AP CSP Create Task: Exemplar

1. Program Code

Please see the final pages of this PDF file for complete program code, including comments.

2. Video

The video demonstration my program can be found here.

3. Written Responses

a)

i. The program's purpose is to help someone improve at multiplication. This benefits elementary school-aged children initially mastering arithmetic and also high school students since knowing the factors of a given number is a key skill required to factor polynomials efficiently.

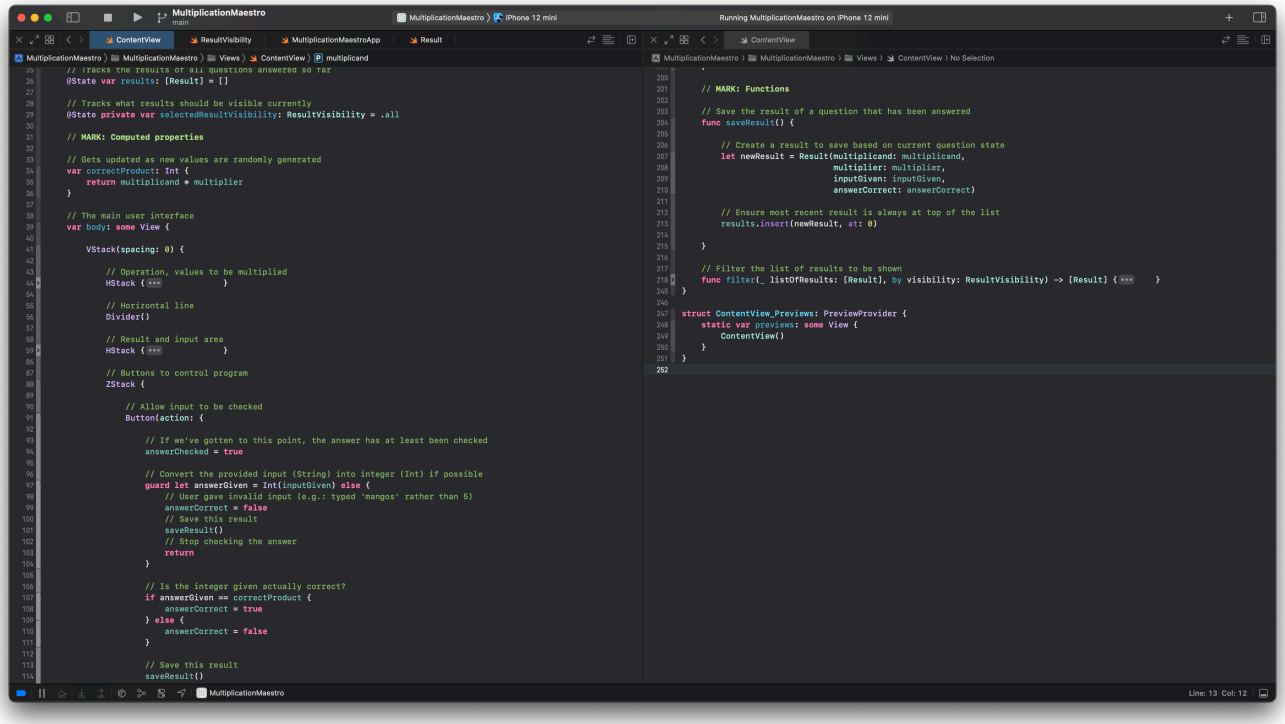
ii. Up to 00:35 the user completes questions and sometimes provides correct responses. The program indicates whether an answer is correct or not by showing a checkmark or an "x". A reverse-chronological history of the user's results is generated in a scrollable list below. From 00:44 to 00:54 the user reviews and filters results to see what questions were answered correctly and which were not.

iii. Input occurs when the user answers a question or selects how to filter prior results. Output occurs when the program generates numbers, indicates whether an answer is correct or not, and when showing the list of prior results.

responses continue on next page

b)

i.

