ← TRUE
         ENDIF
      UNTIL ThisColumn = 7 OR BlankFound = TRUE
   UNTIL ThisRow = 6 OR BlankFound = TRUE
   IF BlankFound = FALSE
      THEN
         OUTPUT  "It is a draw"
         GameFinished ← TRUE
   ENDIF
ENDPROCEDURE

PROCEDURE SwapThisPlayer
   IF ThisPlayer = 'O'
      THEN
         ThisPlayer ← 'X'
      ELSE
         ThisPlayer ← 'O'
   ENDIF
ENDPROCEDURE
```

Python

```
BLANK = "."
# Board(0:6, 0:7) : str (but ignore row 0 and column
0)
# ThisPlayer : str[1]
# GameFinished, WinnerFound : bool
# ColumnNumber : int
# ValidColumn, ValidRow : int

def InitialiseBoard() :
   global Board
   Board = [[BLANK for Column in range(8)] for Row in
range(7)]


def SetUpGame() :
   global ThisPlayer
   global GameFinished
   ThisPlayer = "O" # Player O always starts
   GameFinished = False


def OutputBoard() :
   for Row in range(6, 0, -1) :
      for Column in range (1, 8) :
         print(Board[Row][Column], end='') # don"t
move to next line
      print()               # move to next line

def ColumnNumberValid() :   # returns whether or not
the column number is valid
   global ColumnNumber
   Valid = False
   if (ColumnNumber >= 1) and (ColumnNumber <= 7) :
      if Board[6][ColumnNumber] == BLANK : # at least
1 empty space in column
         Valid = True
   return Valid


def ThisPlayerChoosesColumn() : # returns a valid
column number
   global ColumnNumber
   print("Player ", ThisPlayer, "'s turn.")
   ColumnNumber = int(input("Enter a valid column
number: "))
   while ColumnNumberValid() == False : # check
whether the column number is valid
      ColumnNumber = int(input("Enter a valid column
number: "))
   return ColumnNumber

def FindNextFreePositionInColumn() : # returns the
next free position
   ThisRow = 1
   while Board[ThisRow][ValidColumn] != BLANK : # find
first empty cell
      ThisRow +=  1
   return ThisRow

def ThisPlayerMakesMove() :
   global Board
   global ValidColumn
   global ValidRow
   ValidColumn = ThisPlayerChoosesColumn() # use a
module to return valid column number
```

```
   ValidRow = FindNextFreePositionInColumn() # use a
module to return row number
   Board[ValidRow][ValidColumn] = ThisPlayer

def CheckHorizontalLineInValidRow() :
   global WinnerFound
   for i in range(1,5) :
      if ((Board[ValidRow][i] == ThisPlayer) and
         (Board[ValidRow][i + 1] == ThisPlayer) and
         (Board[ValidRow][i + 2] == ThisPlayer) and
         (Board[ValidRow][i + 3] == ThisPlayer)) :
         WinnerFound = True

def CheckVerticalLineInValidColumn() :
   global WinnerFound
   if (ValidRow == 4) or (ValidRow == 5) or (ValidRow
== 6):
      if ((Board[ValidRow][ValidColumn] == ThisPlayer)
and
         (Board[ValidRow - 1][ValidColumn] ==
ThisPlayer) and
         (Board[ValidRow - 2][ValidColumn] ==
ThisPlayer) and
         (Board[ValidRow - 3][ValidColumn] ==
ThisPlayer)):
         WinnerFound = True

def CheckForFullBoard() :
   global GameFinished
   BlankFound = False
   ThisRow = 0
   while (ThisRow != 6) and (BlankFound == False) :
      ThisColumn = 0
      ThisRow +=  1
      while (ThisColumn != 7) and (BlankFound ==
False) :
         ThisColumn +=  1
         if Board[ThisRow][ThisColumn] == BLANK :
            BlankFound = True
   if BlankFound == False :
      print("It is a draw")
      GameFinished = True


def CheckIfThisPlayerHasWon() :
   global WinnerFound
   global GameFinished
   WinnerFound = False
   CheckHorizontalLineInValidRow()
   if WinnerFound == False :
      CheckVerticalLineInValidColumn()
   if WinnerFound == True :
      GameFinished = True
      print(ThisPlayer, " is the winner")
   else :
      CheckForFullBoard()

def SwapThisPlayer() :
   global ThisPlayer
   if ThisPlayer == "O" :
      ThisPlayer = "X"
   else :
      ThisPlayer = "O"

def main() :
   InitialiseBoard()
   SetUpGame()
   OutputBoard()
   while GameFinished == False :
      ThisPlayerMakesMove()
      OutputBoard()
      CheckIfThisPlayerHasWon()
      if GameFinished == False :
         SwapThisPlayer()

main()
```

VB.NET

```
Module Module1

    Const BLANK = "_"
    Dim Board(6, 7) As Char
    Dim ThisPlayer As Char
    Dim GameFinished, WinnerFound As Boolean
    Dim ColumnNumber As Integer
    Dim ValidColumn, ValidRow As Integer

    Sub InitialiseBoard()
        Dim Row, Column As Integer
        For Row = 1 To 6
            For Column = 1 To 7
                Board(Row, Column) = BLANK ' use a suitable value for
blank
            Next
        Next
    End Sub

    Sub SetUpGame()
        ThisPlayer = "O" ' Player O always starts
        GameFinished = False
    End Sub

    Sub OutputBoard()
        Dim Row, Column As Integer
        For Row = 6 To 1 Step -1
            For Column = 1 To 7
                Console.Write(Board(Row, Column)) ' don"t move to
next line
            Next
            Console.WriteLine()                ' move to next line
        Next
    End Sub

    Function ColumnNumberValid() As Boolean    ' returns whether or
not the column number is valid
```

```
        Dim Valid As Boolean
        Valid = False
        If (ColumnNumber >= 1) And (ColumnNumber <= 7) Then
            If Board(6, ColumnNumber) = BLANK Then ' at least 1 empty
space in column
                Valid = True
            End If
        End If
        Return Valid
    End Function

    Function ThisPlayerChoosesColumn() As Integer ' returns a valid
column number
        Console.WriteLine("Player " & ThisPlayer & " 's turn.")
        Do
            Console.Write("Enter a valid column number: ")
            ColumnNumber = Console.ReadLine
        Loop Until (ColumnNumberValid() = True) ' check whether the
column number is valid
        Return ColumnNumber
    End Function

    Function FindNextFreePositionInColumn() As Integer ' returns the
next free position
        Dim ThisRow As Integer
        ThisRow = 1
        Do While Board(ThisRow, ValidColumn) <> BLANK  ' find first
empty cell
            ThisRow = ThisRow + 1
        Loop
        Return ThisRow
    End Function

