COMP 1601A: Mobile Application Development

Carleton University
Winter 2022 Midterm Exam Solutions

March 4, 2022

1. [2] Create a simple SwiftUI program that shows "Hello, World" at the bottom of the screen. Give the contents of ContentView.swift only.
   A: All you need is the regular Text(Hello, World) and a Spacer() to move it to the bottom.

```
struct ContentView: View {
    var body: some View {
        Spacer()
        Text("Hello, world!")
            .padding()
    }
}
```

2. [2] Consider the following code:

```
var colours = ["red", "blue"]
colours.append("green")
```

   (a) [1] What will be the value of `colours` after the following Swift code is executed?
   A: ["red", "blue", "green"] (an array of three strings)

   (b) [2] Would this code work if we declared `colours` with `let` rather than `var`? Why?
   A: No, because let defines immutable values (constants), and so you wouldn't be able to modify the array with an append operation.

3. [4] How could you change textanalyzer-2 so the title would be "Text Analyzer: Mode" where Mode is the current analysisMode? Please give line by line changes that you would make and explain why you made each change.
   A: We need to change the title, which is displayed by the AppTitle view. The simplest way is to change what is passed to AppTitle, replacing line 15's

```
AppTitle(title: "Text Analyzer")
```

   with

```
AppTitle(title: "Text Analyzer: " + analysisMode)
```

   The displayed title then becomes the combination of the two strings, and since analysisMode is a @State variable, AppTitle will be refreshed every time analysisMode changes.

   Alternately, we can pass analysisMode to the AppTitle view by changing line 15 to:

```
AppTitle(title: "Text Analyzer", mode: $analysisMode)
```

   Then we'd add a @Binding variable for analysisMode (connecting it to the state variable that was passed in) by changing line 26 to be

```
@Binding var mode: String
```

   And then we change how the title is displayed on line 29 to be:

```
Text(title + ": " + mode).font(.title).bold().padding()
```

Note that this version will initially display "Text Analyzer: None", so a better version would use a conditional somewhere to use the old title if the analysisMode is "None".

(2 points for the code changes, 2 for the explanation/description.)

4. [2] When you select a new mode in textanalyzer-2, the analysis of the text immediately changes. What part(s) of the program, in its ContentView.swift, are responsible for making sure the analysis result is updated when a new mode is selected? Be specific.
A: Line 11 is ultimately responsible for making sure that the analysis result is updated when a new mode is selected. Here analysisMode is declared as an @State variable. Because of this, changes in the state of analysisMode (i.e., the user choosing a different mode from the menu) will cause any referencing view to be refreshed, such as AnalysisResult.

5. [2] Lines 73–77 in textanalyzer-2 uses a `for` loop. Could we replace this `for` loop with a `ForEach` loop (such as the one on lines 56–61)? If you can, rewrite the loop. If you can't, explain why.
A: You can't replace it with a ForEach loop because ForEach is used to create SwiftUI views, not do arbitrary flow control. You can only use ForEach in a View, not in a regular function. (There is a forEach in Swift that is a method of sequences such as arrays that invokes a given closure on each element of the sequence. Giving an example of forEach would get you half credit for this question.)

6. [2] How could you add a button "Erase Input" to textanalyzer-2 that erases the current user input? Place it after the user input area but above the result area. Be specific.
A: We just need to place a Button between the TextField and AnalysisResult in the body of ContentView (lines 16 and 17). The Button should have the following code:

```
Button("Erase Input", action: {t = ""})
```

Note the action is just an anonymous function that sets t (the @State variable for the TextField on line 16) to the empty string.

7. [2] In remotePicViewer2, what is the purpose of `images`? And, what kind of data structure is it?
A: `images` is a dictionary that maps String's to String's. Its purpose is to store the list of known image URLs and their associated shorter names, with those shorter names displayed for selection in the "Known Images" menu. When a new URL is saved, it is added to `images` dictionary. `images` stores the URL associated with each image name, with the image names being in the menu of available images.

