Wang Dong Yue – Pseudoku Graded Assignment

Task 1

```
function MakeVector(row)
        new Vector puzzle(4)
        puzzle[1] <- row
        puzzle[2] <- row
        puzzle[3] <- row
        puzzle[4] <- row
        return puzzle
end function
```

Task 2

```
function PermuteVector(row, p)
        if p = 0 then
                return row
        end if
        // ---------------------------------
        new Queue q
        i <- 1
        while i <= LENGTH[row] do
                ENQUEUE(row[i], q)
                i = i + 1
        end while
        // ---------------------------------
        if p > 0 && p <= 3 then
                j <- 1
                while j <= p do
                        temp <- DEQUEUE(q)
                        ENQUEUE(temp, q)
                        j = j + 1
                end while
        end if
        return row
end function
```

Task 3

```
function PermuteRows(puzzle, x, y, z)
        if 0 <= x <= 3  && 0 <= y <= 3  &&  z <= 3
                puzzle[2] <- PermuteVector(puzzle[2], x)
                puzzle[3] <- PermuteVector(puzzle[3], y)
                puzzle[4] <- PermuteVector(puzzle[4], z)
        end if
        return puzzle
end function
```

Task 4

```
function SearchStack(stack, item)
        new Stack second
        bool <- TRUE
        while TOP[stack] != item && EMPTY[stack] = FALSE do
                temp <- TOP[stack]
                PUSH[temp, second]
                POP[stack]
        end while
        // ---------------------------------
        if EMPTY[stack] = TRUE then
                bool <- FALSE
        else
                POP[stack]
        end if
        // ---------------------------------
        while EMPTY[second] != TRUE do
                temp <- TOP[second]
                PUSH[temp, stack]
                POP[second]
        end while
        // ---------------------------------
        if bool = FALSE then
                return FALSE
        end if
        return stack
end function
```

Task 5

```
function CheckColumn(puzzle, j)
        new Stack numbers
        i <- 1
        while i < 5 do
                PUSH[k, numbers]
                i = i + 1
        end while
        // ---------------------------------
        k <- 1
        value <- puzzle[k][j]
        while k < 5
                if SearchStack(numbers, value) = FALSE
                        return FALSE
                else
                        k = k + 1
        end while
        // ---------------------------------
        if k = 5
                return TRUE
        end if
end function



// I will place the ColChecks function referred from the assignment here for reference later

function ColChecks(puzzle)
        j <- 1
        for 1 <= j <= 4 do
                if CheckColumn(puzzle, j) = FALSE then
                        return FALSE
                else
                        j = j + 1
                end if
        end for
        return TRUE
end function
```

Task 6

This function aims to programmatically get the values of the subgrid so that it is scalable for larger sudoku sets, rather than hardcoding subgrids into the function.

```
function CheckGrids(puzzle)
        y <- 1
        for 1 <= y <= 4 do
                x <- 1
                for 1 <= x <= 4 do
                        // ---------------------------------
                        // RESET THE STACK
                        new Stack numbers
                        i <- 1
                        while i < 5 do
                                PUSH[k, numbers]
                                i = i + 1
                        end while
                        // ---------------------------------
                        // CHECK SUBGRID NOW
                        a <- x
                        b <- y
                        c <- b + 1
                        for b <= c do
                                value <- puzzle[a][b]
                                if SearchStack(numbers, value) = FALSE
                                        return FALSE
                                else
                                        b = b + 1
                        end for
                        a <- a + 1
                        b <- b - 1
                        for b <= c do
                                value <- puzzle[a][b]
                                if SearchStack(numbers, value) = FALSE
                                        return FALSE
                                else
                                        b = b + 1
                        end for
                        x = x + 2
                end for
                y = y + 2
        end for
        if y = 5
                return TRUE
        end if
end function
```

Task 7

Using the function describe in the introductory portion of task 6 of ColChecks to complement this: Check if the values in the column add up to 10, since 4 + 3 + 2 + 1 gives 10. It is impossible for the sum of 4 values to give a value of 10 and the product of 4 values to give 24 both together without the combination of 4, 3, 2, 1.

```
function CheckColumn(puzzle, j)
        new Vector column(4)
        i <- 1
        for 1 <= i <= 4 do
                column[i] <- puzzle[i][j]
                i = i + 1
        end for
        // ---------------------------------
        new Queue q
        x <- 1
        while x <= LENGTH[column] do
                ENQUEUE(column[x], q)
                x = x + 1
        end while
        // ---------------------------------
        y <- 1
        sum <- 0
        product <- 1
        for 1 <= y <= 4 do
                store <- DEQUEUE(q)
                if store != 0 then
                        sum = sum + store
                        product = product * store
                else
                        return FALSE
                y = y + 1
        end for
        // ---------------------------------
        if sum = 10 && product = 24 then
                return TRUE
        else
                return FALSE
        end if
end function
```

Task 8

```
function MakeSolution(row)
        FixedPuzzle <- MakeVector(row)
        bool <- FALSE
        while bool = FALSE
                Puzzle <- PermuteRows(FixedPuzzle, x, y, z)
                if CheckGrids(puzzle) && ColChecks(puzzle) = TRUE
                        bool <- TRUE
                        return Puzzle
                end if
        end while
end function
```

Task 9

It is possible to generate multiple coordinates that are the same through the RandonNumber() function, thus not producing a sudoku puzzle with the correct number of blanks input.
To solve it, there needs to be an if conditional statement placed in order to filter out if the coordinates has already been blanked out before, and if the same coordinate is generated, than the loop should start again at the same value of n to generate another coordinate.

Task 10

There should be a function that allows us to generate subgrids.

```
function MakeGrid(puzzle)
        new Vector grid[4]
        a = 1
        b = 1
        i = 1
        for 1 <= b <= 4 do
                for 1 <= a <= 4 do
                        grid[i] = [ [puzzle[a][b], puzzle[a][b+1], puzzle[a+1][b], puzzle[a+1][b+1] ]
                        a = a + 2
                        i = i + 1
                end for
                b = b + 2
        end for
end function
```

This function allows us to take the values from the subgrids into 4 array rows so we can more easily compare the array of 4 numbers with the SearchStack to see if the numbers meet the criteria.