    Sub ThisPlayerMakesMove()
        ValidColumn = ThisPlayerChoosesColumn() ' use a module to
return valid column number
        ValidRow = FindNextFreePositionInColumn() ' use a module to
return row number
        Board(ValidRow, ValidColumn) = ThisPlayer
    End Sub

    Sub CheckHorizontalLineInValidRow()
        Dim i As Integer
        For i = 1 To 4
            If (Board(ValidRow, i) = ThisPlayer) And
               (Board(ValidRow, i + 1) = ThisPlayer) And
               (Board(ValidRow, i + 2) = ThisPlayer) And
               (Board(ValidRow, i + 3) = ThisPlayer) Then
                WinnerFound = True
            End If
        Next
    End Sub

    Sub CheckVerticalLineInValidColumn()
        If (ValidRow = 4) Or (ValidRow = 5) Or (ValidRow = 6) Then
            If (Board(ValidRow, ValidColumn) = ThisPlayer) And
               (Board(ValidRow - 1, ValidColumn) = ThisPlayer) And
               (Board(ValidRow - 2, ValidColumn) = ThisPlayer) And
               (Board(ValidRow - 3, ValidColumn) = ThisPlayer) Then
                WinnerFound = True
            End If
        End If
    End Sub

    Sub CheckForFullBoard()
        Dim BlankFound As Boolean
        Dim ThisRow, ThisColumn As Integer
        BlankFound = False
        ThisRow = 0
        Do
            ThisColumn = 0
            ThisRow = ThisRow + 1
            Do
                ThisColumn = ThisColumn + 1
                If Board(ThisRow, ThisColumn) = BLANK Then
                    BlankFound = True
                End If
            Loop Until (ThisColumn = 7) Or (BlankFound = True)
        Loop Until (ThisRow = 6) Or (BlankFound = True)
        If BlankFound = False Then
            Console.WriteLine("It is a draw")
            GameFinished = True
        End If
    End Sub

    Sub CheckIfThisPlayerHasWon()
        WinnerFound = False
        CheckHorizontalLineInValidRow()
        If WinnerFound = False Then
            CheckVerticalLineInValidColumn()
        End If
        If WinnerFound = True Then
            GameFinished = True
            Console.WriteLine(ThisPlayer & " is the winner")
        Else
            CheckForFullBoard()
        End If
    End Sub

    Sub SwapThisPlayer()
        If ThisPlayer = "O" Then
            ThisPlayer = "X"
        Else
            ThisPlayer = "O"
        End If
    End Sub

    Sub main()
        InitialiseBoard()
        SetUpGame()
        OutputBoard()
        Do While GameFinished = False
            ThisPlayerMakesMove()
            OutputBoard()
            CheckIfThisPlayerHasWon()
            If GameFinished = False Then
                SwapThisPlayer()
            End If
        Loop
        Console.ReadLine()
    End Sub
End Module
```

Pascal

```
program Project2;

{$APPTYPE CONSOLE}

uses
  SysUtils;

const BLANK = '_';
var Board : array[1..6, 1..7] of char;
    ThisPlayer : char;
    GameFinished, WinnerFound : Boolean;
    ColumnNumber : integer;
    ValidColumn, ValidRow : integer;

procedure InitialiseBoard;
var Row, Column : integer;
begin
   for Row :=  1 to 6  do
      for Column  := 1 to 7 do
         Board[Row, Column] :=  BLANK;
end;

procedure SetUpGame;
begin
   ThisPlayer := 'O'; // Player O always starts
   GameFinished :=  FALSE;
end;

procedure OutputBoard;
var Row, Column : integer;
begin
   for Row := 6 downto 1  do
      begin
         for Column := 1 to 7 do
            Write(Board[Row, Column]); // don't move
to next line
         WriteLn;               // move to next line
      end;
end;

function ColumnNumberValid : Boolean;
var Valid : Boolean;
begin
   Valid :=  FALSE;
   if (ColumnNumber >= 1) AND (ColumnNumber <= 7)
      then
         if Board[6, ColumnNumber] = BLANK
            then
               Valid := TRUE;
   Result := Valid;
end;

function ThisPlayerChoosesColumn : integer;
begin
   WriteLn( 'Player ', ThisPlayer, ' ''s turn.' );
   repeat
      Write('Enter a valid column number:' );
      ReadLn(ColumnNumber);
   until ColumnNumberValid = TRUE;
   Result := ColumnNumber;
end;

function FindNextFreePositionInColumn : integer;
var ThisRow : integer;
begin
   ThisRow := 1;
   while  Board[ThisRow, ValidColumn] <> BLANK do
      ThisRow := ThisRow + 1;
   Result := ThisRow;
end;

procedure ThisPlayerMakesMove;
begin
   ValidColumn := ThisPlayerChoosesColumn;
   ValidRow := FindNextFreePositionInColumn;
   Board[ValidRow, ValidColumn] := ThisPlayer;
end;

procedure CheckHorizontalLineInValidRow;
var i : integer;
begin
   for i := 1 TO 4  do
      if (Board[ValidRow, i] = ThisPlayer) AND
         (Board[ValidRow, i + 1] = ThisPlayer) AND
         (Board[ValidRow, i + 2] = ThisPlayer) AND
         (Board[ValidRow, i + 3] = ThisPlayer)
         then
            WinnerFound := TRUE;
end;

procedure CheckVerticalLineInValidColumn;
begin
   IF (ValidRow = 4) OR (ValidRow = 5) OR (ValidRow =
6)
      THEN
         IF (Board[ValidRow, ValidColumn] =
ThisPlayer) AND
            (Board[ValidRow - 1, ValidColumn] =
ThisPlayer) AND
            (Board[ValidRow - 2, ValidColumn] =
ThisPlayer) AND
            (Board[ValidRow - 3, ValidColumn] =
ThisPlayer)
            then
               WinnerFound := TRUE;
end;

procedure CheckForFullBoard;
var BlankFound : Boolean;  ThisRow, ThisColumn :
integer;
```