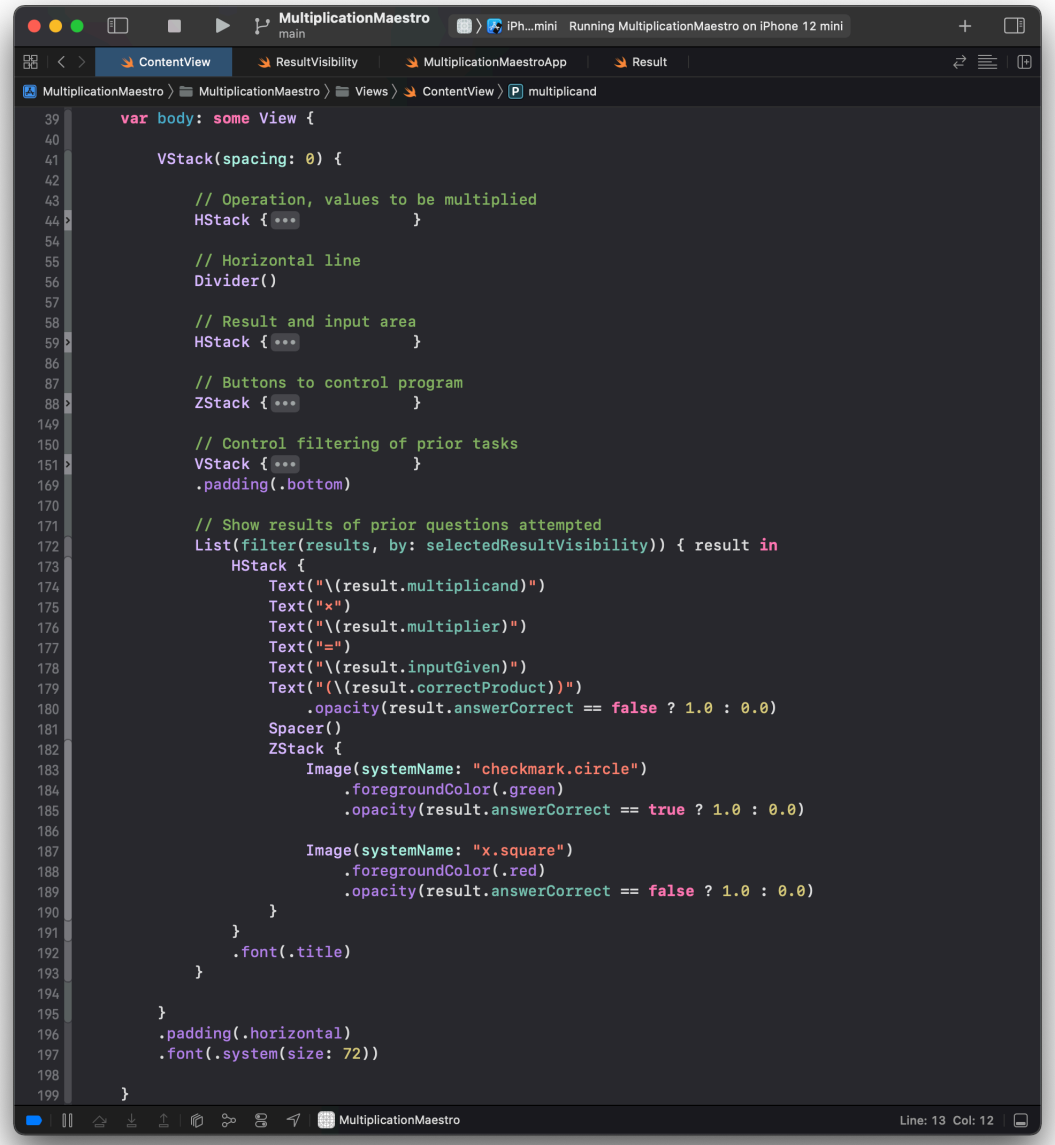


```
    // Tracks the results of all questions answered so far
    @State var results: [Result] = []

    // Tracks what results should be visible currently
    @State private var selectedResultVisibility: ResultVisibility = .all

    // MARK: Computed properties

    // Gets updated as new values are randomly generated
    var correctProduct: Int {
        return multiplicand * multiplier
    }

    // The main user interface
    var body: some View {

        VStack(spacing: 0) {

            // Operation, values to be multiplied
            HStack { ... }

            // Horizontal line
            Divider()

            // Result and input area
            HStack { ... }

            // Buttons to control program
            ZStack {

                // Allow input to be checked
                Button(action: {

                    // If we've gotten to this point, the answer has at least been checked
                    answerChecked = true

                    // Convert the provided input (String) into integer (Int) if possible
                    guard let answerGiven = Int(inputGiven) else {
                        // User gave invalid input (e.g.: typed 'mangos' rather than 5)
                        answerCorrect = false
                        // Save this result
                        saveResult()
                        // Stop checking the answer
                        return
                    }

                    // Is the integer given actually correct?
                    if answerGiven == correctProduct {
