Pseudo Code Interpreter

Computer science coursework

JACOB RYAN Sieradzki

2018

Table of Contents

[Background 5](#_Toc512684236)

[Interview 7](#_Toc512684237)

[Evaluation of Interview 11](#_Toc512684238)

[Objectives 11](#_Toc512684239)

[Pseudo Code Cheat Sheet for CIE iGCSE and AQA A-Level 12](#_Toc512684240)

[Basics 12](#_Toc512684241)

[Loops 13](#_Toc512684242)

[Other bits and pieces 13](#_Toc512684243)

[Example Program 14](#_Toc512684244)

[Existing Methods of Teaching Pseudo Code 14](#_Toc512684245)

[iOS Applications and Swift 16](#_Toc512684246)

[Swift Syntax 17](#_Toc512684247)

[Methods 17](#_Toc512684248)

[Optionals 17](#_Toc512684249)

[Types in Swift 18](#_Toc512684250)

[Conditionals 19](#_Toc512684251)

[Enums and Associated Values 20](#_Toc512684252)

[Other information 20](#_Toc512684253)

[Designs 21](#_Toc512684254)

[Modelling 23](#_Toc512684255)

[Class Diagram 24](#_Toc512684256)

[Data Dictionary 25](#_Toc512684257)

[Regular Expressions (Regex) 25](#_Toc512684258)

[Shunting Yard Algorithm 26](#_Toc512684259)

[ProgramInterpreter Class 32](#_Toc512684260)

[Program Class 37](#_Toc512684261)

[Instruction Class 38](#_Toc512684262)

[InputInstruction Class 42](#_Toc512684263)

[OutputInstruction Class 43](#_Toc512684264)

[AssignmentInstruction Class 43](#_Toc512684265)

[ForInstruction Class 44](#_Toc512684266)

[EndForInstruction Class 46](#_Toc512684267)

[IfInstruction Class 46](#_Toc512684268)

[ElseInstruction & EndIfInstruction Classes 47](#_Toc512684269)

[RepeatInstruction Class 47](#_Toc512684270)

[RepeatUntilInstruction Class 48](#_Toc512684271)

[WhileInstruction 49](#_Toc512684272)

[CommandParser Class 51](#_Toc512684273)

[CodeErrorHandler Class 54](#_Toc512684274)

[Incorrect Case Syntax Error 55](#_Toc512684275)

[Unterminated String Literal Syntax Error 56](#_Toc512684276)

[Other Syntax Errors 56](#_Toc512684277)

[Shunting Yard Parser 56](#_Toc512684278)

[Operator Class 58](#_Toc512684279)

[Queue Class 59](#_Toc512684280)

[Stack 60](#_Toc512684281)

[Polish 60](#_Toc512684282)

[Boolean Expression Parser 61](#_Toc512684283)

[RegexExtractor Class 62](#_Toc512684284)

[Text Matches function 62](#_Toc512684285)

[Matches function 63](#_Toc512684286)

[RegexSubstituter Class 64](#_Toc512684287)

[Output Substitution 65](#_Toc512684288)

[Assignment Substitution 65](#_Toc512684289)

[For Loop Substitution 65](#_Toc512684290)

[If Substitution 66](#_Toc512684291)

[IF A THEN 66](#_Toc512684292)

[iPad App Interface 66](#_Toc512684293)

[Development Screen 66](#_Toc512684294)

[UITableView 67](#_Toc512684295)

[Console and Controls 70](#_Toc512684296)

[Input Screen 71](#_Toc512684297)

[Error Detail Screen 72](#_Toc512684298)

[Library Screen 73](#_Toc512684299)

[Technical 75](#_Toc512684300)

[Interpreter 75](#_Toc512684301)

[Instructions 79](#_Toc512684302)

[Parsers 83](#_Toc512684303)

[Math Expression Parser 87](#_Toc512684304)

[String Expression Parser 92](#_Toc512684305)

[Boolean Expression Parser 93](#_Toc512684306)

[Regex 95](#_Toc512684307)

[Views 98](#_Toc512684308)

[Home 98](#_Toc512684309)

[Input 108](#_Toc512684310)

[Errors 110](#_Toc512684311)

[Library 112](#_Toc512684312)

[User Interface Files 114](#_Toc512684313)

[Testing Strategy 131](#_Toc512684314)

[Application Testing 131](#_Toc512684315)

[Pseudo Code Testing 133](#_Toc512684316)

[INPUT, Assignment and OUTPUT Test 133](#_Toc512684317)

[Mathematical Expressions Test 135](#_Toc512684318)

[String Expressions Test 135](#_Toc512684319)

[Boolean Expressions Test 136](#_Toc512684320)

[If And Else Statement Test 137](#_Toc512684321)

[For Next Loop Test 138](#_Toc512684322)

[Repeat Until Loop Test 138](#_Toc512684323)

[While Loop Test 139](#_Toc512684324)

[NaN Error Message Test 140](#_Toc512684325)

[Example Program from Analysis Document 140](#_Toc512684326)

[Add and Average Numbers in a Loop 141](#_Toc512684327)

[2015 Specimen Paper Question 2 Test 142](#_Toc512684328)

[Error Testing 144](#_Toc512684329)

[Unsupported syntax error message 145](#_Toc512684330)

[Unterminated String Literal error message 145](#_Toc512684331)

[Incorrect case of keywords error message 145](#_Toc512684332)

[Objectives 147](#_Toc512684333)

[The product should be in the form of an iPad application where a user can run a pseudo code program 148](#_Toc512684334)

[The user should have clear controls on how to input the program 148](#_Toc512684335)

[The user should be informed of any errors in the program 149](#_Toc512684336)

[✓ 149](#_Toc512684337)

[? 149](#_Toc512684338)

[✓ 149](#_Toc512684339)

[The program should use and enforce specific into inputted code 149](#_Toc512684340)

[The program should be simple, clear and easy to use 150](#_Toc512684341)

[Evaluation from the Client 151](#_Toc512684342)

[Current Errors in Program 152](#_Toc512684343)

[Improvements for the Future 153](#_Toc512684344)

[Links 156](#_Toc512684345)

[RegEx 156](#_Toc512684346)

[Shunting Yard 156](#_Toc512684347)

Analysis

Pseudo code is a high level description of the steps of a programming language or an algorithm. It is often used in teaching beginners to program as it is a language neutral exam and covers the underlying principles of the majority of popular programming languages.

It uses normal structural conventions of most normal programming languages but is intended for human reading rather than machine interpretation. It often omits some of the unnecessary detail that is required by machines to understand the code. It uses natural language description details rather than programming language specific keywords.

Below is an example of pseudo code. The following code will input a set of 10 grades from the user and output the average to the console.

|  |
| --- |
| Total <- 0  GradeCounter <- 1  WHILE GradeCounter < 11 THEN  INPUT NextGrade  Total <- Total + NextGrade  END WHILE  Average <- Total / 10  OUTPUT Average |

For example, if the list of inputs were:

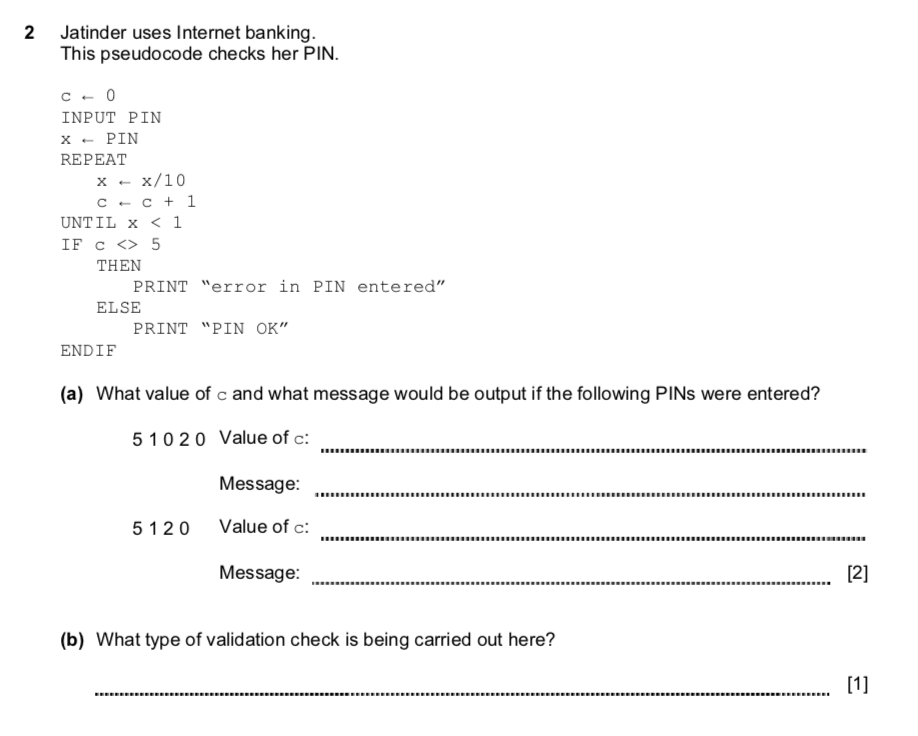
[90, 70, 64, 83, 54, 55, 61, 100, 71, 99]

At the end of the program Total would contain the value 747, GradeCounter would contain 11, and Average would contain 74.7. Therefore, the output to console would be: 74.7

# Background

As one of my GCSE options, I did Computer Science. Pseudo code was used frequently used throughout the paper as they could not specify or favour one programming language that everybody had to learn. It was used in algorithm questions and in trace table questions and I became quite familiar with it.

Below is an example question from my exam board from a past GCSE Computer Science paper:



The answers are as follows:

* 1. **Value of c:** 5  
     **Message:** “PIN OK”
  2. **Value of c:** 4  
     **Message:** “error in PIN entered”

1. A check to see if the length of the input PIN is 5 digits long.

The value of c is found by dividing x for a number of times until the value of x is less than 1. The value of c is then equal to the number of times it was divided. If c does not equal 5, it is an error, else the PIN is okay.

That being said, I found it fairly confusing.