```
begin
   BlankFound := FALSE;
   ThisRow := 0;
   repeat
      ThisColumn := 0;
      ThisRow := ThisRow + 1;
      repeat
         ThisColumn := ThisColumn + 1;
         if Board[ThisRow, ThisColumn] = BLANK
            then
               BlankFound := TRUE;
       until (ThisColumn = 7) OR (BlankFound = TRUE);
   until (ThisRow = 6) OR (BlankFound = TRUE);
   if BlankFound = FALSE
      then
         begin
            WriteLn('It is a draw');
            GameFinished := TRUE;
         end;
end;

procedure CheckIfThisPlayerHasWon;
begin
   WinnerFound := FALSE;
   CheckHorizontalLineInValidRow ;
   if WinnerFound = FALSE
      then
         CheckVerticalLineInValidColumn;
   if WinnerFound = TRUE
      then
         begin
            GameFinished := TRUE;
            WriteLn(ThisPlayer, ' is the winner');
         end
      else
         CheckForFullBoard;
end;

procedure SwapThisPlayer;
begin
   if ThisPlayer = 'O'
      then
         ThisPlayer:= 'X'
      else
         ThisPlayer := 'O';
end;

begin
   InitialiseBoard;
   SetUpGame;
   OutputBoard;
   while GameFinished = FALSE do
   begin
      ThisPlayerMakesMove;
      OutputBoard;
```

```
         CheckIfThisPlayerHasWon;
         if GameFinished = FALSE
            then
               SwapThisPlayer;
      end;

      ReadLn
   end.
```

Task 14.03

```
FUNCTION Factorial (Number : INTEGER) : INTEGER
DECLARE Product : INTEGER
   Product ← 1
   FOR n ← 2 TO Number
      Product ← Product * n
   ENDFOR
   RETURN Product
ENDFUNCTION
```

Python

```
def Factorial (Number) :
   Product = 1
   for n in range(2,Number + 1) :
      Product = Product * n
   return Product
```

VB.NET

```
    Function Factorial(Number) As Integer

        Dim Product, n As Integer
        Product = 1
        For n = 2 To Number
            Product = Product * n
        Next
        Return (Product)
    End Function
```

Pascal

```
function Factorial (Number : integer) : integer;
var Product, n : integer;
begin
   Product := 1;
   for n := 2 to Number  do
      Product := Product * n;
   Result := Product;
end;
```

Task 14.04 part 1

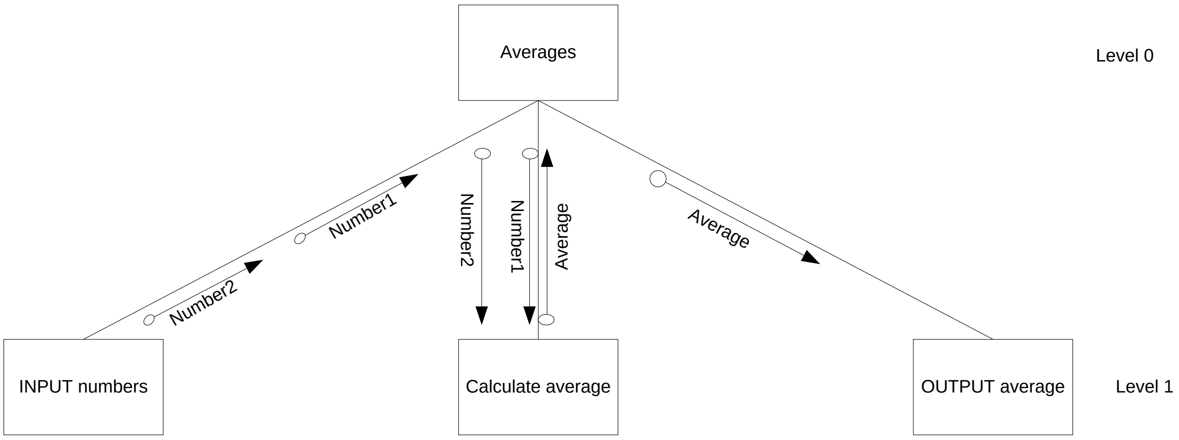


Figure 12.05

Python

```
def CalculateAverage(N1, N2) :
   Average = (N1 + N2) / 2
   return(Average)

def InputNumbers() :
   N1 = int(input("Number1: "))
   N2 = int(input("Number2: "))
   return(N1, N2)

def OutputMessage(a) :
   print("The average is ", a)

def Averages() :
   Number1, Number2 = InputNumbers()
   Average = CalculateAverage(Number1, Number2)
   OutputMessage(Average)

Averages()
```

VB.NET

```
    Function CalculateAverage(N1 As Integer, N2 As Integer) As Single
        Dim Average As Single
        Average = (N1 + N2) / 2
        Return Average
    End Function

    Sub InputNumbers(ByRef N1 As Integer, ByRef N2 As Integer)
        Console.Write("Number1: ")
        N1 = Console.ReadLine()
        Console.Write("Number2: ")
        N2 = Console.ReadLine()
    End Sub

    Sub OutputMessage(a As Single)
        Console.WriteLine("The average is " & a)
    End Sub

    Sub Averages()
        Dim Average As Single
        InputNumbers(Number1, Number2)
        Average = CalculateAverage(Number1, Number2)
        OutputMessage(Average)
    End Sub
```

Pascal

```
    function CalculateAverage(N1, N2 : Integer) : real;
    var Average : real;
    begin
        Average := (N1 + N2) / 2;
        Result := Average;
    end;

    procedure InputNumbers(var N1, N2 : Integer);
    begin
        Write('Number1: ');
        ReadLn(N1);
        Write('Number2: ');
        ReadLn(N2);
    end;

    procedure OutputMessage(a : real);
    begin
        WriteLn('The average is ', a:5:2);
    end;