                        answerCorrect = true
                    } else {
                        answerCorrect = false
                    }

                    // Save this result
                    saveResult()
```

```
    // MARK: Functions

    // Save the result of a question that has been answered
    func saveResult() {

        // Create a result to save based on current question state
        let newResult = Result(multiplicand: multiplicand,
                               multiplier: multiplier,
                               inputGiven: inputGiven,
                               answerCorrect: answerCorrect)

        // Ensure most recent result is always at top of the list
        results.insert(newResult, at: 0)

    }

    // Filter the list of results to be shown
    func filter(_ listOfResults: [Result], by visibility: ResultVisibility) -> [Result] { ... }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

Second program code segment can be found on the following page.

ii.



iii. The name of the list is *results* – see line 26 of ContentView.swift where it is initialized.

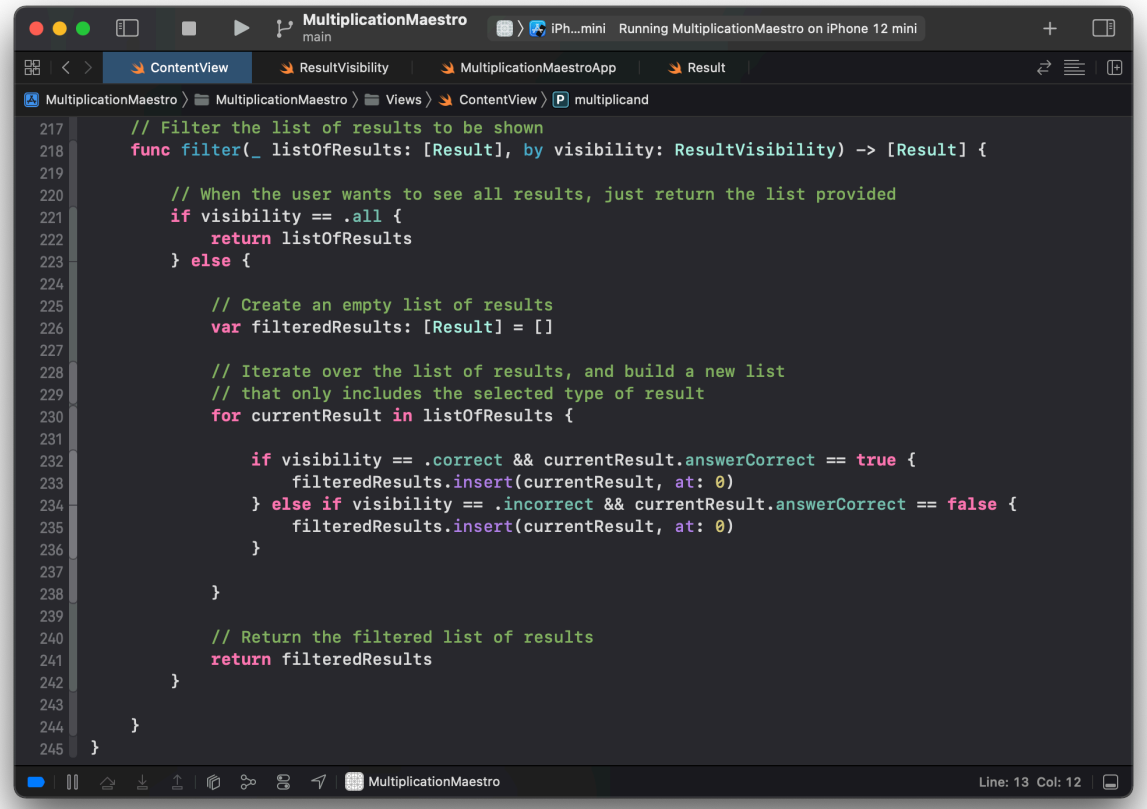
iv. The data contained in the list represent the user's history of results in answering multiplication questions. In the first program code segment the list is populated (lines 101 and 114) when a user's response is checked for accuracy. The *saveResult* function is called and it creates an instance of the Result structure (lines 207 to 210) and then inserts that instance as the first element in the *results* list (line 213).