I was taught that the idea behind pseudo code was that it was a syntax-relaxed language, where it could take any form that you wanted, as long as it resembled the structure and commands of a normal programming language. However, for my GCSE, there was a strict syntax required for the questions that were set out by the exam board. This meant that I often ended up losing marks for writing slightly incorrect answers.

|  |  |
| --- | --- |
| In the following examples, on the left there are 7 errors. The specific syntax required:   1. Capitals for keywords (INPUT, OUTPUT) 2. A <- symbol for assignment instead of an =, even though many modern programming languages use an = for assignment. 3. The MOD keyword for the remained function instead of a % operator, even though many modern programming languages use a % for the operator function. | |
| Input Total  Input N  Average = Total / N  Remainder = Total % N  Output Average  Output Remainder | **INPUT** Total  **INPUT** N  Average **<-** Total / N  Remainder **<-** Total **MOD** N  **OUTPUT** Average  **OUTPUT** Remainder |
| **X Wrong – 7 errors** | **✓ Correct** |

I also had no way of checking my answer without manually translating it into a real programming language that I knew and then running the program. This still left a lot of room for error as I could’ve made a mistake in translating.

My project will be an aid to students similar to me when I was sitting my GCSEs, to help them understand and be more confident in pseudo code questions when preparing for their exam.

# Interview

This project was suggested to me by my A Level computing teacher, Mr Macleod. Below shows an interview I conducted with him to discover more about what he wants from the project.

**Introduce yourself**

I’m Graham Macleod, I’m head of academic computing at Merchant Taylors’ School.

**Briefly outline what is the application you would like to be built**

What I’m looking for is a teaching aid for the Divisions and Fifth form. For their GCSE, they have to master Pseudo code which allows them to master a programming language. We teach them C# for programming which is satisfying in a sense because they can write programs, run them and see whether they work or not.

For the examination, they have to compose their answers in pseudo code and that is a completely paper exercise meaning that if they write out pseudo code, they have no real way of testing it as they have to give it me to be marked. It’s quite time consuming to work through pseudo code and mark it and rather frustrating for the pupils because they can’t see whether or not it works or not before they give it to me since they can’t fix small errors that they probably could have found if they had a little help.

I would like something that allows them to experiment with pseudo code, something that helps them to understand how it works and something that helps me to mark it. Those are the three main advantages.

**How much do you use Pseudo code to teach in lessons?**

It starts in the Divisions; we do it in bursts really because it can get kind of boring to be honest if you do too much of it. Pseudo code will be intermittently taught for about two months over the two years. Quite a lot of the course involves pseudo code and pseudo code skills are important for the final result. They’re probably more important than anything else.

**Why do you think Pseudo code is useful for teaching?**

I think we could teach programming without it. I think the main reason for using pseudo code at GCSE is that the exam is language neutral and is also completely paper based and so the exams are all conducted in pseudo code because the exam board doesn’t want to favour or disfavour any particular language.

**Are there any disadvantages to using Pseudo code?**

The main disadvantage is that you can’t run and that is a significant problem. It makes the whole thing seem like a fake and the students sort of get that, they appreciate that it’s not a real programming language, they can’t really run anything that they write in it, they don’t see any results, it’s just a paper exercise and as such it’s a bit dull.

**Do you ever find there are misunderstandings for some pupils between actual programming languages and pseudo code as a result of teaching pseudo code?**

There are some misunderstandings about pseudo code in that it has slightly weird rules. It’s rather old fashioned compared to C# for example, it uses structures which I’m comfortable with coming from a background of 1980’s programming languages but somebody working with modern languages like C# are going to find quite odd. You get some pupils at the end of the year who are confused between C# and pseudo code, they put bits of C# in their pseudo code and bits of pseudo code in their C# so it’s like we’re teaching them two things at the same time and it can get confusing.

**Apart from just running pseudo code in the application, are there any other features that you would find useful?**

What would be terrific but is really optional is the kind of thing that Visual Studio does where it would highlight any errors as you type. There are some quite helpful things that a modern IDE does to help you get the code right first time. Certainly just a way of checking to see that the pseudo code is correct, that it’s comprehensible, that the syntax is correct and ideally a way of actually seeing the result of a piece of code written in pseudo code.

In built exercises which automatically checks the output would be fantastic or perhaps just in built exam questions, but that’s not a priority.

**What about a translation to other languages?**

That would be nice. It wasn’t quite what I had in mind. If you could take a pseudo code that would generate C# or Swift or some other language from the pseudo code, that would be kind of cute. This is not a first priority but if it’s not deliverable, that would be okay. Equally something that would take C# and generate pseudo code automatically would also be fun to have.

**What are the existing methods you use for teaching pseudo code?**

Just pen and paper and I give pupils a cheat sheet that outlines what we need to cover.

**What would be the most important things to include into this application?**

Input and output are important, perhaps having multiple inputs separated by commas. The output, the basic print statement needs to work. It definitely needs to work in that you can concatenate string literals and variables, separating them with commas in the way the cheat sheet indicates you can do it. That also means you will have to handle number and string conversion.

The arrow for the assignment is really important, not the equals, since the single equals sign is used as an equality operator, using in if statements for example.

It would be also be good if it enforced the use of upper case for the keywords and command words like INPUT, OUTPUT, IF, THEN, ELSE, etc.

Arrays are not important, nice to have but not important.

IF, ELSE, END IF is important. Case, end case is not important. FOR NEXT and REPEAT UNTIL need to be in there. WHILE and ENDWHILE as well.

Converting to INT would be nice. Not equals to, the <> sign needs to be there as well.

Arithmetic operators are a must, perhaps the modulus operator which would be the keyword MOD.

I would be perfectly happy if it just outputs text.

Error messages are important. If there is a parsing error, if you can’t understand the pseudo code, and you’ll get a lot of that because most of the code that you’ll get will be wrong. So the most helpful error message that you can give, the quality of the error messages will be important and the ability to highlight where exactly the error has occurred will also be useful.

**What common mistakes are there that you come across with pupils when working with pseudo code?**

A particular error that you’re likely to come across is that they haven’t terminated a literal whereas you haven’t got the second set of quotes, which is super common. If’s without end if’s are also common. I would like it if you forced indentation or maybe automatically indentation depending on the structure that you find in there. The use of equals signs instead of assignment errors is common. That is actually a logical expression which would evaluate to false probably. The use of an assignment error instead of an equals sign also because people get confused about this.

Missing out quotes is quite common and an error to complain when a null variable has been used in an output command for example would be very useful.

**What would be the advantages of using this application for teaching?**

It would just be something to do. The danger with pseudo code is that it’s a lot of teacher talk and a lot of bogus exam questions which is kind of tedious. The interactive element as well will be nice; people like fiddling with computers, it’s a big part of why they do this subject and it can help them work at their own pace rather than them have to wait for the rest of the class to catch up so that they can experiment and come up with different approaches and different answers to a question without constantly having to get the teacher to tell them if they’re right or not because the device would be doing it for them.

**What sort of programs/algorithms would you use with this application?**

The level of algorithms at GCSE are sort of things like input 10 numbers and find the highest number or swap two numbers. The most sophisticated number that we do at GCSE is basically input numbers and type 0 when you’ve finished inputting numbers and output the highest, lowest and average. The pre-release material and the solutions would be useful to try in this application.

**What age or level of pupil would you use this tool with?**

GCSE pupils so 14 through 16.

**How should the user interface be designed?**

As an iPad app, it should be simple and broadly follow iPad conventions. So icons and touchscreen. Presumably bring up the soft keyboard so you can type in the code and a relatively full featured editor.

The kind of thing I’m imagining is on an iPad, a text entry window and an output window. A bit like Python where you have a text input and an output window. And a button at the top saying check which would look through and make sure it can compile it and it could run it if it wanted to and maybe perhaps apply automatic indentation at that point. Maybe a little checkbox that could turn on and off automatic indentation if you wanted to which would hopefully get pupils into good habits.

# Evaluation of Interview

From my interview with my client, I have learnt about exactly what the problem is that needs solving and how the product should be built. It will be in the form of iPad application which should be aimed at GCSE students with simple and clear to use controls. It should be informative, describing exactly what each part of the code means, exactly the reason behind a syntax error and exactly why the output was displayed in the way it was. From the information I learnt in the interview, I have created the following set of objectives:

# Objectives

1. **The product should be in the form of an iPad application where a user can run a pseudo code program:**
   1. The app should have an input panel where the user can input a program.
   2. The app should have an output panel, displaying the output of the program typed into the input panel.
2. **The user should have clear controls on how to input the program:**
   1. There should be buttons alongside the keyboard to input keywords into the program.
   2. There should be provided instructions on the correct syntax for the pseudo code being used.
3. **The user should be informed of any errors in the program:**
   1. The app should highlight exactly which line is incorrect.
   2. The app should provide reason(s) why the line is incorrect.
   3. The app should show ways to correct the code in detail.
4. **The program should use and enforce specific into inputted code:**
   1. Enforce the use of the “<-“ arrow for assignment.
   2. INPUT, OUTPUT commands must be supported.
   3. IF, ELSE, END IF conditionals must be supported.
   4. Not equals to <> command must be supported.
   5. FOR NEXT, REPEAT UNTIL loops must be supported.
   6. WHILE and END WHILE loops should be supported.
   7. Conversion to INT must be supported.
   8. Arithmetic operators must be supported, including the MOD command.
5. **The program should be simple, clear and easy to use:**
   1. This app is aimed at GCSE students, aged 14 to 16, it should be designed in a friendly way.
   2. The app should be clearly laid out, with very obvious instructions and prompts of how to use.
   3. The app should follow iPad and iOS conventions.

# Pseudo Code Cheat Sheet for CIE iGCSE and AQA A-Level

Since there is no standard syntax for pseudo code, my client has given me the following the following set of rules. It is a cheat sheet revision guide that he gives to all his pupils in preparation for the exams.