    procedure Averages;
    var Average : real; Number1, Number2 : integer;
    begin
        InputNumbers(Number1, Number2);
        Average := CalculateAverage(Number1, Number2);
        OutputMessage(Average);
    end;
```

Task 14.04 part 2

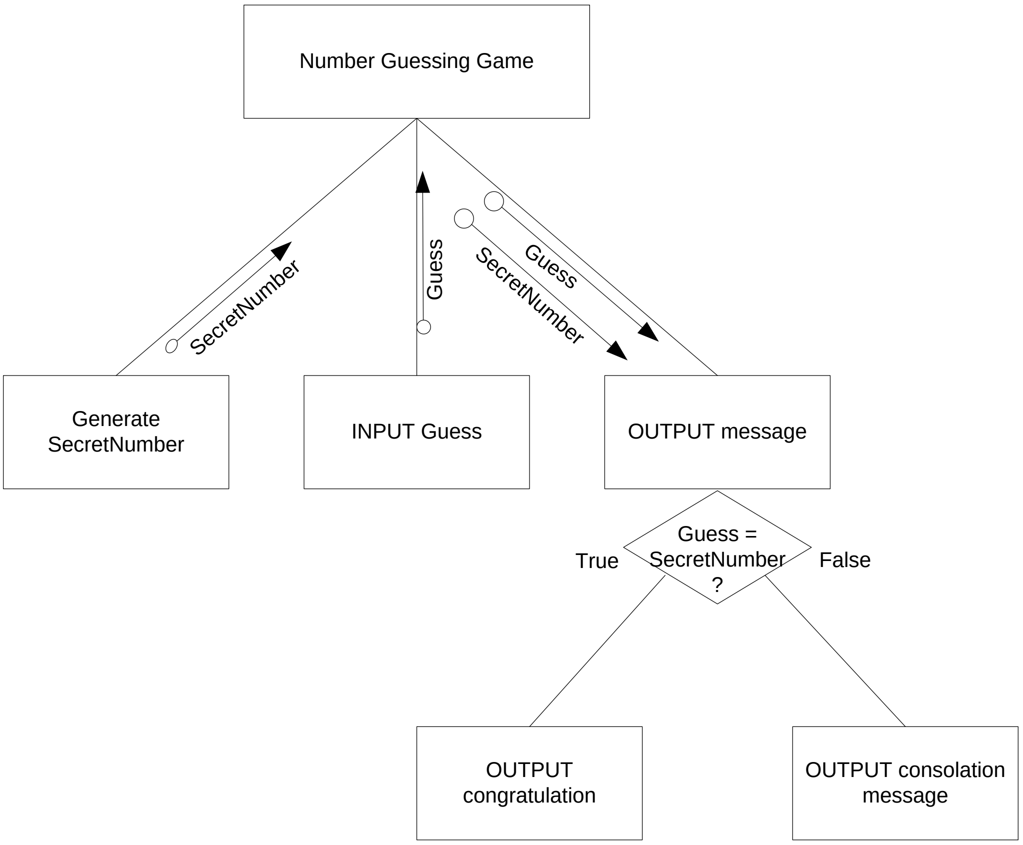


Figure 12.06

Python

```
from random import randint

def GenerateSecretNumber() :
   return(randint(1,1000))

def InputGuess() :
   Guess = int(input("What is your guess? "))
   return(Guess)

def OutputMessage(x,y) :
   if x == y :
      print("Congratulations, you have guessed
correctly")
   else :
      print("Sorry, the secret number is ", y)

def NumberGuessingGame() :
   SecretNumber = GenerateSecretNumber()
   Guess = InputGuess()
   OutputMessage(Guess, SecretNumber)

NumberGuessingGame()
```

VB.NET

```
Function GenerateSecretNumber() As Integer
    Dim RandomNumber As New Random
    Return RandomNumber.Next(1, 1000)
  End Function
```

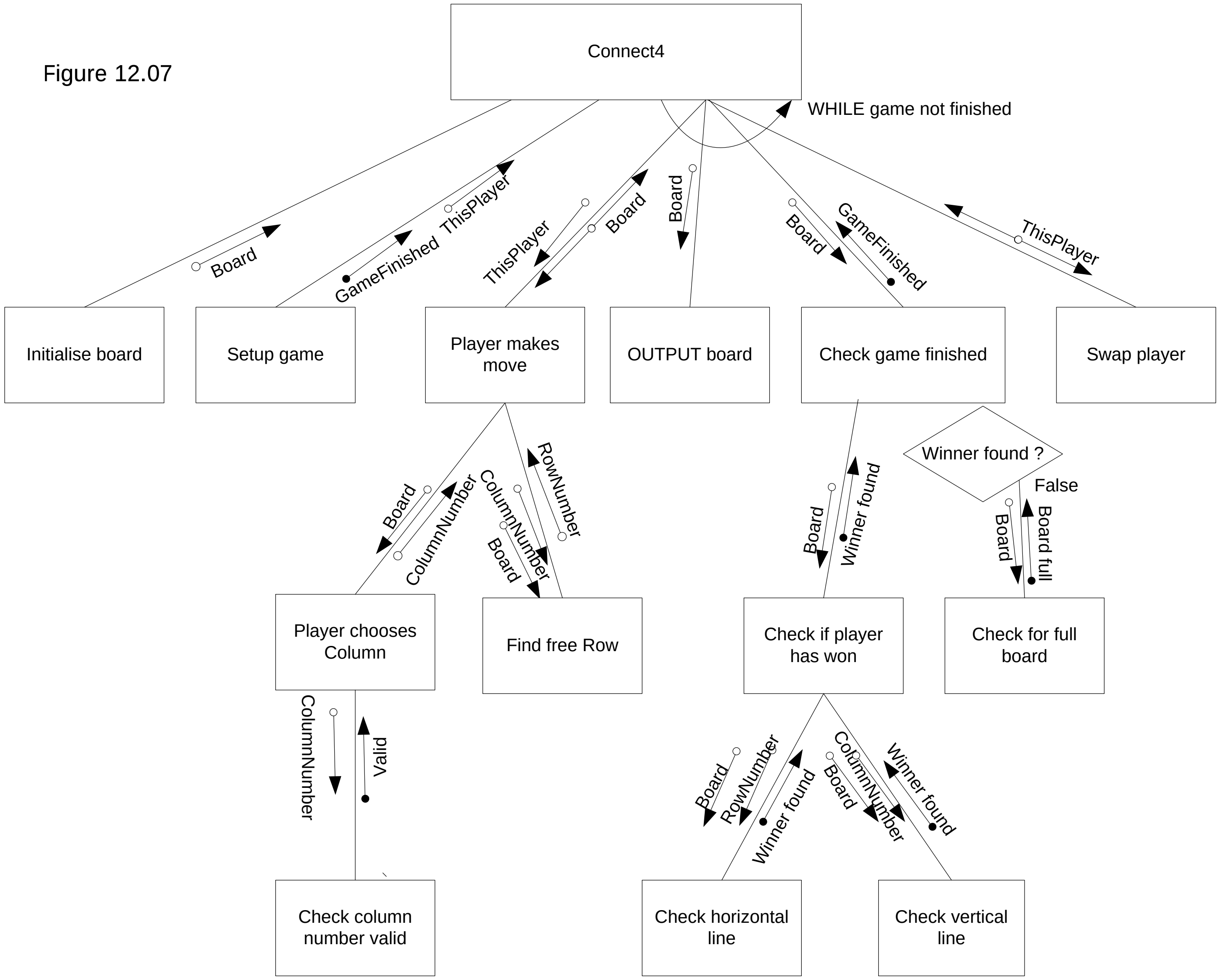
```
    Function InputGuess() As Integer
        Dim Guess As Integer

        Console.Write("What is your guess? ")
        Guess = Console.ReadLine()
        Return Guess
    End Function