v. The *results* list (the array defined on line 26) manages complexity because it is not possible to build a scrollable list in the user interface (lines 172 to 193) with individual variables. This is because the List structure in the SwiftUI framework requires a list (array) that it will iterate over to build the scrollable list.

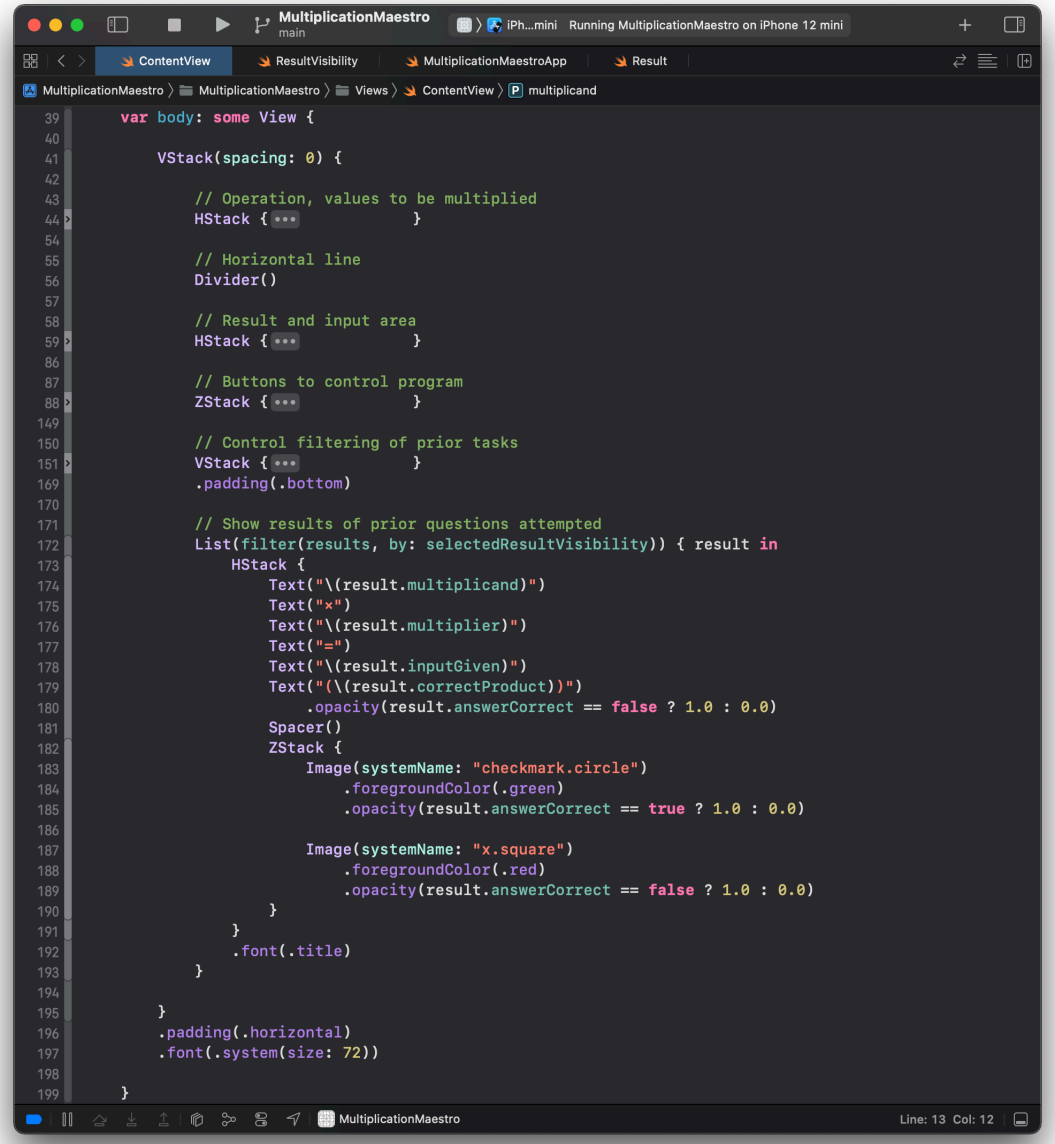
Further, even if it were possible to build a scrollable list from individual variables, how many questions the user will complete is open-ended, so it would not be possible to anticipate how many individual variables would be needed to store results without the use of the results list (array) defined on line 26.

c)

i.



ii.



iii. The *filter* function limits the list of results displayed based on what the user chose to see. In the included video from 00:42 to 00:54 you can see the *filter* function in action. Without this function it would not be possible to show a subset of the results the user has generated by answering questions.

iv. The *filter* function uses sequence by first checking to see if the user wants to see all results (line 221). If so the function simply returns the original list (line 222).

If the user chose to see only some results an empty list of filtered results is initialized (line 226). The function then iterates over the provided list of results (lines 230 to 238). Within the body of the loop a selection statement uses two conditions to ensure that when correct answers are to be shown, that only results where the user answered correctly are added to the filtered list (lines 232 and 233). Otherwise, two more conditions are used, similarly, to determine when results with incorrect answers are added to the filtered list (lines 234 and 235). Finally, after iteration over the provided list of results is complete, the filtered list is returned to be displayed in the user interface (line 241).

d)

i. The first call to the *filter* function occurs on line 172 of ContentView.swift and the argument for the "by" parameter, the content of *selectedResultVisibility,* is ".correct". The argument's value was set as a result of input to a picker control through the user interface (at 00:47 in the included video); see line 164 of ContentView.swift.

The second call to the *filter* function also occurs on line 172 of ContentView.swift but the argument this time for the "by" parameter, the content of *selectedResultVisibility*, is ".all". The argument's value was set as a result of input to a picker control through the user interface (at 00:49 in the included video); see line 160 of ContentView.swift.

ii. The conditions tested for the first call are on line 232.

The condition tested for the second call is on line 221.

iii. When "visibility" (the internal parameter name for the external parameter "by") is set to ".correct" only results where the user answered the question correctly are returned within the filtered list.

When "visibility" is set to ".all" there is no need to select what results are shown and so, simply, the original and unfiltered list of results is returned.