***General Rules:*** *Use CAPITALS for command words. Only initialize variables that don’t get assigned (for example running totals). Use the <- arrow for assignment.*

## **Basics**

|  |
| --- |
| **INPUT**  INPUT Name, Address, Telephone  You don’t need to have declared these variables and you don’t need to worry about type. Just separate them with commas and the variables will be created with the input, in order. |
| **OUTPUT**  PRINT “Hello”, stringVar, intVar  You don’t need to worry about type, just separate variables with commas. You can include text in quotes, again – separate with commas from the variables. |
| **CREATING AND ASSIGNING VARIABLES**  MyTotal <- 0  MyString <- “Sausages”  You can drop these in wherever you like. Give them sensible names. Use the arrow to show assignment. |
| **IF .. THEN .. ELSE .. END IF**  IF Value <= 26 THEN  PRINT “Bigger”  ELSE  PRINT “Smaller”  END IF  This is different from what you’re used to and you are guaranteed that at least one will be needed. Make sure you’re practised in making these and be careful to include the END IF. |

## **Loops**

|  |
| --- |
| **FOR … NEXT**  For Counter <- 1 TO 30  PRINT “This is “, Counter  NEXT  FOR … NEXT loops are nice and simple and very useful. The Counter variable (you can call it what you like) is created for you and it’s automatically incremented each time. You can read its value inside the loop, but you shouldn’t try to change it. |
| **REPEAT … UNTIL**  C <- 0  REPEAT  INPUT number  C <- C + 1  UNTIL C > 9  REPEAT … UNTIL loops are more complicated to make than FOR … NEXT. You must set up the initial counter variable and be sure to increment it inside the loop. These loops should only be used when you don’t know in advance how many times to loop. |
| **WHILE … END WHILE**  C < - 0  WHILE C > 9  INPUT number  C <- C + 1  END WHILE  WHILE … END WHILE loops are very similar to REPEAT … UNTIL loops, with the difference that the test is performed at the beginning, not the end. This means that it’s possible that the loop won’t execute at all, whereas for REPEAT loops it will always execute at least once. |

## **Other bits and pieces**

|  |  |
| --- | --- |
| **CONVERTING TO INT**  The following code checks if a number is an integer, by using INT to get the integer part and seeing if anything is left.  Difference <- INT(number) – number  IF Difference = 0 THEN  PRINT “Integer”  END IF | **NOT EQUAL TO**  Use <>  IF Value <> 0 THEN  PRINT “Value is not 0”  END IF |

# Example Program

My client initially provided me with this sample program below so I could figure out how to go about creating the application. He said that this is a good example of the types of program he would be using this application for and that programs would not get more complicated than this for use in GCSE teaching. I have used this program in all my designs and modelling below.

INPUT HowMany

Total <- 0

FOR I <- 0 TO HowMany

Total <- Total + i

NEXT

IF Total < 10 THEN

OUTPUT ‘Hello’

ELSE

OUTPUT ‘Goodbye’

END IF

This excerpt of code makes use of several key features which were included in my objectives. It includes an INPUT for inputting external variables from the user into the program, assignment to new and already existing variables, a FOR loop for repeating code a set number of times, IF and ELSE statements to logically decide which block of code to execute based on a Boolean expression and an OUTPUT command to print text to the console.

# Existing Methods of Teaching Pseudo Code

As my client said during our interview, the only existing method that he uses for teaching pseudo code to his GCSE students are pen and paper exercises. The students then hand in the exercise into him for marking, all of which is an incredibly tedious process.

After research online, I found that there are very few interactive tools online for teaching pseudo code to students.

**BBC Bitesize**

*Bitesize* is the BBC's free online study support resource for school-age students in the United Kingdom. It is designed to aid students in both schoolwork and, for older students, exams.

*Bitesize’s* approach for teaching pseudo code to students is a brief one-page document as part of a lesson on algorithms. It starts with a short explanation of what pseudo code is, simple explanations of each of the keywords included in common syntax and then an example program.

There is then 10 question multiple choice quiz included, only 2 of which are about pseudo code (the correct answer in bold):

* What is pseudo code?
  + **A way of describing a set of instructions in text form.**
  + A specific programming language that all computers use.
  + A diagrammatic representation of a set of instructions
* How would a condition loop be created when writing pseudo code?
  + With a FOR loop
  + With a WHILE loop
  + **With a WHILE loop or a REPEAT-UNTIL loop**

This method of teaching is incredibly brief and certainly does not give pseudo code nearly enough attention as it deserves. During the interview, my client said:

“quite a lot of the course involves pseudo code and pseudo code skills are important for the final result. They’re probably more important than anything else.”

For one of the most important skills in a Computing GCSE Exam, this is not nearly sufficient enough to be a useful resource.

**Code2Flow**

*Code2Flow* is a Pseudo-code to flowchart tool. The user types code into the panel on the left and a flow chart appears on the panel on the right (as shown in the image below).

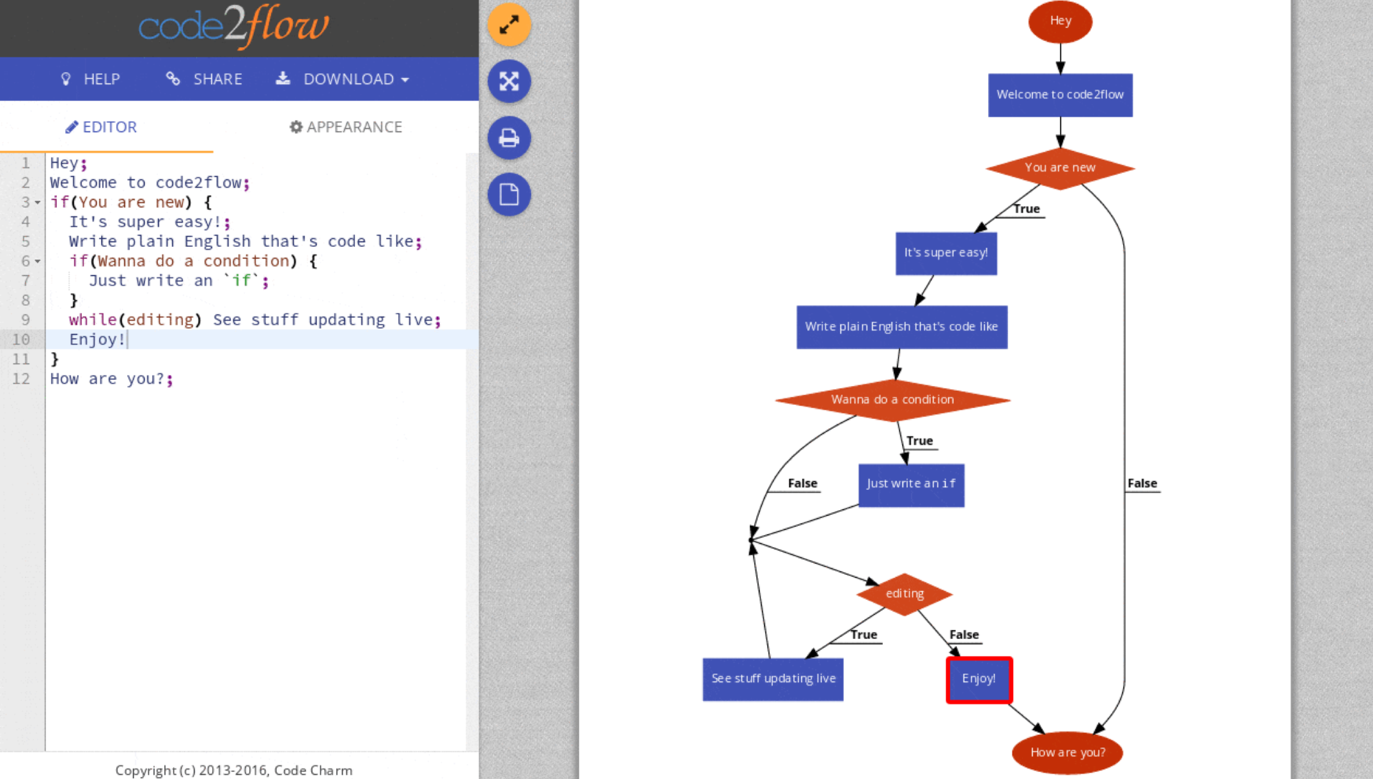
This program is very useful to students for visualising a program when starting to learn to code, especially that flowcharts are frequently used in exams too. Students find flowcharts much easier to follow than code. There is no specific required syntax apart from loops and conditionals. All other text will appear in a blue box saying the text that was inputted on that line. For example, students can write:

Add 5 to the Students array

Rather than:

Students[Students.Length – 1] <- 5

However, *Code2flow* has some disadvantages. Mainly, the code is not executable, there is no resulting output and there is no option to input values from the user. The only purpose of this program is to create a visual representation of the code that is written. There is also no checking of a specific pseudo code syntax. This would not be useful for the GCSE exam that my client is preparing his students for.

****

# iOS Applications and Swift

My client and I have chosen to use an iOS application, coded in Swift using the program Xcode to develop for this project. I have a number of years coding experience using iOS apps and Swift from my own interest.

A tablet application would be suitable for this project, especially as it is aimed at GCSE students (aged 14 to 16) and has an objective to be interactive and easy to use. Many students this age are familiar with how iPads works and use iPads to play games and complete many daily tasks.

To write iOS applications, Apple provides their own programming environment called Xcode and their own programming language called Swift. Apple describes Swift on their website:

Swift is a powerful and intuitive programming language for macOS, iOS, watchOS and tvOS. Writing Swift code is interactive and fun, the syntax is concise yet expressive, and Swift includes modern features developers love. Swift code is safe by design, yet also produces software that runs lightning-fast.

Swift was released in 2014, so hasn’t been around for very long but has already caught on in a big way with developers. Stack Overflow conducted a developer survey in 2015 and Swift was voted as the most loved programming language, just a year after it was released. Swift has been praised for its easy to use syntax, its speed and some of its features.

## Swift Syntax

Swift syntax for defining variables is using either the let or the var keyword. One should use let if the variable is a constant and var if the variable is changed or mutated at any point in the program. Types are inferenced in Swift from the result of the variable. So in the following example, the aStringConstant is never changed but the aStringVariable is changed to add the constant to the end.

let aStringConstant = “Jake!”

var aStringVariable = “Hello, “

aStringVariable += aStringConstant

print(aStringVariable) **// “Hello, Jake!”**

Swift syntax does not require the use of a semicolon to terminate a line of a string, unless two statements are on the same line.

For string concatenation, either of the following syntaxes can be used:

let name = “Jake”

let greetingA = “Hello, “ + name **// Hello, Jake**

let greetingB = “Hello, \(name)” **// Hello, Jake**

The \(value) syntax can be placed inside a string literal where value is the name of a variable to be inserted at that point within the string.