    Sub OutputMessage(ByVal x As Integer, ByVal y As Integer)
        If x = y Then
            Console.WriteLine("Congratulations, you have guessed
correctly")
        Else
            Console.WriteLine("Sorry, the secret number is " & y)
        End If
    End Sub

    Sub NumberGuessingGame()
        Dim SecretNumber, Guess As Integer
        SecretNumber = GenerateSecretNumber()
        Guess = InputGuess()
        OutputMessage(Guess, SecretNumber)
    End Sub
```

Pascal

```
function GenerateSecretNumber : integer;
begin
   result := random(1000);
end;

function InputGuess : integer;
var Guess : integer;
begin
   Write('What is your guess? ');
   ReadLn(Guess);
   Result := Guess;
end;

procedure OutputMessage(x, y : integer);
begin
   if x = y
      then
         WriteLn('Congratulations, you have guessed
correctly')
      else
         WriteLn('Sorry, the secret number is ', y);
end;

procedure numberGuessingGame;
var SecretNumber, Guess : integer;
begin
   randomize;
   SecretNumber := GenerateSecretNumber;
   Guess := InputGuess;
   OutputMessage(Guess, SecretNumber);
end;
```

Task 14.04 part 3

Figure 12.07



Python

```
BLANK = "."
# Board(0:6, 0:7) : str (but ignore row 0 and column
0)
# ThisPlayer : str[1]
# GameFinished, WinnerFound : bool
# ColumnNumber : int
# ValidColumn, ValidRow : int

def InitialiseBoard() :
   Board = [[BLANK for Column in range(8)] for Row in
range(7)]
   return Board

def SetUpGame() :
   ThisPlayer = "O" # Player O always starts
   GameFinished = False
   return ThisPlayer, GameFinished
```

```
def OutputBoard(Board) :
   for Row in range(6, 0, -1) :
      for Column in range (1, 8) :
         print(Board[Row][Column], end='') # don"t
move to next line
      print()               # move to next line

def ColumnNumberValid(Board, ColumnNumber) :   #
returns whether or not the column number is valid
   Valid = False
   if (ColumnNumber >= 1) and (ColumnNumber <= 7) :
      if Board[6][ColumnNumber] == BLANK : # at least
1 empty space in column
         Valid = True
   return Valid


def ThisPlayerChoosesColumn(ThisPlayer, Board) : #
returns a valid column number
   print("Player ", ThisPlayer, "'s turn.")
   ColumnNumber = int(input("Enter a valid column
number: "))
   while ColumnNumberValid(Board, ColumnNumber) ==
False : # check whether the column number is valid
      ColumnNumber = int(input("Enter a valid column
number: "))
   return ColumnNumber

def FindNextFreePositionInColumn(Board, ValidColumn) :
# returns the next free position
   ThisRow = 1
   while Board[ThisRow][ValidColumn] != BLANK : # find
first empty cell
      ThisRow +=  1
   return ThisRow

def ThisPlayerMakesMove(ThisPlayer, Board) :
   ValidColumn = ThisPlayerChoosesColumn(ThisPlayer,
Board) # use a module to return valid column number
   ValidRow = FindNextFreePositionInColumn(Board,
ValidColumn) # use a module to return row number
   Board[ValidRow][ValidColumn] = ThisPlayer
   return ValidRow, ValidColumn

def CheckHorizontalLineInValidRow(Board, ValidRow,
ThisPlayer) :
   WinnerFound = False
   for i in range(1,5) :
      if ((Board[ValidRow][i] == ThisPlayer) and
         (Board[ValidRow][i + 1] == ThisPlayer) and
         (Board[ValidRow][i + 2] == ThisPlayer) and
         (Board[ValidRow][i + 3] == ThisPlayer)) :
         WinnerFound = True
   return WinnerFound
```

```
def CheckVerticalLineInValidColumn(Board, ValidRow,
ValidColumn, ThisPlayer) :
   WinnerFound = False
   if (ValidRow == 4) or (ValidRow == 5) or (ValidRow
== 6):
      if ((Board[ValidRow][ValidColumn] == ThisPlayer)
and
         (Board[ValidRow - 1][ValidColumn] ==
ThisPlayer) and
         (Board[ValidRow - 2][ValidColumn] ==
ThisPlayer) and
         (Board[ValidRow - 3][ValidColumn] ==
ThisPlayer)):
         WinnerFound = True
   return WinnerFound

def CheckForFullBoard(Board) :
   GameFinished = False
   BlankFound = False
   ThisRow = 0
   while (ThisRow != 6) and (BlankFound == False) :
      ThisColumn = 0
      ThisRow +=  1
      while (ThisColumn != 7) and (BlankFound ==
False) :
         ThisColumn +=  1
         if Board[ThisRow][ThisColumn] == BLANK :
            BlankFound = True
   if BlankFound == False :
      print("It is a draw")
      GameFinished = True
   return GameFinished


def CheckIfThisPlayerHasWon(Board, ValidRow,
ValidColumn, ThisPlayer) :
   GameFinished = False
   WinnerFound = CheckHorizontalLineInValidRow(Board,
ValidRow, ThisPlayer)
   if WinnerFound == False :
      WinnerFound =
CheckVerticalLineInValidColumn(Board, ValidRow,
ValidColumn, ThisPlayer)
   if WinnerFound == True :
      GameFinished = True
      print(ThisPlayer, " is the winner")
   else :
      GameFinished = CheckForFullBoard(Board)
   return GameFinished

def SwapThisPlayer(ThisPlayer) :
   if ThisPlayer == "O" :
      ThisPlayer = "X"
   else :
      ThisPlayer = "O"
   return ThisPlayer

def main() :
   Board = InitialiseBoard()
   ThisPlayer, GameFinished = SetUpGame()
   OutputBoard(Board)
   while GameFinished == False :
      ValidRow, ValidColumn =
ThisPlayerMakesMove(ThisPlayer, Board)
      OutputBoard(Board)
      GameFinished = CheckIfThisPlayerHasWon(Board,
ValidRow, ValidColumn, ThisPlayer)
      if GameFinished == False :
         ThisPlayer = SwapThisPlayer(ThisPlayer)
```

VB.NET

```
Module Module1

    Const BLANK = "_"

    Sub InitialiseBoard(ByRef Board)
        Dim Row, Column As Integer
        For Row = 1 To 6
            For Column = 1 To 7
                Board(Row, Column) = BLANK
            Next
        Next
    End Sub

    Sub SetUpGame(ByRef ThisPlayer, ByRef GameFinished)
        ThisPlayer = "O"
        GameFinished = False
    End Sub

