```swift
//
//  Processes.swift
//  SimulatedPageAllocation
//
//  Created by Brandon Baars on 3/28/18.
//  Copyright © 2018 Brandon Baars. All rights reserved.
//

import Foundation

// Type to store which type the 'Frame' is
enum Type {
    case data
    case code
    case empty
}


// Our Frame holds the frame size
// whether or not it's currently occupied
// and the data associated with it.
// Also, the index (offset) it is in the frame array
struct Frame {

    var frameSize = 512.0
    var isOccupied: Bool
    var index: Int
    var type: Type
    var data: ProcessData?

    // Reset our frame by reseting all values
    public mutating func resetFrame(withIndex index: Int) {
        isOccupied = false
        self.index = index
        type = .empty
        data = nil
    }

    // dynamically set the frame size
    public mutating func setFrameSize(newSize size: Double) {
        self.frameSize = size
    }
}

class Processes {

    private(set) var textFileData: String?
    private(set) var textFileLine: [String]?

    // array of all our processes
    private(set) var processes = [ProcessData]()
```

```
private var currentIndex = 0

// our current process (starts at 0)
// and is incrememnted when the user clicks next
public var currentProcess: ProcessData!

init(withFilename file: String) {

    // Load in our Text File
    if let path = Bundle.main.path(forResource: file, ofType: ".txt") {
        do {
            let data = try String(contentsOfFile: path, encoding: .utf8)
            let parsedData = data.replacingOccurrences(of: "\r", with: "")
            textFileData = parsedData
            var lines = parsedData.components(separatedBy:
             CharacterSet(charactersIn: "\n"))
            lines.removeLast()
            // parse the lines
            parseLines(withFileContents: lines)

        } catch {
            print ("Error has occured")
        }
    }

    currentProcess = processes[0]
    currentIndex = 0
}

private func parseLines(withFileContents contents: [String]) {

    textFileLine = [String]()

    // Each line creates a 'new process' and it's added
    // to our process array
    // process with -1 are set to 'terminated'
    for line in contents {
        var page: ProcessData
        var processLine = line.split(separator: Character(" "))
        textFileLine?.append(line)
        if processLine.count > 2 {
            page = ProcessData(numOfDataPages: 0, numOfCodePages: 0,
             codePageTable: [:], dataPageTable: [:], isTerminated: false,
             processNumber: Int(processLine[0])!, codeLength:
             Int(processLine[1])!, dataLength: Int(processLine[2])!)
        } else {
            page = ProcessData(numOfDataPages: 0, numOfCodePages: 0,
             codePageTable: [:], dataPageTable: [:], isTerminated: true,
             processNumber: Int(processLine[0])!, codeLength: 0,
             dataLength: 0)
        }
```

```
            processes.append(page)
        }
    }

    public func getLineForCurrentProcess() -> String? {
        return textFileLine?[currentIndex]
    }

    public func resetProcesses() {
        currentIndex = 0
        currentProcess = processes[0]
    }

    // returns the process to be terminated by process number
    public func getProcessFromProcessNumber(pid: Int) -> ProcessData? {
        return processes.filter({$0.processNumber == pid && !
         $0.isTerminated}).first
    }

    // incrememnt the variable to the next process in our array
    public func toNext() -> Bool {
        currentIndex += 1
        if currentIndex > (processes.count - 1) { return false }
        let pageData = processes[currentIndex]
        currentProcess = pageData
        return true
    }

    // returns the next process in our array without actually incrememnting to
     it
    public func getNext() -> ProcessData? {
        if currentIndex + 1 > (processes.count - 1) { return nil}
        return processes[currentIndex + 1]
    }
}
```