8. [2] After running the following code, what type is `paths`? What is its contents?

```
let PATH = "/usr/local/bin:/usr/bin:/bin"
let paths = PATH.split(separator: ":")
```

A: `paths` is an array of String's (technically, SubString's) with the contents of ["/usr/local/bin", "/usr/bin", "/bin"].

9. [2] Are triple tap actions well supported in SwiftUI? Explain and give evidence.
A: Triple tap actions are not well supported in SwiftUI. We know this because in Assignment 2, question 8, we tried implementing a triple tap gesture and as a consequence we lost the double tap functionality that was previously working.

10. [2] Lines 81-92 in remotePicViewer call a number of methods that modify the state and behaviour of the displayed image. Is the order of these method calls significant? How do you know?
A: The order is significant. One example of why is in A2Q11, we moved .scaleEffect and .rotationEffect from being after .position to just before it. Before this change scaling and rotating wouldn't work if the image had been dragged to a new position; after this change, they work as you would expect, with the zooming and rotating happening around the middle of the image and not the middle of the screen. Another is if you move the single tap handler to before the double tap handler (swap lines 91 and 92), the program will no longer recognize the double tap gestures.

11. [3] When specifying a gesture handler, when would you use an `.onChanged` handler? When would you use an `.onEnded` handler? Give a specific example for each.
A: You would use an .onChanged handler when you want to get updates as the gesture happens; an .onEnded handler should be used when you want to be informed when the gesture is finished. If you want to update the display as the user is performing a drag (say, to make a circle track the user's finger), you would use an .onChanged handler. If you want to know where a drag ends (say, whether a dragged circle is dropped inside of a square), you would use an .onEnded handler. (1 point for difference between the two, 1 each for the two examples)

End of exam. Code listings begin on the next page.

A textanalyzer-2 Conversions.swift

Link to text version of code.

```
7  import SwiftUI
8
9  struct ContentView: View {
10     @State private var t = ""
11     @State private var analysisMode = "None"
12     var body: some View {
13         VStack{
14             ModeMenu(analysisMode: $analysisMode)
15             AppTitle(title: "Text Analyzer")
16             TextField("Enter Text", text: $t).padding()
17             AnalysisResult(mode: $analysisMode,
18                            userInput: $t)
19             Spacer()
20         }
21     }
22 }
23
24 struct AppTitle: View {
25     var title: String
26
27     var body: some View {
28         Spacer()
29         Text(title).font(.title).bold().padding()
30         Spacer()
31     }
32 }
33
34 struct AnalysisResult: View {
35     @Binding var mode: String
36     @Binding var userInput: String
37
38     var body: some View {
39         Spacer()
40         if let analysisFunc = analysis[mode] {
41             Text(mode + ": " + analysisFunc(userInput))
42         } else {
43             Text("Please Select a Mode")
44         }
45         Spacer()
46     }
47 }
48
49 struct ModeMenu: View {
50     @Binding var analysisMode: String
51
52     var body: some View {
53         let availModes = [String](analysis.keys)
54
55         Menu("Analysis menu") {
56             ForEach(availModes, id: \.self) {
57                 mode in
58                 Button(mode, action: {
59                     analysisMode = mode
```

```
                })
            }
        }

    }
}

func countUpper(s: String) -> String {
    var count = 0
    let upperCase: Set<Character> =
    ["A","B","C","D","E","F","G","H","I","J","K","L","M",
     "N","O","P","Q","R","S","T","U","V","W","X","Y","Z"]

    for c in s {
        if (upperCase.contains(c)) {
            count += 1
        }
    }

    return String(count)
}

func countPetsMentioned(s: String) -> String {
    var count = 0
    let pets = ["Roshi", "Tab", "Shift"]

    for p in pets {
        if (s.contains(p)) {
            count += 1
        }
    }
    return String(count)
}

let analysis: [String: (String) -> String] = [
    "Count": {s in return String(s.count)},
    "Empty": {s in return (s == "") ? "Yes" : "No"},
    "Upper Case": countUpper,
    "Pets Mentioned": countPetsMentioned
]
```