```swift
//
//  MultiplicationMaestroApp.swift
//  MultiplicationMaestro
//

import SwiftUI

@main
struct MultiplicationMaestroApp: App {
    var body: some Scene {
        WindowGroup {
            ContentView()
        }
    }
}
```

```swift
//
//  ResultVisibility.swift
//  MultiplicationMaestro
//

import Foundation

// An enumeration is like a Bool, or boolean value, in that
// an enumeration has a defined list of possible values.
// The advantage is that with an enumeration, we can control
// what the possible values are.

// ResultVisibility is used to determine what results
// should be visible in the list of results below the
// question and answer interface.
enum ResultVisibility: String {
    case all = "All"
    case incorrect = "Incorrect"
    case correct = "Correct"
}
```

```swift
//
//  Result.swift
//  MultiplicationMaestro
//

import Foundation

// Tracks the result of answering a single question
struct Result: Identifiable {

    // MARK: Stored properties
    let id = UUID()
    let multiplicand: Int
    let multiplier: Int
    let inputGiven: String
    let answerCorrect: Bool

    // MARK: Computed properties
    var correctProduct: Int {
        return multiplicand * multiplier
    }

}
```

```swift
//
//  ContentView.swift
//  MultiplicationMaestro
//

import SwiftUI

struct ContentView: View {

    // MARK: Stored properties

    // Values to be multiplied
    @State var multiplicand = Int.random(in: 1...12)
    @State var multiplier = Int.random(in: 1...12)

    // Holds the user's input
    @State var inputGiven = ""

    // Tracks whether the input has even been checked yet
    @State var answerChecked: Bool = false

    // Tracks whether the provided answer is correct or not
    @State var answerCorrect: Bool = false

    // Tracks the results of all questions answered so far
    @State var results: [Result] = []

    // Tracks what results should be visible currently
    @State private var selectedResultVisibility: ResultVisibility = .all

    // MARK: Computed properties

    // Gets updated as new values are randomly generated
    var correctProduct: Int {
        return multiplicand * multiplier
    }

    // The main user interface
    var body: some View {

        VStack(spacing: 0) {

            // Operation, values to be multiplied
            HStack {
                Text("×")

                Spacer()

                VStack(alignment: .trailing) {
                    Text("\(multiplicand)")
                    Text("\(multiplier)")
                }
            }

            // Horizontal line
            Divider()

            // Result and input area
            HStack {

                ZStack {
                    Image(systemName: "checkmark.circle")
                        .foregroundColor(.green)
                        // Only show this when the answer given is correct
```

```
65                      //            CONDITION          true  false
66                      .opacity(answerCorrect == true ? 1.0 : 0.0)
67
68                  Image(systemName: "x.square")
69                      .foregroundColor(.red)
70                      // Show this when both of the following situations are true:
71                      // 1. Answer has been checked.
72                      // 2. Answer was not correct.
73                      // Necessary since if we show this only when an answer is incorrect,
74                      // with no other conditions, it would show as soon as a new
75                      // question is generated.
76                      //                 CONDITION1  AND  CONDITION2             true  false
77                      .opacity(answerChecked == true && answerCorrect == false ? 1.0 : 0.0)
78              }
79
80              Spacer()
81
82              TextField("", text: $inputGiven)
83                  // Ensure input is right-aligned
84                  .multilineTextAlignment(.trailing)
85          }
86
87          // Buttons to control program
88          ZStack {
89
90              // Allow input to be checked
91              Button(action: {
92
93                  // If we've gotten to this point, the answer has at least been checked
94                  answerChecked = true
95
96                  // Convert the provided input (String) into integer (Int) if possible
97                  guard let answerGiven = Int(inputGiven) else {
98                      // User gave invalid input (e.g.: typed 'mangos' rather than 5)
99                      answerCorrect = false
100                     // Save this result
101                     saveResult()
102                     // Stop checking the answer
103                     return
104                 }
105
106                 // Is the integer given actually correct?
107                 if answerGiven == correctProduct {
108                     answerCorrect = true
109                 } else {
110                     answerCorrect = false
111                 }
112
113                 // Save this result
114                 saveResult()
115
116             }, label: {
117                 Text("Check Answer")
118                     .font(.largeTitle)
119             })
120                 // Only show this button when an answer has not been checked
121                 .opacity(answerChecked == false ? 1.0 : 0.0)
122                 .padding()
123                 .buttonStyle(.bordered)
124
125             // Allow new question to be generated
126             Button(action: {
127
128                 // Generate a new question
129                 multiplicand = Int.random(in: 1...12)
```