    Sub OutputBoard(Board)
        Dim Row, Column As Integer
        For Row = 6 To 1 Step -1
            For Column = 1 To 7
                Console.Write(Board(Row, Column))
            Next
            Console.WriteLine()
        Next
    End Sub

    Function ColumnNumberValid(ByVal Board, ByVal ColumnNumber) As
Boolean
        Dim Valid As Boolean
        Valid = False
        If (ColumnNumber >= 1) And (ColumnNumber <= 7) Then
            If Board(6, ColumnNumber) = BLANK Then
                Valid = True
            End If
        End If
        Return Valid
    End Function

    Function ThisPlayerChoosesColumn(ByVal Board, ByVal ThisPlayer)
As Integer
        Dim ColumnNumber As Integer
        Console.WriteLine("Player " & ThisPlayer & " 's turn.")
        Do
            Console.Write("Enter a valid column number: ")
            ColumnNumber = Console.ReadLine
        Loop Until (ColumnNumberValid(Board, ColumnNumber) = True)
        Return ColumnNumber
    End Function

    Function FindNextFreePositionInColumn(ByVal Board, ByVal 
ValidColumn) As Integer
        Dim ThisRow As Integer
        ThisRow = 1
        Do While Board(ThisRow, ValidColumn) <> BLANK
            ThisRow = ThisRow + 1
        Loop
        Return ThisRow
    End Function

    Sub ThisPlayerMakesMove(ByVal ThisPlayer, ByRef Board, ByRef 
ValidRow, ByRef ValidColumn)
        ValidColumn = ThisPlayerChoosesColumn(Board, ThisPlayer)
        ValidRow = FindNextFreePositionInColumn(Board, ValidColumn)
        Board(ValidRow, ValidColumn) = ThisPlayer
    End Sub

    Sub CheckHorizontalLineInValidRow(ByVal Board, ByVal ValidRow, 
ByVal ThisPlayer, ByRef WinnerFound)
        Dim i As Integer
        For i = 1 To 4
            If (Board(ValidRow, i) = ThisPlayer) And
               (Board(ValidRow, i + 1) = ThisPlayer) And
               (Board(ValidRow, i + 2) = ThisPlayer) And
               (Board(ValidRow, i + 3) = ThisPlayer) Then
                WinnerFound = True
            End If
        Next
    End Sub

    Sub CheckVerticalLineInValidColumn(ByVal Board, ByVal ValidRow, 
ByVal ValidColumn, ByVal ThisPlayer, ByRef WinnerFound)
        If (ValidRow = 4) Or (ValidRow = 5) Or (ValidRow = 6) Then
            If (Board(ValidRow, ValidColumn) = ThisPlayer) And
               (Board(ValidRow - 1, ValidColumn) = ThisPlayer) And
               (Board(ValidRow - 2, ValidColumn) = ThisPlayer) And
               (Board(ValidRow - 3, ValidColumn) = ThisPlayer) Then
                WinnerFound = True
            End If
        End If
    End Sub

    Sub CheckForFullBoard(ByVal Board, ByRef GameFinished)
        Dim BlankFound As Boolean
        Dim ThisRow, ThisColumn As Integer
        BlankFound = False
        ThisRow = 0
        Do
            ThisColumn = 0
            ThisRow = ThisRow + 1
            Do
                ThisColumn = ThisColumn + 1
                If Board(ThisRow, ThisColumn) = BLANK Then
                    BlankFound = True
                End If
            Loop Until (ThisColumn = 7) Or (BlankFound = True)
        Loop Until (ThisRow = 6) Or (BlankFound = True)
        If BlankFound = False Then
            Console.WriteLine("It is a draw")
            GameFinished = True
        End If
    End Sub

    Sub CheckIfThisPlayerHasWon(ByVal Board, ByRef ThisPlayer, ByVal
ValidRow, ByVal ValidColumn, ByRef GameFinished)
        Dim WinnerFound As Boolean
        WinnerFound = False
        CheckHorizontalLineInValidRow(Board, ValidRow, ThisPlayer,
WinnerFound)
        If WinnerFound = False Then
            CheckVerticalLineInValidColumn(Board, ValidRow,
ValidColumn, ThisPlayer, WinnerFound)
        End If
        If WinnerFound = True Then
            GameFinished = True
            Console.WriteLine(ThisPlayer & " is the winner")
        Else
            CheckForFullBoard(Board, GameFinished)
        End If
    End Sub

    Sub SwapThisPlayer(ByRef ThisPlayer)
        If ThisPlayer = "O" Then
            ThisPlayer = "X"
        Else
            ThisPlayer = "O"
        End If
    End Sub

    Sub main()
        Dim ThisPlayer As Char
        Dim GameFinished As Boolean
        Dim Board(6, 7) As Char
        Dim ValidRow, ValidColumn As Integer

        InitialiseBoard(Board)
        SetUpGame(ThisPlayer, GameFinished)
        OutputBoard(Board)
        Do While GameFinished = False
            ThisPlayerMakesMove(ThisPlayer, Board, ValidRow,
ValidColumn)
            OutputBoard(Board)
            CheckIfThisPlayerHasWon(Board, ThisPlayer, ValidRow,
ValidColumn, GameFinished)
            If GameFinished = False Then
                SwapThisPlayer(ThisPlayer)
            End If
        Loop
        Console.ReadLine()
    End Sub
End Module
```

Pascal

```
program Project2;

{$APPTYPE CONSOLE}
```

```
uses
  SysUtils;

const BLANK = '_';

type BoardType = array[1..6, 1..7] of char;

procedure InitialiseBoard(var Board : BoardType);
var Row, Column : integer;
begin
   for Row :=  1 to 6  do
      for Column  := 1 to 7 do
         Board[Row, Column] :=  BLANK;
end;

procedure SetUpGame(var ThisPlayer : char; var
GameFinished : Boolean);
begin
   ThisPlayer := 'O';
   GameFinished :=  FALSE;
end;

procedure OutputBoard(Board : BoardType);
var Row, Column : integer;
begin
   for Row := 6 downto 1  do
      begin
         for Column := 1 to 7 do
            Write(Board[Row, Column]);
         WriteLn;
      end;
end;

function ColumnNumberValid(Board : BoardType;
ColumnNumber : integer) : Boolean;
var Valid : Boolean;
begin
   Valid :=  FALSE;
   if (ColumnNumber >= 1) AND (ColumnNumber <= 7)
      then
         if Board[6, ColumnNumber] = BLANK
            then
               Valid := TRUE;
   Result := Valid;
end;