B remotePicViewer2 Conversions.swift

Link to text version of code.

```
13 import SwiftUI
14
15 var images: [String: String] = [
16     "Kittens": "https://homeostasis.scs.carleton.ca/~soma/mad-2022w/images/kittens.
       jpeg",
17     "Sad Dog":
18         "https://homeostasis.scs.carleton.ca/~soma/mad-2022w/images/roshi.jpeg",
19 ]
20
21 @available(macOS 12.0, *)
22 struct ContentView: View {
23     @State private var imageName = "Kittens"
24     @State private var theImage = getImage("Kittens")
25     @State private var moved = false
26     @State private var finalAmount: CGFloat = 1
27     @State private var angle = Angle(degrees: 0.0)
28     @State private var imageNames = [String] (images.keys)
29
30     func resetState() {
31         moved = false
32         finalAmount = 1
33         angle = Angle(degrees: 0.0)
34     }
35
36     var body: some View {
37         VStack {
38             Text("Image Viewer")
39                 .font(.title)
40             TextField("Enter an image URL", text: $theImage)
41             Menu("Known Images") {
42                 ForEach(imageNames, id: \.self) {s in
43                     Button(s, action: {
44                         theImage = getImage(s)
45                         resetState()
46                     })
47                 }
48                 Divider()
49                 Button("Remember Image", action: {
50                     let urlParts = theImage.split(separator: "/")
51                     let newName = String(urlParts[urlParts.count - 1])
52                     imageNames.append(newName)
53                     images[newName] = theImage
54                 })
55             }
56             Divider().background(Color.black)
57             ActiveImage(theImage: $theImage, imageName: $imageName, moved: $moved,
                   finalAmount: $finalAmount, angle: $angle)
58         }
59     }
60 }
61
62 func getImage(_ imageName: String) -> String {
63     return images[imageName] ?? "https://homeostasis.scs.carleton.ca/~soma/year.jpg"
```

```
}

@available(macOS 12.0, *)
struct ActiveImage: View {
    @State private var position = CGPoint(x: 0, y: 0)
    @State private var currentAmount: CGFloat = 0

    @Binding var theImage: String
    @Binding var imageName: String
    @Binding var moved: Bool
    @Binding var finalAmount: CGFloat
    @Binding var angle: Angle

    var body: some View {
        GeometryReader {g in
            AsyncImage(url: URL(string: theImage), content: {image in
                image
                    .resizable()
                    .scaledToFit()
                    .position(moved ? position :
                                CGPoint(x: g.size.width / 2,
                                        y: g.size.height / 2))
                    .scaleEffect(finalAmount + currentAmount)
                    .gesture(dragging)
                    .gesture(magnifying)
                    .rotationEffect(self.angle)
                    .onLongPressGesture(perform: tapReset)
                    .onTapGesture(count: 2, perform: zoom)
                    .onTapGesture(count: 1, perform: rotateImage)
            }, placeholder: {
                ProgressView()
                    .position(moved ? position :
                                CGPoint(x: g.size.width / 2,
                                        y: g.size.height / 2))
            })
        }
    }

    func zoom() {
        self.finalAmount = self.finalAmount * 2
    }

    func tapReset() {
        self.finalAmount = 1
        self.moved = false
        self.angle = Angle(degrees: 0.0)
    }

    func rotateImage() {
        let oldDegrees = self.angle.degrees
        self.angle = Angle(degrees: oldDegrees + 45)
    }

    var magnifying: some Gesture {
        MagnificationGesture().onChanged { amount in
            self.currentAmount = amount - 1
        }
```

```
121             .onEnded { amount in
122                 self.finalAmount += self.currentAmount
123                 self.currentAmount = 0
124             }
125         }
126 
127         var dragging: some Gesture {
128             DragGesture()
129                 .onChanged {s in
130                     self.moved = true
131                     self.position = s.location
132                 }
133                 .onEnded {s in
134                     self.moved = true
135                     self.position = s.location
136                 }
137         }
138 }
```