### Methods

Methods are defined using the following syntax:

func someMethod(strVar: String, intVar: Int) -> [String] {

// code with a return

}

let array = someMethod(strVar: “abc”, intVar: 123)

Where someMethod is the method name, strVar is a parameter of type String, intVar is a variable of type Int and the return value is of type String array.

### Optionals

One important feature in Swift is optionals, where a value may be null or not. This means that non-optional types cannot result in a null-pointer error. For example, String can have two types: String or String?, where String? is the optional form of the type String. To access the value inside an optional type, assuming it is not null, it must be unwrapped using the ! operator. For example:

let myValue = anOptionalInstance!.someMethod()

This will unwrap the value of anOptionalInstance which is an optional type. If anOptionalInstance is null and the ! operator is used, then the compiler will throw a null-pointer error. There is also a feature called optional chaining to test whether the value is not-null and then unwrap if it is not-null:

let myValue = anOptionalInstance?.someMethod()

In this case, someMethod() will only be called if anOptionalInstance is not null. If it is null, then myValue will result in null.

The ?? operator in Swift is used when returning a value. It takes left and right hand side parameters and will return the left one if it is not null. If it is null, then it will return the right one. These can be chained together as many times as desired. For example:

let a = null

let b = null

let c = “Finally, not null!”

let d = a ?? b ?? c

print(d) **// Finally, not null!**

### Types in Swift

Type casting can be used with optionals to change the the type of an object. All objects in Swift can be expressed as the type Any, so the following could be expressed in Swift where there is currently an object of type Any which is actually a String object called object.

let str = object as? String

print(str!)

Another feature of Swift is type inference where the type does not have to be defined in the code if the compiler can inference it from the definition.

The method type(of: ) returns a string of type of the object which is passed in. For example, the following code would have these results:

let str = “Hello!”

print(type(of: str)) **// String**

let num = 123

print(type(of: num)) **// Int**

However, sometimes types do need to be specified. In the above example, the type of num was guessed to be an integer. However, we may have wanted it to be a double which was a round number. In that case, we can specify the type like this:

let num: Double = 123

print(type(of: num)) **// Double**

With type inference in Swift, class names for static methods often can be removed. In iOS, the UIColor object represents colours. The normal way to access standard colours is a static variable on the UIColor object:

let black = UIColor.black

However, with type inference, we can remove the UIColor type if the variable is already of type UIColor:

let black: UIColor = .black

Or, if we had a method defined as set(color: UIColor) which took a parameter UIColor, the following would be valid:

set(color: .black)

This method of removing the type name can also be applied to enums.

### Conditionals

Swift also has some powerful conditional capabilities. A constant can be defined within an if statement to check for not-null. For example, if there is an optional value, an if statement can be used to redefine the constant as an unwrapped, non-optional value within the scope of that if statement.

let object: String? = “abc”

if let object = object {

print(object) **// abc**

}

This will only print the variable object if it is non-null. Outside of the scope of the if statement, object is of type String?. However, inside the scope of the if statement, object is of type String and therefore does not need to be unwrapped with the ! operator.

Swift also has a featured called guard statements which can be thought of as the opposite to an if statement. It will evaluate the Boolean expression and run the code after the guard statement if it is true, else run the code inside the block. The code inside the guard statement will usually contain a return statement and Xcode will show a warning if a return statement is not called inside here. Guard statements are usually used for ‘guarding’ against errors. For example:

let error: Error? = nil

guard let error = error else {

return

}

print(“Success!”) **// Success!**

### Enums and Associated Values

Swift has a feature called associated values for enumerations. This means that you can pass a value along with a case in the enum. For example, we define the enum Result as a case called success with no associated values and a case called error which contains a String associated value which is the message of the error.

enum Result {

case success

case error(String)

}

To pass in an associated value, simply put the value in brackets after the case.

let result: Result = .error(“An error occurred!”)

To access the associated value, it can either be done in a switch statement, where message is a variable of type String containing the associated value.

switch result {

case .success:

print(“Success!”)

case .error(let message):

print(“Error: \(message)”) **// Error: An error occurred!**

}

Or an if let statement can be used to get the associated value:

if case let .error(let message) = match {

print(“Error: \(message)”) // Error: An error occurred!

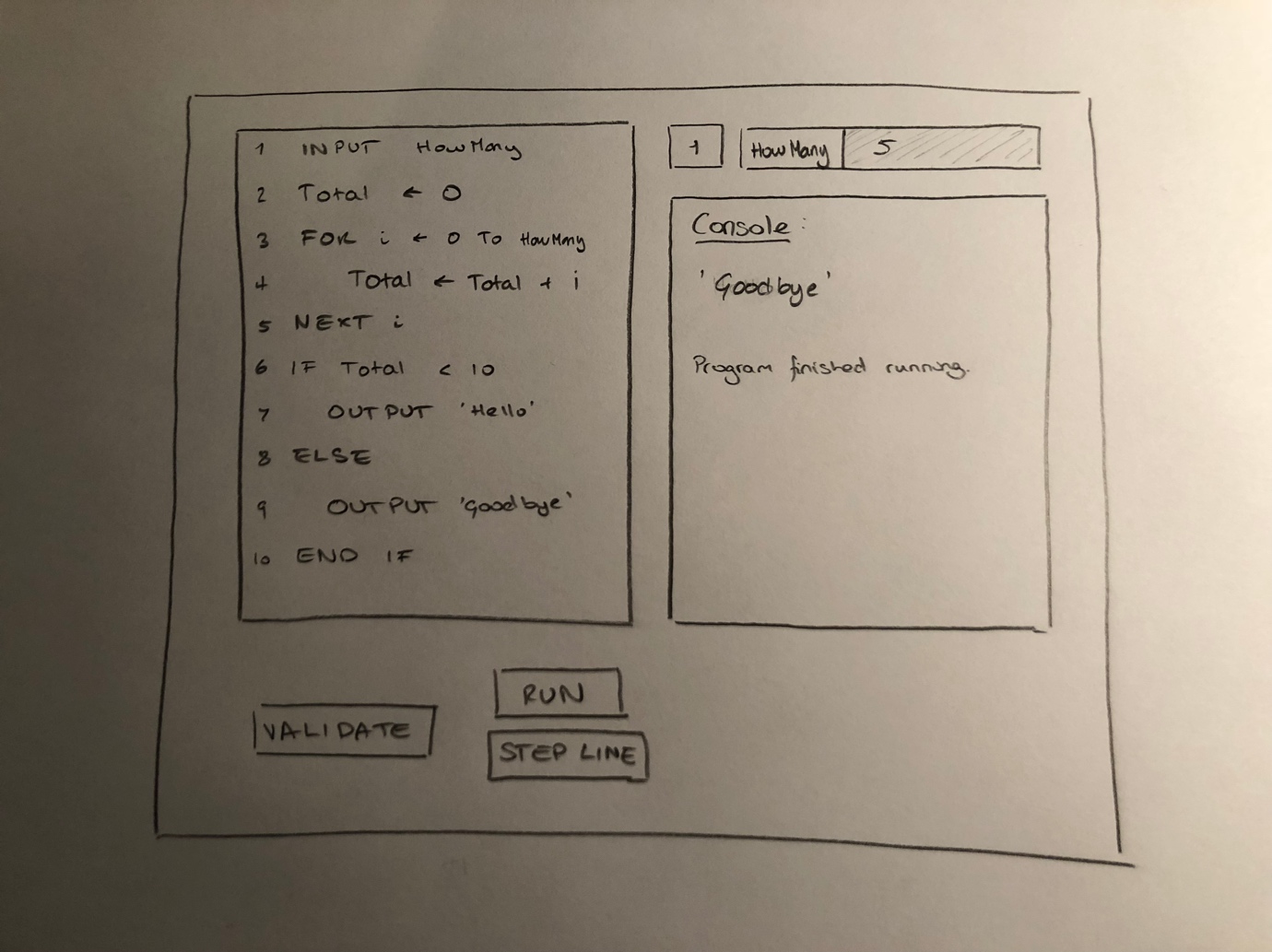
}

### Other information

* A constructor is usually called an initializer in Swift. An optional initializer is any initializer which may or may not return null, dependent on the parameters passed in.
* To reference a property within it’s own class, use the self keyword.
* If there are no local variables with the same name as a property, or the code is not referencing a property within a closure, the self keyword is not needed.
* The syntax for an array is the type of each element surrounded by square brackets, e.g. [String]. These can be compounded to have arrays of arrays [[String]]. Arrays in Swift has no capacity, therefore the concept of Lists do not exist in Swift.
* The syntax for a dictionary is [String : URL] where String is the type of the key and URL is the type of the value. These can be compounded to have dictionaries within dictionaries: [String : [String : Int]].
* The syntax for a for loop is for index in 0…100 where index is the name of the variable containing the index, 0 is the start index and 100 is the end index. The equivalent of this in pseudocode would be FOR index <- 0 TO 100.
* The syntax for a foreach loop is for item in array where item is the object passed into the loop and array is the array which contains object at some index within itself.
* A protocol in Swift defines a blueprint of methods, properties, and other requirements that suit a particular task or piece of functionality. The protocol can then be adopted by a class, structure, or enumeration to provide an actual implementation of those requirements. Any type that satisfies the requirements of a protocol is said to conform to that protocol.
* There is a feature in Swift called Automatic Reference Counting (ARC). This is a feature which means that strong and weak references to objects do not need to be specified. The running program will automatically release objects when they are no longer in use and can no longer be accessed.

# Designs

As my client said, the designs needed to be fun, clear and easy to use. I came up with some sketches of how the iPad app should look on paper below.