function ThisPlayerChoosesColumn(Board : BoardType;
ThisPlayer : char) : integer;
var ColumnNumber : integer;
begin
   WriteLn( 'Player ', ThisPlayer, ' ''s turn.' );
   repeat
      Write('Enter a valid column number:' );
      ReadLn(ColumnNumber);
   until ColumnNumberValid(Board, ColumnNumber) =
TRUE;
   Result := ColumnNumber;
end;

function FindNextFreePositionInColumn(Board :
BoardType; Validcolumn : integer) : integer;
var ThisRow : integer;
begin
   ThisRow := 1;
   while  Board[ThisRow, ValidColumn] <> BLANK do
      ThisRow := ThisRow + 1;
   Result := ThisRow;
end;

procedure ThisPlayerMakesMove(ThisPlayer : char; var
Board : BoardType; var ValidRow, ValidColumn :
integer);
begin
   ValidColumn := ThisPlayerChoosesColumn(Board,
ThisPlayer);
   ValidRow := FindNextFreePositionInColumn(Board,
ValidColumn);
   Board[ValidRow, ValidColumn] := ThisPlayer
end;

procedure CheckHorizontalLineInValidRow(Board :
BoardType; ValidRow : integer; ThisPlayer : char; var
WinnerFound : Boolean);
var i : integer;
begin
   for i := 1 TO 4  do
      if (Board[ValidRow, i] = ThisPlayer) AND
         (Board[ValidRow, i + 1] = ThisPlayer) AND
         (Board[ValidRow, i + 2] = ThisPlayer) AND
         (Board[ValidRow, i + 3] = ThisPlayer)
         then
            WinnerFound := TRUE;
end;

procedure CheckVerticalLineInValidColumn(Board :
BoardType; ValidRow, ValidColumn : integer; ThisPlayer
: char; var WinnerFound : Boolean);
begin
   IF (ValidRow = 4) OR (ValidRow = 5) OR (ValidRow =
6)
      THEN
         IF (Board[ValidRow, ValidColumn] =
ThisPlayer) AND
            (Board[ValidRow - 1, ValidColumn] =
ThisPlayer) AND
            (Board[ValidRow - 2, ValidColumn] =
ThisPlayer) AND
            (Board[ValidRow - 3, ValidColumn] =
ThisPlayer)
            then
```

```
               WinnerFound := TRUE;
end;

procedure CheckForFullBoard(Board : BoardType; var
GameFinished : Boolean);
var BlankFound : Boolean;  ThisRow, ThisColumn :
integer;
begin
   BlankFound := FALSE;
   ThisRow := 0;
   repeat
      ThisColumn := 0;
      ThisRow := ThisRow + 1;
      repeat
         ThisColumn := ThisColumn + 1;
         if Board[ThisRow, ThisColumn] = BLANK
            then
               BlankFound := TRUE;
       until (ThisColumn = 7) OR (BlankFound = TRUE);
   until (ThisRow = 6) OR (BlankFound = TRUE);
   if BlankFound = FALSE
      then
         begin
            WriteLn('It is a draw');
            GameFinished := TRUE;
         end;
end;

procedure CheckIfThisPlayerHasWon(Board : BoardType;
var ThisPlayer : Char; ValidRow, ValidColumn :
integer; var GameFinished : Boolean);
var WinnerFound : Boolean;
begin
   WinnerFound := FALSE;
   CheckHorizontalLineInValidRow(Board, ValidRow,
ThisPlayer, WinnerFound);
   if WinnerFound = FALSE
      then
         CheckVerticalLineInValidColumn(Board,
ValidRow, ValidColumn, ThisPlayer, WinnerFound);
   if WinnerFound = TRUE
      then
         begin
            GameFinished := TRUE;
            WriteLn(ThisPlayer, ' is the winner');
         end
      else
         CheckForFullBoard(Board, GameFinished);
end;

procedure SwapThisPlayer(var ThisPlayer : char);
begin
   if ThisPlayer = 'O'
      then
         ThisPlayer:= 'X'
      else
         ThisPlayer := 'O';
end;


procedure PlayGame;
var Board : BoardType;
    ThisPlayer : char;
    GameFinished : Boolean;
    ValidColumn, ValidRow : integer;
begin
   InitialiseBoard(Board);
   SetUpGame(ThisPlayer, GameFinished);
   OutputBoard(Board);
   while GameFinished = FALSE do
   begin
      ThisPlayerMakesMove(ThisPlayer, Board, ValidRow,
ValidColumn);
      OutputBoard(Board);
      CheckIfThisPlayerHasWon(Board, ThisPlayer,
ValidRow, ValidColumn, GameFinished);
      if GameFinished = FALSE
         then
            SwapThisPlayer(ThisPlayer);
   end;
end;

begin
   PlayGame;
   ReadLn
end.
```

Exam-style questions in Chapter 14

1

Python

```
def OutputTimesTable(n) :
   for i in range(1,11) :
      Product = i * n
      print("{0:2} x {1:2} = {2:3}".format(i, n,
Product))
```

VB.NET

```
    Sub OutputTimesTable(n As Integer)
        Dim i, Product As Integer
        For i = 1 To 10
            Product = i * n
            Console.WriteLine("{0,2} x {1,2} = {2,3}", i, n, Product)
        Next
    End Sub
```

Pascal

```
    procedure OutputTimesTable(n : Integer);
    var i, Product : Integer;
    begin
      For i := 1 To 10 do
         begin
            Product := i * n;
            WriteLn(i:2, ' x ', n:2, ' = ', Product:3)
         end;
    end;
```

2

Python

```
def IsDivisible(x, y) :
   Remainder = x % y
   if Remainder == 0 :
      return True
   else :
      return False
```

VB.NET

```
    Function IsDivisible(x As Integer, y As Integer) As Boolean
        Dim Remainder As Integer
        Remainder = x Mod y
        If Remainder = 0 Then
            Return True
        Else
            Return False
        End If
    End Function
```

Pascal

```
function IsDivisible(x, y : integer) : Boolean;
var Remainder : Integer;
begin
   Remainder := x Mod y;
   if Remainder = 0
      then
         IsDivisible := True
      else
         IsDivisible := False;
end;
```

3

Python

```
def EggsIntoBoxes(NumberOfEggs) :
   EggsLeftOver = NumberOfEggs % 6
   NumberOfBoxes = NumberOfEggs // 6
   return NumberOfBoxes, EggsLeftOver
```

VB.NET

```
    Sub EggsInBoxes(ByVal NumberOfEggs As Integer, ByRef
NumberOfBoxes As Integer, ByRef EggsLeftOver As Integer)
        EggsLeftOver = NumberOfEggs Mod 6
        NumberOfBoxes = NumberOfEggs \ 6
    End Sub
```

Pascal