```
130                     multiplier = Int.random(in: 1...12)
131
132                     // Reset properties that track what's happening with the current question
133                     answerChecked = false
134                     answerCorrect = false
135
136                     // Reset the input field
137                     inputGiven = ""
138
139                 }, label: {
140                     Text("New question")
141                         .font(.largeTitle)
142                 })
143                     .padding()
144                     .buttonStyle(.bordered)
145                     // Only show this button when an answer has been checked
146                     .opacity(answerChecked == true ? 1.0 : 0.0)
147
148             }
149
150             // Control filtering of prior tasks
151             VStack {
152                 // Label for picker
153                 Text("Filter by...")
154                     .font(Font.caption.smallCaps())
155                     .foregroundColor(.secondary)
156
157                 // Picker to allow user to select what tasks to show
158                 Picker("Filter", selection: $selectedResultVisibility) {
159                     Text(ResultVisibility.all.rawValue)
160                         .tag(ResultVisibility.all)
161                     Text(ResultVisibility.incorrect.rawValue)
162                         .tag(ResultVisibility.incorrect)
163                     Text(ResultVisibility.correct.rawValue)
164                         .tag(ResultVisibility.correct)
165                 }
166                 .pickerStyle(.segmented)
167                 .padding(.horizontal)
168             }
169             .padding(.bottom)
170
171             // Show results of prior questions attempted
172             List(filter(results, by: selectedResultVisibility)) { result in
173                 HStack {
174                     Text("\(result.multiplicand)")
175                     Text("×")
176                     Text("\(result.multiplier)")
177                     Text("=")
178                     Text("\(result.inputGiven)")
179                     Text("(\(result.correctProduct))")
180                         .opacity(result.answerCorrect == false ? 1.0 : 0.0)
181                     Spacer()
182                     ZStack {
183                         Image(systemName: "checkmark.circle")
184                             .foregroundColor(.green)
185                             .opacity(result.answerCorrect == true ? 1.0 : 0.0)
186
187                         Image(systemName: "x.square")
188                             .foregroundColor(.red)
189                             .opacity(result.answerCorrect == false ? 1.0 : 0.0)
190                     }
191                 }
192                 .font(.title)
193             }
194
```

```
195         }
196         .padding(.horizontal)
197         .font(.system(size: 72))
198 
199     }
200 
201     // MARK: Functions
202 
203     // Save the result of a question that has been answered
204     func saveResult() {
205 
206         // Create a result to save based on current question state
207         let newResult = Result(multiplicand: multiplicand,
208                                multiplier: multiplier,
209                                inputGiven: inputGiven,
210                                answerCorrect: answerCorrect)
211 
212         // Ensure most recent result is always at top of the list
213         results.insert(newResult, at: 0)
214 
215     }
216 
217     // Filter the list of results to be shown
218     func filter(_ listOfResults: [Result], by visibility: ResultVisibility) -> [Result] {
219 
220         // When the user wants to see all results, just return the list provided
221         if visibility == .all {
222             return listOfResults
223         } else {
224 
225             // Create an empty list of results
226             var filteredResults: [Result] = []
227 
228             // Iterate over the list of results, and build a new list
229             // that only includes the selected type of result
230             for currentResult in listOfResults {
231 
232                 if visibility == .correct && currentResult.answerCorrect == true {
233                     filteredResults.insert(currentResult, at: 0)
234                 } else if visibility == .incorrect && currentResult.answerCorrect == false {
235                     filteredResults.insert(currentResult, at: 0)
236                 }
237 
238             }
239 
240             // Return the filtered list of results
241             return filteredResults
242         }
243 
244     }
245 }
246 
247 struct ContentView_Previews: PreviewProvider {
248     static var previews: some View {
249         ContentView()
250     }
251 }
```