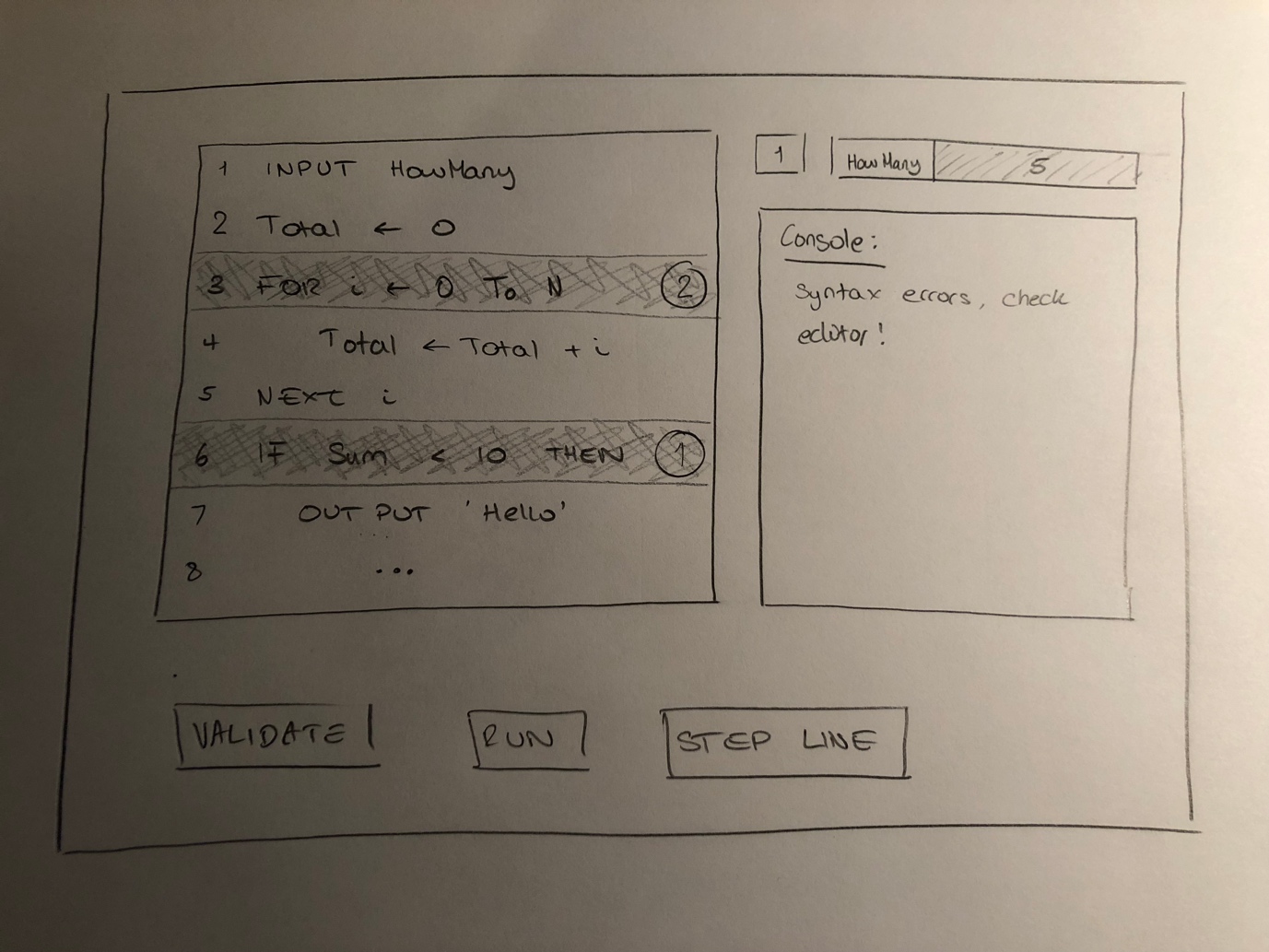
****

This is how the iPad app will look, including the sample code that my client previously gave me. There are two main panes: one for inputting code (the editor) and one for the output of the code (the console). There are also fields which appear after validation of the pseudo code to decide all the inputs that are included throughout the program.

There are 3 buttons at the bottom of the screen: validate, run and step line. Run and step line are disabled on app launch and any time there are changes made to the code. Once code has finished being entered by the user, they will select the validate button to check the program for errors. If the program is valid, the console will say ‘Program ready to run’ and the run and step line buttons will be enabled. The run button will run all the code one by one instantly and indicate when it is finished. However, the step line button will run the code line by line, indicating clearly which line it has already executed so that the user can see exactly why output appears in the console due to which line in the code.

This code has already been run and has shown the output of the code to the console and a message to say that the program has finished execution.

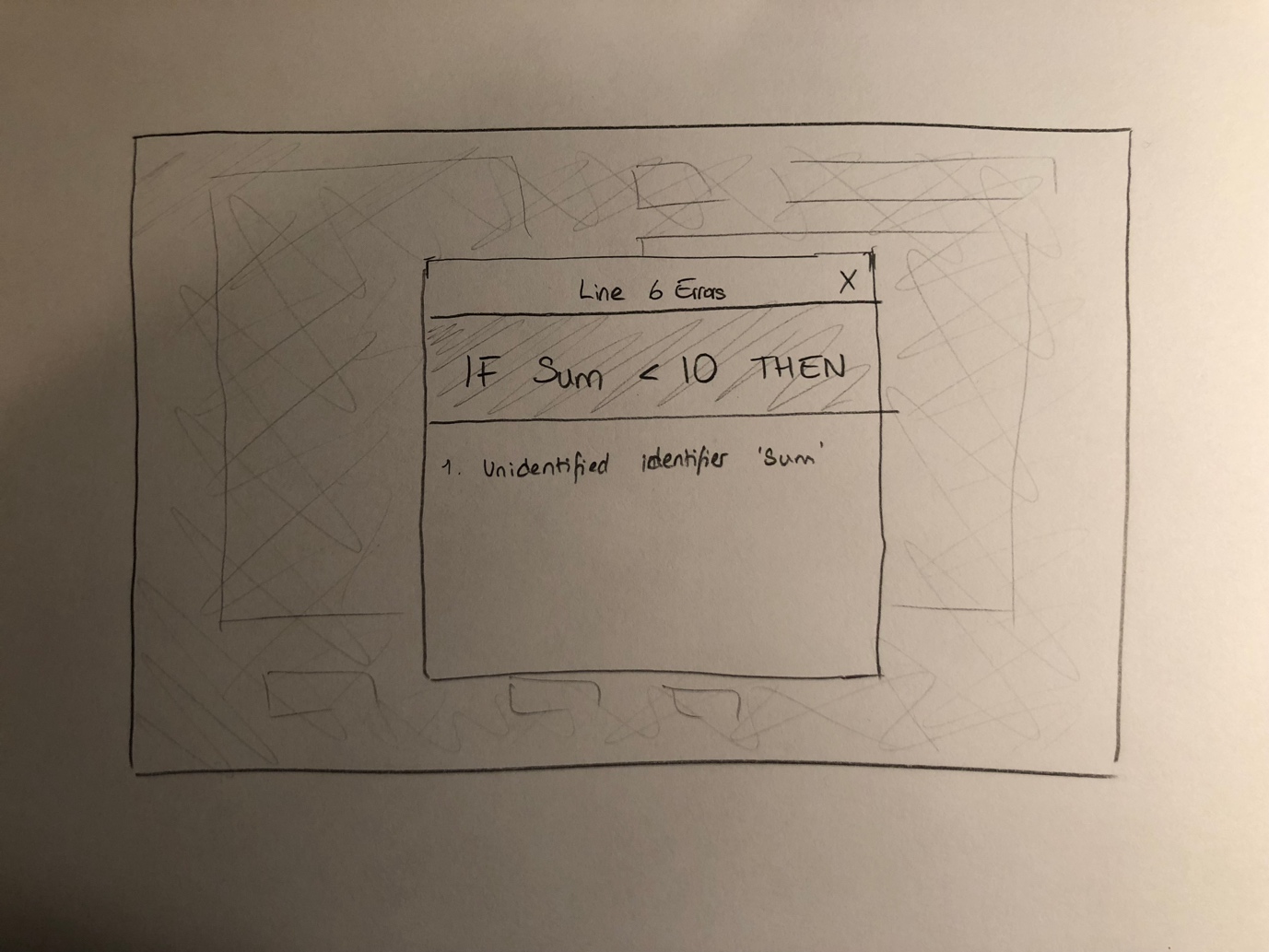
This code will run with HowMany being 5, which will result in the Total variable being equal to 15 at the end of the for loop. Therefore, it will run the Goodbye output because it failed the if statement expression since 15 is greater than 10.



This screen shows when the user has clicked validate after inputting code but there were errors in the inputted code. The run and step line buttons are currently disabled and the console clearly shows that there are errors in the program so therefore it cannot run.

The editor clearly highlights which lines includes errors and how many errors there are in the program. For example, on line 3 of the program it is a for statement. Firstly, the To is incorrect as all keywords in pseudo code should be capitalised and therefore the To should be actually a TO. Secondly, it has used a variable N as the upper limit of the for loop. There is no variable yet created called N (the user probably meant to use the variable HowMany) and therefore there is a unexpected identifier error since N does not exist. There were 2 errors in this line, so therefore the line is highlighted and a 2 is displayed on the far right of that line.

The 2 in a circle is a button, and will display the next pop up screen when pressed…



This screen will pop up as an overlay over the rest of the program and shows exactly the reason for each of the errors. The user here has pressed the 1 button next to line 6. Here the user has the variable Sum (he probably meant to use the variable Total), therefore it will show the line code and show that there is an unexpected identifier in the line called Sum.

This popup can easily be dismissed by pressing the close button in the top right corner of the pop up.

