Chapter 37: The Defer Statement

Section 37.1: When to use a defer statement

A `defer` statement consists of a block of code, which will be executed when a function returns and should be used for cleanup.

As Swift's `guard` statements encourage a style of early return, many possible paths for a return may exist. A `defer` statement provides cleanup code, which then does not need to be repeated every time.

It can also save time during debugging and profiling, as memory leaks and unused open resources due to forgotten cleanup can be avoided.

It can be used to deallocate a buffer at the end of a function:

```
func doSomething() {
    let data = UnsafeMutablePointer<UInt8>(allocatingCapacity: 42)
    // this pointer would not be released when the function returns
    // so we add a defer-statement
    defer {
        data.deallocateCapacity(42)
    }
    // it will be executed when the function returns.

    guard condition else {
        return /* will execute defer-block */
    }

}   // The defer-block will also be executed on the end of the function.
```

It can also be used to close resources at the end of a function:

```
func write(data: UnsafePointer<UInt8>, dataLength: Int) throws {
    var stream:NSOutputStream = getOutputStream()
    defer {
        stream.close()
    }

    let written = stream.write(data, maxLength: dataLength)
    guard written >= 0 else {
        throw stream.streamError! /* will execute defer-block */
    }

}    // the defer-block will also be executed on the end of the function
```

Section 37.2: When NOT to use a defer statement

When using a defer-statement, make sure the code remains readable and the execution order remains clear. For example, the following use of the defer-statement makes the execution order and the function of the code hard to comprehend.

```
postfix func ++ (inout value: Int) -> Int {
    defer { value += 1 } // do NOT do this!
    return value
}
```