```
procedure EggsInBoxes(NumberOfEggs : integer; var
NumberOfBoxes, EggsLeftOver : Integer);
begin
   EggsLeftOver := NumberOfEggs Mod 6;
   NumberOfBoxes := NumberOfEggs DIV 6;
end;
```

Exercise 12.01

```
INPUT Player A's choice of secret word
INPUT Player B's guess
IF secret word = guess
   THEN
      OUTPUT "Well done"
   ELSE
      OUTPUT "Sorry, incorrect"
ENDIF
WHILE word not guessed and number of guesses less than 10
   INPUT Player B's guess
   INCREMENT number of guesses
ENDWHILE
IF word guessed correctly
   THEN
      OUTPUT number of guesses
   ELSE
      OUTPUT end game message and secret word
ENDIF
```

Identifier table

Identifier	**Data type**	**Description**
SecretWord	STRING	stores the word to be guessed
Guess	STRING	the word the current player enters
Correct	BOOLEAN	FALSE when the word has not been guessed TRUE when the secret word has been guessed
Count	INTEGER	stores the number of times the player has entered a word

Pseudocode

```
Correct ← FALSE
INPUT SecretWord
INPUT Guess
Count ← 1
IF secret word = guess
   THEN
      OUTPUT "Well done"
   ELSE
      OUTPUT "Sorry, incorrect"
ENDIF
WHILE Correct = FALSE AND Count < 10
   INPUT Guess
   Count ← Count + 1
   IF Guess = SecretWord
      THEN
```

```
         Correct ← TRUE
ENDWHILE
IF Correct = TRUE
   THEN
      OUTPUT "It took " COUNT "guesses"
   ELSE
      OUTPUT "Game over. The secret word was: " SecrectWord
ENDIF
```

Structured English

```
Generate a random number n
Open text file
Read the nth word and make this the secret word
INPUT Player A's guess
IF secret word = guess
   THEN
      OUTPUT "Well done"
   ELSE
      OUTPUT "Sorry, incorrect"
ENDIF
WHILE word not guessed and number of guesses less than 10
Swap player
   INPUT Player B's guess
   INCREMENT number of guesses
ENDWHILE
IF word guessed correctly
   THEN
      OUTPUT number of guesses
      OUTPUT the winner
   ELSE
      OUTPUT end game message and secret word
ENDIF
```

Exercise 12.02

Identifier table

Identifier	Data type	Description
SecretWord	STRING	stores the word to be guessed
Guess	STRING	the word the current player enters
Correct	BOOLEAN	FALSE when the word has not been guessed TRUE when the secret word has been guessed
Count	INTEGER	stores the number of times the players have entered a word
CurrentPlayer	CHAR	A or B, to show which player's turn it is

Pseudocode

```
Correct ← FALSE
n ← RandomNumber()
OPENFILE TextFile “SecretWords.TXT” FOR READ
FOR x ← 1 TO n-1
   ReadLine(TextFile)
ENDFOR
SecretWord ← Readline(TextFile)
CLOSEFILE TextFile
CurrentPlayer ← “A”
INPUT Guess
Count ← 1
IF secret word = guess
   THEN
      OUTPUT “Well done”
   ELSE
      OUTPUT “Sorry, incorrect”
ENDIF
WHILE Correct = FALSE AND Count < 10
   IF CurrentPlayer = “A”
      THEN
         CurrentPlayer ← “B”
      ELSE
         CurrentPlayer ← “A”
   ENDIF
   INPUT Guess
   Count ← Count + 1
   IF Guess = SecretWord
      THEN
         Correct ← TRUE
ENDWHILE
IF Correct = TRUE
   THEN
      OUTPUT “It took “ COUNT “guesses”
      OUTPUT CurrentPlayer “ is the winner”
   ELSE
      OUTPUT “Game over. The secret word was: “ SecrectWord
ENDIF
```

Exercise 12.03

Structured English

```
INPUT Player A’s choice of secret word
Set up a display word of the same length as the secret word
Each character of the display word is #
WHILE word not guessed
   INPUT guess
   IF guess is a letter in the secret word
      THEN
```

```
         Update the display word
      ELSE
         Increase penalty score
   ENDIF
   OUTPUT display word
ENDWHILE
OUTPUT penalty score
```

Identifier table

Identifier	Data type	Description
SecretWord	STRING	stores the word to be guessed
DisplayWord	STRING	stores the word with # in place of each letter not yet guessed
Guess	CHAR	the letter the player enters
Correct	BOOLEAN	FALSE when the word has not been guessed TRUE when the whole secret word has been guessed
PenaltyScore	INTEGER	a count of how many letters the player chose but are not present in the secret word

Pseudocode

```
Correct ← FALSE
INPUT SecretWord
DisplayWord ← “”
FOR i ← 1 TO LENGTH(SecretWord)
   DisplayWord ← DisplayWord & “#”
ENDFOR
WHILE Correct = FALSE
   INPUT Guess
   FOR i ← 1 TO Length(SecretWord)
      IF Guess = SecretWord[i]
         THEN
            GoodGuess ← TRUE
            DisplayWord[i] ← Guess
      ENDIF
   ENDFOR
   IF GoodGuess = FALSE
      THEN
         PenaltyScore ← PenaltyScore + 1
      ELSE
         GoodGuess ← FALSE
   ENDIF
   OUTPUT DisplayWord
   IF SecretWord = DisplayWord
      THEN
         Correct ← TRUE
   ENDIF
ENDWHILE
OUTPUT “You have “ PenaltyPoints “penalty points”
```

Exercise 14.01

Pseudocode

```
FUNCTION GetMenuChoice() RETURNS INTEGER
   OUTPUT MenuOption 1
   OUTPUT MenuOption 2
   OUTPUT MenuOption 3
   ChoiceString ← “”
   WHILE ChoiceString not in [“1”,”2”,”3”]
      OUTPUT “Enter your choice (1, 2 or 3): “
      INPUT ChoiceString
   ENDWHILE
   RETURN int(ChoiceString)
ENDFUNCTION
PROCEDURE Program1()
   OUTPUT “Program 1 called”
ENDPROCEDURE
PROCEDURE Program2()
   OUTPUT “Program 2 called”
ENDPROCEDURE
PROCEDURE Program3()
   OUTPUT “Program 3 called”
ENDPROCEDURE
PROCEDURE Main()
   CASE OF GetMenuChoice
      1: CALL Program1
      2: CALL Program2
      3: CALL Program3
   ENDCASE
ENDPROCEDURE
```