# Modelling

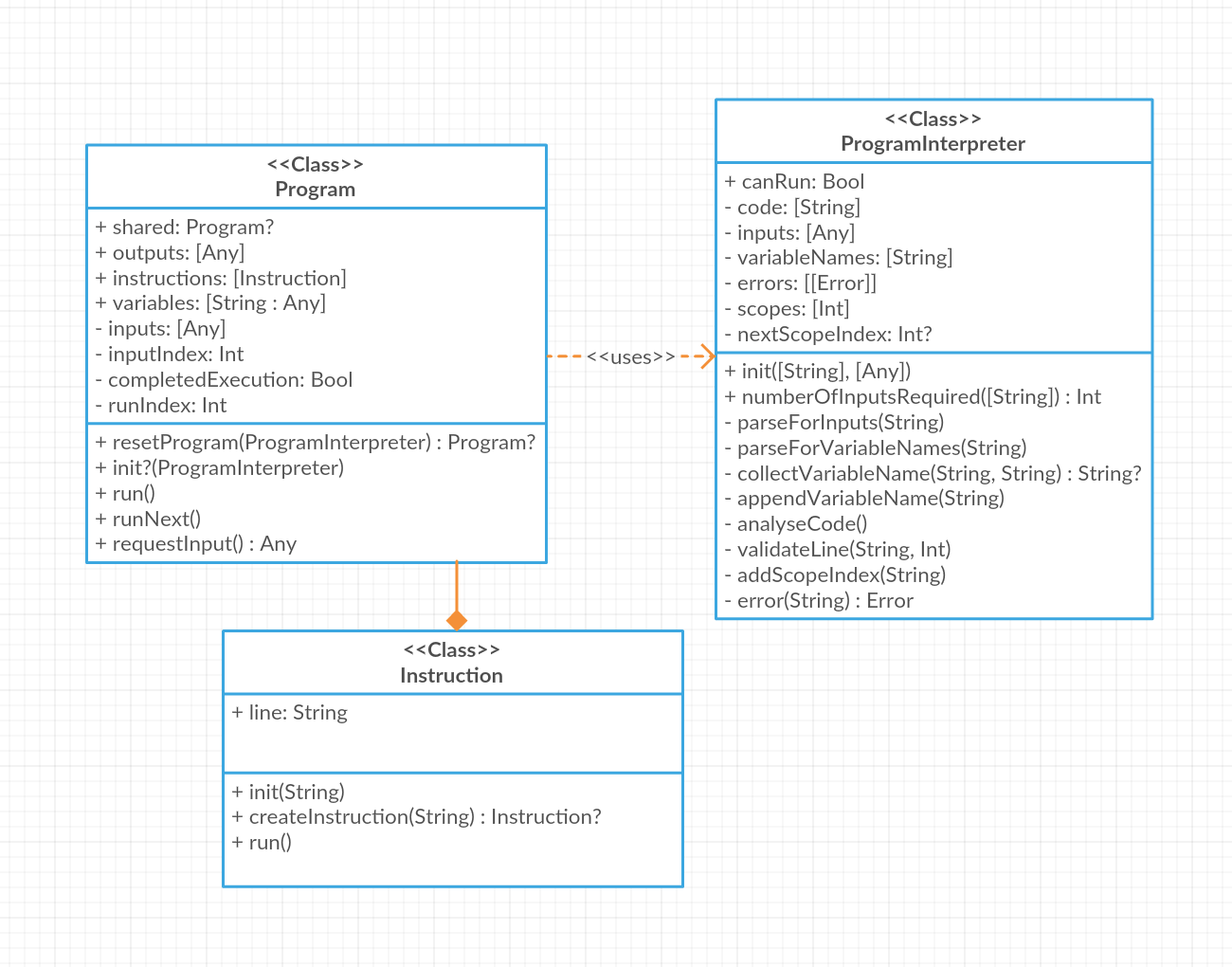
A pseudo code interpreter will be essentially an input of an array of strings (each line of code) and will have to be analysed, understood and manipulated to create a storage of variables, logic to decide which section of code to run and a mix of output to the console.

## Class Diagram

There will be three stages to running a pseudo code program. First will be the validation of the code, then the processing of the instructions that each line of code represents and then the execution of the program that all of the code represents.

|  |  |  |
| --- | --- | --- |
| **Validation** | **Processing** | **Execution** |

The validation of this code will be done by an interpreter this can be represented by a class called ProgramInterpreter. The processing of each line of code can be done in a class called Instruction, where each instance of an Instruction object represents each line of code. Finally, the execution will be done by a class which will be called Program. The variables and methods that will be required are illustrated by the class diagram below:



Here the Program class uses the ProgramInterpreter class to initialize the program to be ready for execution. The Program object also contains an array of Instruction instances which represent each line of code in the program.

## Data Dictionary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field Name** | **Field Purpose** | **Field Type** | **Field Size** | **Example Data** |
| Code | Stores the lines of code that the user has entered. | List of type String | ∞ | [“INPUT MyVar”, “OUTPUT MyVar”] |
| Inputs | Stores the set of inputs that the user has entered which should be used within the executable program represented by the code in the “Code” field. | List of type Any | ∞ | [123, “Abc”, TRUE] |

## Regular Expressions (Regex)

I have decided to use Regex to analyse and manipulate the strings. Regex is short for regular expression and is a computer science theory that define a search pattern within a sequence of characters. It was developed by an American mathematician Stephen Cole Kleene in the 1950s and quickly came into common use in Unix text-processing utilities. Today, regular expressions are widely used in search engines, search and replace dialogs of word processors and other text processing utilities.

Regular expressions can be used to validate strings to check that they conform to a specific syntax and can be used to extract values from a string at a specific location. This is very useful for my program as I can use it to validate the syntax of lines of code and extract values from these lines of code to manipulate the variables in memory.

Regular expressions have an input text and a string pattern. The input text can be anything whereas the string pattern follows a specific regular expression set syntax. Below is a brief overview of the syntax for regular expression patterns (source: regexr.com):

|  |  |
| --- | --- |
| **Character Classes** | |
| . | any character except newline |
| \w \d \s | word, digit, whitespace |
| \W \D \S | not word, digit, whitespace |
| [abc] | any of a, b or c |
| [^abc] | not a, b or c |
| [a-g] | character between a & g |
| **Anchors** |  |
| ^abc$ | stard/end of the string |
| /b /B | word, not-word boundary |
| **Escaped characters** | |
| \. \\* \\ | escaped special characters |
| \t \n \r | tab, linefeed, carriage return |
| \u00A9 | unicode escaped Ⓒ |
| **Groups & Lookaround** | |
| (abc) | capture group |
| /1 | backreference to group #1 |
| (?:abc) | non-capturing group |
| (?=abc) | positive lookahead |
| (?!abc) | negative lookahead |
| **Quantifiers & Alternation** | |
| a\* a+ a? | 0 or more, 1 or more, 0 or 1 |
| a{5} a{2,} | exactly five, two or more |
| a{1,3} | between one & three |
| a+? a{2,}? | match as few as possible |
| ab|cd | match ab or cd |

These in combination can be used to create some very powerful pattern recognition. For example, I found through my research and own interest, the standard regular expression pattern for an email address:

(?:[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+(?:\.[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+)\*|"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])\*")@(?:(?:[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?|\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]\*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\])

The capture groups that are mentioned in the cheat sheet table above are used to extract values from an expression. For example, if I had the string and pattern:

**Pattern:** Hello, my name is (\w+)

**String:** “Hello, my name is Jake”

The first capture group would return the value “Jake” with a range of location 18 and length 4. In my project, I am going to use these capture values to extract values from lines of code and to substitute existing values using these ranges of capture groups that are provided using regular expressions.

## Shunting Yard Algorithm

My program is going to need to understand mathematical expressions for adding, subtracting, multiplying and dividing different variables together. I have chosen to use the Shunting Yard algorithm for this.

Shunting Yard is a method for parsing mathematical expressions specified in infix notation. It will produce a postfix notation string which is much easier to calculate the final value than infix notation, which is the mathematical notation that most people are used to. The Shunting Yard algorithm was invented by Edsger Dijkstra and named “shunting yard” because its operation resembles that of a railroad shunting yard.

For example, the shunting yard algorithm would turn the left into the right in the following examples:

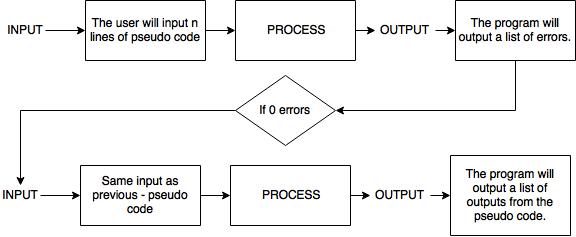
|  |  |
| --- | --- |
| **Infix Notation** | **Postfix Notation** |
| 3 + 4 | 3 4 + |
| 3 + 4 x ( 2 – 1 ) | 3 4 2 1 – x + |

Design

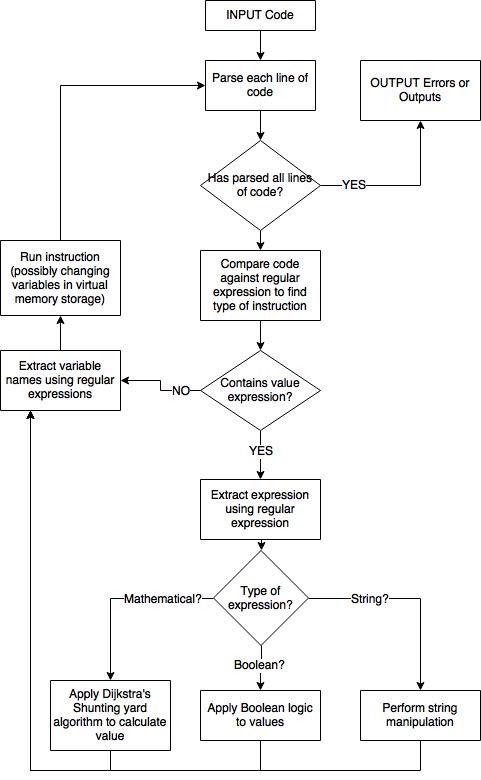
The overall program uses an input of the lines of pseudo code which the user types into the editor. This will be an array of strings where each item in the array represents a line of code written by the user. My code will process the inputted code array using various algorithms and methods to simulate an interpreter validation to check whether the code is valid or not. It will then produce an output of a list of errors for each line.

If there are no errors, the interpreter will mark the code as valid. Then my code will process the inputted code using additional algorithms and methods to simulate an interpreter execution and will process the pseudo commands inputted by the user and produce an array of outputs from the pseudo code program.

