```swift
//
//  ViewController.swift
//  SimulatedPageAllocation
//
//  Created by Brandon Baars on 3/27/18.
//  Copyright © 2018 Brandon Baars. All rights reserved.
//

import UIKit

class ViewController: UIViewController {

    // MARK: - IBOutlets
    @IBOutlet weak var processTextView: UITextView!
    @IBOutlet weak var processDataTextView: UITextView!
    @IBOutlet weak var nextButton: UIButton!
    @IBOutlet var frameLabels: [UILabel]!
    @IBOutlet weak var ramStackView: UIStackView!
    @IBOutlet weak var resetButton: UIButton!
    @IBOutlet weak var textPickerView: UIPickerView!
    @IBOutlet weak var frameSizeTextField: UITextField!
    // MARK: - Variables
    var pages: Processes!
    var ram: RAM!

    private var testFiles: [String] = ["test1", "test2", "test3", "test4",
     "test5"]
    private var currentFile: String!

    private var codeColors: [UIColor] = [🟦,🟪,🟥,🟧,🟩,⬛,🟨]
    private var dataColors: [UIColor] = [🟦,🟪,🟥,🟧,🟩,⬜,🟨]

    private var lastHighlightedLine: String!

    override func viewDidLoad() {
        super.viewDidLoad()

        currentFile = testFiles[0]

        // Init our 'processes' with the associated file
        pages = Processes(withFilename: currentFile)

        // create our RAM
        ram = RAM(sizeOfRam: frameLabels.count)

        resetRamLabels()
        nextButton.layer.cornerRadius = 5.0
        resetButton.layer.cornerRadius = 5.0

        processTextView.text = pages.textFileData
        processDataTextView.text = ""
```

```
    textPickerView.delegate = self
    textPickerView.dataSource = self

    frameSizeTextField.delegate = self

    highlight(text: pages.getLineForCurrentProcess()!, forColor:
     UIColor.red)
    lastHighlightedLine = pages.getLineForCurrentProcess()
}

// MARK: - IBActions
@IBAction func resetButtonPressed(_ sender: Any) {
    reset()
}

@IBAction func nextButtonPressed(_ sender: Any) {

    if lastHighlightedLine != nil {
        dehighlight(text: lastHighlightedLine)
    }

    if !pages.currentProcess.isTerminated {
        pages.currentProcess.queueProcessForRAM(withFrameSize:
         Int(ram.RAM[0].frameSize))

        processDataTextView.text! += pages.currentProcess.toString() + "\n"
        findAvailableRAMFrames(forProcess: pages.currentProcess)

        if !pages.toNext() {
           processDataTextView.text! += "End of Program 0\n"
        } else {
            highlight(text: pages.getLineForCurrentProcess()!, forColor:
             UIColor.red)
            lastHighlightedLine = pages.getLineForCurrentProcess()
        }

    } else {
        print(pages.currentProcess.toString())
        print("Attempting to remove process: ",
         pages.currentProcess.processNumber)
        removeProcessFromRAM(process: pages.currentProcess)

        if !pages.toNext() {
            processDataTextView.text! += "End of Program 0\n"
        } else {
            highlight(text: pages.getLineForCurrentProcess()!, forColor:
             UIColor.red)
            lastHighlightedLine = pages.getLineForCurrentProcess()
        }
    }
```

```
}

private func highlight(text: String, forColor color: UIColor) {
    let range = (processTextView.text as NSString).range(of: text)
    let string = NSMutableAttributedString(attributedString:
     processTextView.attributedText)
    string.addAttributes([NSAttributedStringKey.foregroundColor: color],
     range: range)
    processTextView.attributedText = string
}

private func dehighlight(text: String) {
    let range = (processTextView.text as NSString).range(of: text)
    let string = NSMutableAttributedString(attributedString:
     processTextView.attributedText)
    string.addAttributes([NSAttributedStringKey.foregroundColor:
     UIColor.darkGray], range: range)
    processTextView.attributedText = string
}

private func reset() {

    ram.resetRAM()
    pages.resetProcesses()
    dehighlight(text: processTextView.text)
    processDataTextView.text = ""
    resetRamLabels()

    highlight(text: pages.getLineForCurrentProcess()!, forColor:
     UIColor.red)
    lastHighlightedLine = pages.getLineForCurrentProcess()
}

private func removeProcessFromRAM(process: ProcessData) {

    if let processToBeRemoved = pages.getProcessFromProcessNumber(pid:
     process.processNumber) {

        let frameCodeIndices = Array(processToBeRemoved.codePageTable.keys)
        let framePageIndices = Array(processToBeRemoved.dataPageTable.keys)

        print(processToBeRemoved.toString())

        for index in frameCodeIndices {
            print("Removing Index from Code frame: ", index)

            UIView.transition(with:
             self.ramStackView.arrangedSubviews[index], duration: 0.35,
             options: .transitionCrossDissolve, animations: {
                self.ram.RAM[index].isOccupied = false
                self.ram.RAM[index].data = nil
```

```
                self.frameLabels[index].text = "Free"
                (self.ramStackView.arrangedSubviews[index] as?
