STAT2005 Programming Languages for Statistics
Exercise for Chapter 4

1. (a) Initialize a graph without the origin and border, set the x- and y-axis on the range from 0 to 100. Add a square with vertex (0,0), (0,100), (100,0), and (100,100).

(b) Given four points $A_1 = (0,100), B_1 = (100,100), C_1 = (100,0), D_1 = (0,0)$.

Then, we have the recursive relation

$$A_n = \frac{A_{n-1}+B_{n-1}}{2}, B_n = \frac{B_{n-1}+C_{n-1}}{2}, C_n = \frac{C_{n-1}+D_{n-1}}{2}, D_n = \frac{D_{n-1}+A_{n-1}}{2}, n > 2.$$

Write R codes to add squares $A_1B_1C_1D_1, A_2B_2C_2D_2, A_3B_3C_3D_3, \ldots\ldots, A_{100}B_{100}C_{100}D_{100}$ on the same graph.

2. A number is a monodigit if it is a positive integer consists of a single repeated digit only, e.g. 2, 33, 444, 5555.

(a) Write function `checkmono(x)` to check whether a number is a monodigit. `checkmono(x)` returns the repeated digit if `x` is a monodigit, otherwise returns 0.

(b) Using `checkmono()`, write function `mono(n)` to return the sum of digits of monodigits between 1 to n. For example, the monodigits between 1 to 11 are 1, 2, 3, 4, 5, 6, 7, 8, 9 and 11, `mono(11)` returns 47 because $1+2+3+4+5+6+7+8+9+1+1 = 47$.

(c) Find the sum of digits of monodigits between 1 to 100,000.

(d) Risky thinks the approach in part (b) is too slow. Instead of checking every number, he wants to improve the time complexity of the program by enumerating all monodigits between 1 to n. Using Risky's approach, write function `mono2(n)` to return the sum of digits of monodigits between 1 to n.

3. Apple City is a city which use apples as badges. This city has n citizens. The more apples a citizen has, the higher rank s/he gets. The number of apples of each citizen has are stored in a vector `apple`, i.e. `apple[1]` corresponds to citizen 1, `apple[2]` corresponds to citizen 2, …… , `apple[n]` corresponds to citizen n.

(a) The government gives apples to or takes apples away from citizens whenever necessary. For each action, it add the number of apples from citizen x to citizen y by integer m, where x, y are integers and $1 \le x \le y \le n$. Note that m can be negative and no more apples can be taken away when a citizen does not have any apples. Without using loops, write function `modify(x,y,m)` to add every value from `apple[x]`, `apple[x+1]`,…, `apple[y-1]`, `apple[y]` by m.

(b) A citizen is 'good ' if the number of apples s/he has is within a particular range. Write function `count(lower,upper)` to count the number of 'good ' citizens in Apple City, where `lower` and `upper` represents the lower and upper bound of the range respectively.

1.(a)

```
plot(0, 0, type="n", xlim=c(0,100), ylim=c(0,100), bty ="n", xlab="", ylab="")
polygon( c(0, 0 ,100, 100), c(0,100,0,100))
```

(b)

```
ax <- 0; ay <- 100; bx <- 100; by <- 100
cx <- 100; cy <- 0; dx <- 0; dy <- 0
for (i in 1:100) {
    polygon( c(ax, bx, cx, dx), c(ay, by, cy, dy))
    ax_tmp <- (ax + bx) / 2;
     ....

}
```

2(a)

```
checkmono <- function (x) {
    if (x <= 0) {
        return (F)
    }
    digit <- x %% 10
    while (x >= 10) {
        if (x %% 10 != digit) {
            return (F)
        }
        x <- x %/% 10
    }
    return (T)
}
```

(b)

```
mono <- function (n) {
    monodigits <- c()
    for (i in 1:n) {
        if (checkmono(i)) {
            monodigits <- c(monodigits, i)
        }
    }
    return (sum(monodigits))
```

(c)

```
mono(100008)
```

(d)

```
mono2 <- function (n) {
    mds <- c()
    base <- 1
    repeat {
        for (scale in 1:9) {
            tmp <- base * scale
            if (tmp > n) {
                return (sum(mds))
            }
            mds <- c(mds, tmp)
        }
        base <- base *10 +1
    }
}
```

```
# Q3a
apple <- 1:10 # declare a global vector apple to test this function
# Answer:
modify <- function(x,y,m){
  apple[x:y] <<- apple[x:y]+m
  # use <<- to modify global variable
  apple[apple<0] <<- 0
  # change negative values to 0
}

# Q3b
count <- function(lower,upper){
  return(sum(apple>=lower&apple<=upper))
  # return (length(apple[apple>=lower&apple<=upper])) is also ok
}
```

3(a) note: in function to modify global var, use <<-

```
modify <- function (x, y, m) {
    apply_idx <- ifelse (c(1:n) >= x && (1:n) <= y, 1, 0)
    apply_amount <- apply_idx * m
    apple <- apple + apply amount
    return ( ifelse( apple < 0 , 0, apple))
}
```

(b)

```
count <- function (lower, upper) {
    mask <- ifelse ( apple >= lower && apple <= upper, 1, 0)
    return ( sum (mask))
}
```