Below is an overview of the entire process:

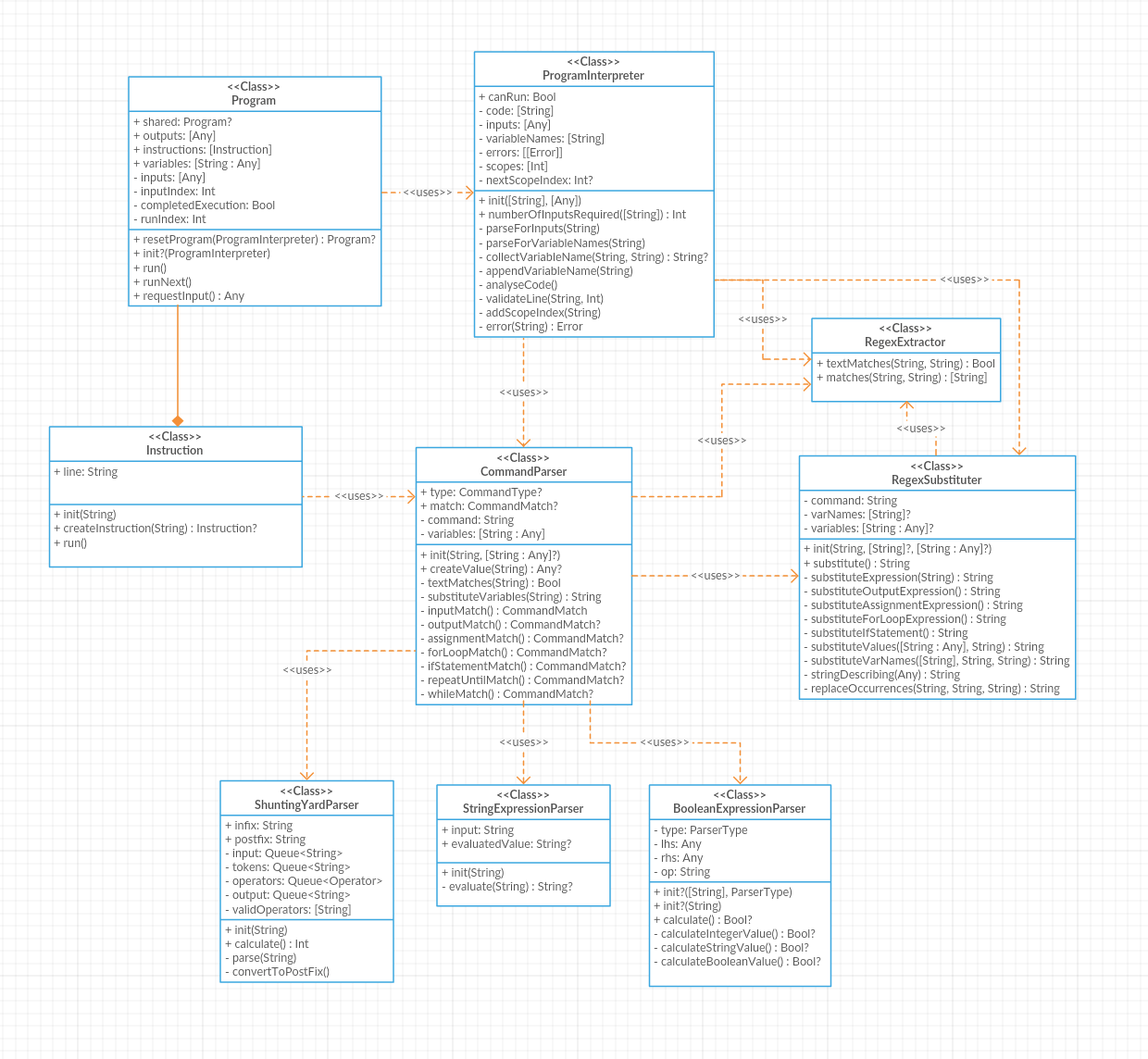


The algorithms and methods involved in processing the inputted pseudo code include Dijkstra’s Shunting Yard algorithm and the use of regular expressions. On the next page is a diagram which displays the summary of steps for producing an output:



The three main classes for running a pseudo code program within the application are ProgramInterpreter, Program and Instruction.

Below is a class diagram showing how most of the main classes interact with each other within my application:



The code begins with an initialization of a ProgramInterpreter object, requiring a string array of the lines of code and an array of all the inputs required for use in the program.

Then, the program interpreter object is passed into an optional initializer for the Program class. (An optional initializer in Swift is an initializer which may return null). In this case, the Program initializer will return null if the ProgramInterpreter has failed its validation of the code.

The Instruction class is also a key class to how the program works. Each Instruction object represents one line of code and subclasses of the Instruction class contains specific and relevant data/logic for each executable line of code.

|  |
| --- |
| **ProgramInterpreter** |
| + canRun: Bool  - code: [String]  - inputs: [Any]  - variableNames: [String]  - errors: [[Error]]  - scopes: [Int]  - nextScopeIndex: Int? |
| + init([String], [Any])  + numberOfInputsRequired([String]) : Int  - parseForInputs(String)  - parseForVariableNames(String)  - collectVariableName(String, String) : String?  - appendVariableName(String)  - analyseCode()  - validateLine(String, Int)  - addScopeIndex(String)  - error(String) : Error |

|  |
| --- |
| **Program** |
| + shared: Program?  + outputs: [Any]  + instructions: [Instruction]  + variables: [String : Any]  - inputs: [Any]  - inputIndex: Int  - completedExecution: Bool  - runIndex: Int |
| + resetProgram(ProgramInterpreter) : Program?  + init?(ProgramInterpreter)  + run()  + runNext()  + requestInput() : Any |

|  |
| --- |
| **Instruction** |
| + line: String |
| + init(String)  + createInstruction(String) : Instruction?  + run() |

The ProgamInterpreter class will check and validate the code. If the code is valid and can run without problems, a Program object can be initialized. The Program object will contain a property called outputs which is a string array of each output made from the program.

If the code is not valid, it will return nil. In this case, the ProgramInterpreter instance has a property errors which will be of type [[String]] for possible multiple errors for each line with a string description of what the syntax error was.

A common design pattern in Swift and iOS development is called singletons. This is where a single object of a type of class is shared throughout all parts of the program. This is done by having a static variable of the type of the class on the class that it is.

In this case, there is a static variable called shared on the Program class which is of type Program?. The resetProgram method will create the Program class and set it to this Program singleton – shared.

|  |
| --- |
| Swift |
| let interpreter = ProgramInterpreter(code: code, inputs: inputs)  let p = Program.resetProgram(withInterpreter: interpreter)  if let program = p {  program.run()  **// Print outputs from program.outputs array**  } else {  **// Print errors from interpreter.errors array**  } |
| Pseudocode |
| interpreter <- ProgramInterpreter(code, inputs)  p <- Program.resetProgram(interpreter)  IF program != null THEN  p.run()  **// Print outputs from program.outputs array**  ELSE  **// Print errors from interpreter.errors array**  END IF |

# ProgramInterpreter Class

Here is the class definition for ProgramInterpreter again:

|  |
| --- |
| **ProgramInterpreter** |
| + canRun: Bool  - code: [String]  - inputs: [Any]  - variableNames: [String]  - errors: [[Error]]  - scopes: [Int]  - nextScopeIndex: Int? |
| + init([String], [Any])  + numberOfInputsRequired([String]) : Int  - parseForInputs(String)  - parseForVariableNames(String)  - collectVariableName(String, String) : String?  - appendVariableName(String)  - analyseCode()  - validateLine(String, Int)  - addScopeIndex(String)  - error(String) : Error |

In the initializer for the ProgramInterpreter class, all class properties are set and the errors property is initialized with an array of empty arrays for each line. A method to analyse each line of code is then called. A for each loop goes through each line of code and does four things:

1. Firstly, it parses the line of code for variable names that are used in the program for an INPUT command using the RegexExtractor class. If the line matches an INPUT command, the variable name is extracted from the line and is added to a private property called variableNames within the ProgramInterpreter class.

|  |
| --- |
| **Example – Variable name parsing** |
| If the line: “INPUT MyVariable” was parsed, the string ‘MyVariable‘ will be added to variableNames. |

1. Secondly, it parses the line for assignment variables (this could be a normal assignment of a variable or an assignment for an index variable within a for loop, while loop, etc.) using the RegexExtractor class. If the line matches an assignment command, the variable name is extractor and added to the variableNames array, if it does not already contain that variable name.

|  |
| --- |
| **Example – Assignment variables parsing** |
| If the line: “Total <- 0” was parsed, the string ‘Total’ will be added to variableNames. |

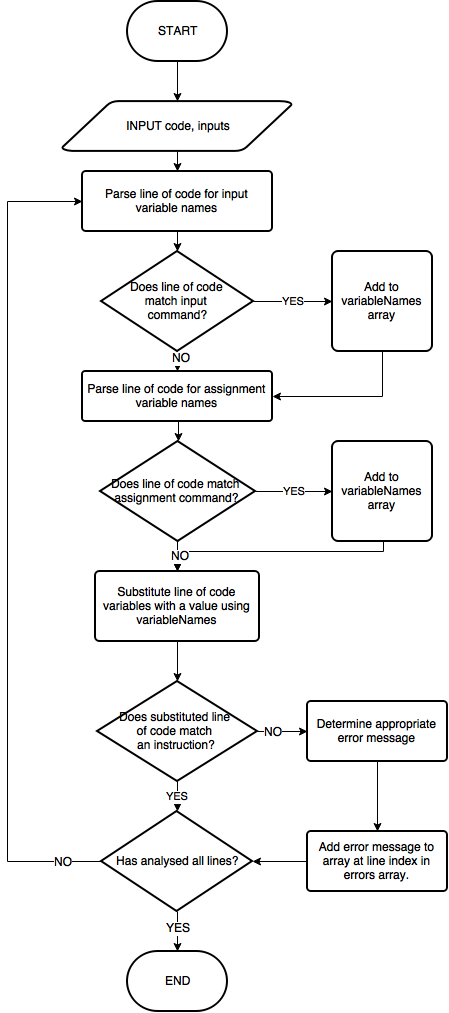
1. Thirdly, it will substitute all variables for that line of code using the RegexSubstituter class. It will substitute with variables that currently exist in the variableNames array at that point, with a 0 value. This is so that the line can be validated in the next step of the interpreter’s algorithm, since the values of the variables do not matter yet at this point. Read the example below for an explanation.

|  |
| --- |
| **Example – Variable substitution** |
| INPUT A  INPUT B  C <- A + B  At the third line, the variableNames would contain [“A”, “B”].  *(The variable C would be also contained in the list of variable names since parsing for variable names first, but for now we will ignore it).*  Therefore, for the third line, it would be transformed into this line:  C <- 0 + 0  *(Even though C would be contained in the variable names list, the RegexSubstituter class does a specific substitution for each type of command. For an assignment command, it will only substitute values for the right hand side of an assignment expression.*  If, however, the program was:  INPUT A  C <- A + B  The line would be transformed into:  C <- 0 + B  This is important for the next step in the interpreter. |

1. Finally, the interpreter uses the CommandParser class to validate each line for a match of an instruction. If the command parser returns nil for the transformed line of code, then the code is run through specific checks inside the program interpreter to determine the issue and provide a specific error message. This error message is added to the array at the line number index within the errors property of the program interpreter. However, if the command parser does not return nil, then the nothing else is executed for this line and the next line is analysed. The implementation of the CommandParser class is explained later in this document.