                 ViewFX)?.backgroundColor = ⬜
                (self.ramStackView.arrangedSubviews[index].subviews[0] as?
                 UILabel)?.textColor = UIColor.darkGray

            }, completion: nil)
        }

        for index in framePageIndices {
            print("Removing Index from Page frame: ", index)

            UIView.transition(with:
             self.ramStackView.arrangedSubviews[index], duration: 0.35,
             options: .transitionCrossDissolve, animations: {
                self.ram.RAM[index].isOccupied = false
                self.ram.RAM[index].data = nil
                self.frameLabels[index].text = "Free"
                (self.ramStackView.arrangedSubviews[index] as?
                 ViewFX)?.backgroundColor = ⬜
                (self.ramStackView.arrangedSubviews[index].subviews[0] as?
                 UILabel)?.textColor = UIColor.darkGray

            }, completion: nil)
        }

        process.removeFromRAM()

        processDataTextView.text! += "Removed Process: \
         (process.processNumber)\n"

    } else {
         print("Could not remove process")
    }
}

private func findAvailableRAMFrames(forProcess process: ProcessData) {

    ram.addProcessToRam(withProcess: process)

    var frameCodeIndices = Array(process.codePageTable.keys)
    var framePageIndices = Array(process.dataPageTable.keys)

    frameCodeIndices.sort(by: {$1 > $0})
    framePageIndices.sort(by: {$1 > $0})

    var code = 0, data = 0

    for index in frameCodeIndices {
```

```
        UIView.transition(with: self.ramStackView.arrangedSubviews[index],
         duration: 0.5, options: .transitionCrossDissolve, animations: {

            self.processDataTextView.text! += "Loaded Code \(code) of
             process \(process.processNumber) to frame \(index)\n"
            self.updateRamView(withIndex: index, withString: "Code - \
             (code) of P\(process.processNumber)")
            (self.ramStackView.arrangedSubviews[index] as?
             ViewFX)?.backgroundColor =
             self.codeColors[process.processNumber % self.dataColors.count]
            (self.ramStackView.arrangedSubviews[index].subviews[0] as?
             UILabel)?.textColor = UIColor.white
            code += 1

        }, completion: nil)
    }

    for index in framePageIndices {

        UIView.transition(with: self.ramStackView.arrangedSubviews[index],
         duration: 0.5, options: .transitionCrossDissolve, animations: {

            self.processDataTextView.text! += "Loaded Data  \(data) of
             process \(process.processNumber) to frame \(index)\n"
            self.updateRamView(withIndex: index, withString: "Data - \
             (data) of P\(process.processNumber)")
            (self.ramStackView.arrangedSubviews[index] as?
             ViewFX)?.backgroundColor =
             self.self.dataColors[process.processNumber %
             self.dataColors.count]
            (self.ramStackView.arrangedSubviews[index].subviews[0] as?
             UILabel)?.textColor = UIColor.white
            data += 1

        }, completion: nil)
    }
}

private func updateRamView(withIndex index: Int, withString processString:
 String) {
    frameLabels[index].text = processString
}


private func resetRamLabels() {
    for (index, label) in frameLabels.enumerated() {
        label.text = "Free"
        (ramStackView.arrangedSubviews[index] as? ViewFX)?.backgroundColor
         = 
        (ramStackView.arrangedSubviews[index].subviews[0] as?
         UILabel)?.textColor = UIColor.darkGray
```

```
        }
    }
}

extension UIView {
    func copyView<T: UIView>() -> T {
        return NSKeyedUnarchiver.unarchiveObject(with:
         NSKeyedArchiver.archivedData(withRootObject: self)) as! T
    }
}

extension ViewController: UITextFieldDelegate, UIPickerViewDelegate,
 UIPickerViewDataSource {

    func textFieldShouldReturn(_ textField: UITextField) -> Bool {

        if let size = textField.text, let sizeDouble = Double(size) {
            pages = nil
            pages = Processes(withFilename: currentFile)
            ram.changeFrameSize(withSize: sizeDouble)
                        reset()
        }

        textField.resignFirstResponder()
        return false
    }

    func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int,
     inComponent component: Int) {

        if testFiles[row] == currentFile { return }

        currentFile = testFiles[row]
        pages = nil
        pages = Processes(withFilename: currentFile)
        processTextView.text = pages.textFileData
        reset()
    }

    func pickerView(_ pickerView: UIPickerView, titleForRow row: Int,
     forComponent component: Int) -> String? {
        return testFiles[row]
    }

    func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent
     component: Int) -> Int {
        return testFiles.count
    }

    func numberOfComponents(in pickerView: UIPickerView) -> Int {
        return 1
```

```
    }
}
```