The program interpreter also contains a computed Boolean property called canRun. This is calculated by creating a sum of the length of each array within the errors array. The canRun property then returns whether the sum is greater than 0. This is shown below:

|  |
| --- |
| Swift |
| var canRun: Bool {  var errorCount = 0  errors.forEach({ errorCount += $0.count })  return errorCount == 0  } |
| Pseudocode |
| FUNCTION getCanRun() RETURNS Bool  errorCount <- 0  FOR i <- 0 TO errors.length-1  lineErrorArray <- errors[i]  errorCount <- errorCount + lineErrorArray.length  NEXT i  return errorCount == 0  END FUNCTION |

This flowchart illustrates the full algorithm for the program interpreter.

# Program Class

Here is the class definition for the Program class again:

|  |
| --- |
| **Program** |
| + shared: Program?  + outputs: [Any]  + instructions: [Instruction]  + variables: [String : Any]  - inputs: [Any]  - inputIndex: Int  - completedExecution: Bool  - runIndex: Int |
| + resetProgram(ProgramInterpreter) : Program?  + init?(ProgramInterpreter)  + run()  + runNext()  + requestInput() : Any |

The static method resetProgram()on the Program class creates an instance of a Program object using the ProgramInterpreter object. This method will also set the singleton Program.shared to this new Program object as mentioned previously.

The resetProgram() method also returns the Program object as well as setting the shared Program. However, the return type uses the keyword @discardableResult which means that it does not need to be used. If @discardableResult was not used, the following code:

Program.resetProgram(…)

would cause the Xcode compiler to show a warning saying the return value is not used. However, adding this keyword will silence this compiler warning. I have added the return type here so that unit testing could be done on the Program object. I have not implemented any unit testing but have left this here for stability improvements of the app in the future.

The Program initializer takes a parameter of type ProgramInterpreter. This will check the code to see if it is valid or not. Remember, the program interpreter object contains two properties: canRun (a Boolean value to determine whether the syntax is correct or not) and code (an array of Strings of each line of code). If the canRun property of the interpreter is false, or the number of lines of code is not greater than 0, the Program initializer will not return an object.

If the code from the interpreter is valid however, it will run through each line of code in the array and parse it to create an Instruction object. Each instruction item will be added to the instructions array property of the Program object.

The Program object contains a few key properties:

* instructions, an array of Instruction objects.  
  This stores objects that represent each instruction in the inputted pseudo code.
* outputs, an array of Any objects.  
  This stores all the objects that are outputted from the pseudo code using the OUTPUT command.
* variables, a dictionary of key type String and value type Any.  
  This stores the virtual memory of all the variables in the program at a point during the program.

The Program object also has a method called run(). This will perform a foreach loop on the instructions array and call another function called run()on the Instruction class and all its subclasses. Alternatively, there is a method called runNext() which will execute one line at a time.

When the program has finished execution using either method, calling either method again will not do anything, since the property completedExecution flags that the program has finished execution.

The creation of the Instruction objects is the core part where the majority of the functionality of commands are implemented. Once validation has been passed from the interpreter, inside the Program initializer, it will loop through each string command which was passed in and create an Instruction object from it.

# Instruction Class

Here is the class definition for the Instruction class again:

|  |
| --- |
| **Instruction** |
| + line: String |
| + init(String)  + createInstruction(String) : Instruction?  + run() |

The Instruction class has a static method called createInstruction() which takes a string parameter of the command and returns an optional instance of an Instruction object.

The CommandParser class is used to validate lines of code and provide information about that line of code and the data it inputs, the data it represents and the actions it wants to perform. Here is a class definition for the CommandParser class:

|  |
| --- |
| **CommandParser** |
| + type: CommandType?  + match: CommandMatch?  - command: String  - variables: [String : Any] |
| + init(String, [String : Any]?)  + createValue(String) -> Any?  - textMatches(String) : Bool  - substituteVariables(String) : String  - inputMatch() : CommandMatch  - outputMatch() : CommandMatch?  **// More match results have been omitted for now. The full class definition is shown later in the CommandParser section of this document.** |

Inside the createInstruction() method, an instance of CommandParser is created using the string command as a parameter for initialization. The type property is retrieved from the CommandParser object which is an optional enum of type CommandType. If this type property is not nil, it is evaluated using a switch statement for each case of the CommandType enum.

The type property of CommandParser is computed by comparing the line of code passed into the initializer against a set of predefined regex patterns which describe each of the types of command that are available for this pseudo code syntax. If one of these matches, it will return that case, otherwise it will return nil.

The Swift definition for the enum CommandType is:

**enum CommandType {**

**case** input, output, assignment, forLoop, endForLoopm, ifStatement, elseStatement, endIfStatement

**}**

In each of these cases in the switch statement, a specific subclass of the Instruction object is returned.

The base instruction class has an initializer which takes a parameter of String type which is the inputted line of pseudo code and has a method run(). The Instruction class itself does not do anything. Initializing an Instruction object will not work and the run()method has no code inside it.

Instead, the Instruction class acts as a template for subclasses of itself. All subclasses of Instruction will inherit the initializer method and the run() method of Instruction. With all Instruction subclasses having the same base class, it means all subclasses can be expressed as a base Instruction object and later casted to the actual subclass that it is.

The createInstruction() method mentioned earlier will determine the type of command the string is, and then return whichever subclass of Instruction represents that type of instruction.

The code below shows the createInstruction() method:

|  |
| --- |
| Swift |
| static func createInstruction(line: String) -> Instruction? {  guard let match = CommandParser(command: line).type else {  return nil  }  switch match {  case .input: return InputInstruction(line: line)  case .output: return OutputInstruction(line: line)  case .assignment: return AssignmentInstruction(line: line)  case .forLoop: return ForInstruction(line: line)  **// and so on for all cases of CommandType**  } |
| Pseudocode |
| FUNCTION createInstruction(String line) RETURNS Instruction  match <- CommandParser(line).type  IF match = null THEN  RETURN null  END IF  IF match IS OF CASE input THEN  RETURN InputInstruction(line)  ELSE IF match IS OF CASE output THEN  RETURN OutputInstruction(line)  ELSE IF match IS OF CASE assignment THEN  RETURN AssignmentInstruction(line)  ELSE IF match IS OF CASE forLoop THEN  RETURN ForInstruction(line)  **// and so on for all cases of CommandType**  END FUNCTION |

Since each type of instruction is a different type of subclass of Instruction, each subclass has a different implementation for its run()method. Inside the run()method for each, the check of the type of command it is through CommandParser is performed again. However, the match property is accessed this time instead instead of the type.

The property match is an optional enum of type CommandMatch. CommandMatch is an enum with all the same cases as the enum CommandType except CommandMatch contains associated values of values necessary to each command. These are specific to each case of CommandMatch. The workings of how match is determined is explained in the CommandParser section.

Returning to the run() method, if the CommandParser‘s property match is not nil, it will retrieve the associated values for the case and perform custom code for each subclass. For example, here is the code for the run() method of the input instruction command:

|  |
| --- |
| Swift |
| class InputInstruction: Instruction {    override func run() {  guard let match = CommandParser(command: line).match else {  return  }  if let case .input(variableName) = match {  **// Run custom code using the associated value which is stored in the variable ‘variableName’.**  **// I will call this code the “instruction implementation code”.**  }  }  } |
| Pseudocode |
| CLASS InputInstruction INHERITS Instruction  OVERRIDE FUNCTION Run()  match <- CommandParser(line).match  IF match = null THEN  RETURN  END IF  IF match IS OF CASE input THEN  variableName <- match.associatedValue  **// Run instruction implementation code**  END IF  END FUNCTION  END CLASS |

This example shows:

1. The initialization of the CommandParser.
2. The retrieval of the match property.
3. The guard to check that the match property is not nil.
4. The check of the case of the match and to retrieve any associated values.
5. Running the custom code which is normally in place of the comment using the associated values of.

## InputInstruction Class

For each of the Instruction subclasses from now on, I will describe each of the associated values which are returned from the match property of the CommandParser and show the code which is used to perform that statement (“Instruction implementation code”).

Most Instruction subclasses will only override the run() method from the Instruction class. Therefore, I will only provide class definitions if more properties or methods are defined beyond this.

**Associated Values:**

|  |  |
| --- | --- |
| String variableName | The target variable name of type String which the inputted value will be stored into. If the command were:  INPUT MyVariable  Then variableName would be ‘MyVariable’. |

**Instruction implementation code:**

|  |
| --- |
| Swift |
| if case let .input(variableName) = match {  let input = Program.shared!.requestInput()  Program.shared!.variables[variableName] = input  } |
| Pseudocode |
| IF match IS OF CASE input THEN  variableName <- match.associatedValue  input <- Program.shared.requestInput()  Program.shared.variables[variableName] <- input  END IF |

This code will request an input from the current program and then set that value as the target variable name in the program’s dictionary of variables.

For a ProgramInterpreter to be initialized, it requires all inputs to be passed in beforehand. These inputs were passed onto the Program class when created. The requestInput() method will retrieve the next input in the array and keep track of the index with a private Int variable in the Program class called inputIndex. If an input is requested and it has already retrieved all inputs, it will return -1.

## OutputInstruction Class

**Associated Values:**

|  |  |
| --- | --- |
| Any value | The value which is to be outputted to the console. In the following code:  MyVariable <- 12345  OUTPUT MyVariable  Then value would be 12345. |

**Instruction implementation code:**

|  |
| --- |
| Swift |
| if case let .output(value) = match {  Program.shared!.outputs.append(value)  } |
| Pseudocode |
| IF match IS OF CASE output THEN  value <- match.associatedValue  Program.shared.outputs.Add(value)  END IF |

The Program class’s property output contains an array of type [Any]. This will contain all the values that are outputted to the console. Here, the new value is simply added to the end of the array.

## AssignmentInstruction Class

The assignment instruction is one of the most important instructions types. Not only is it used in simple assignment commands but it is also used when embedded in other commands such as a for loop to track the index or a while loop for the same reason.

**Associated Values:**

|  |  |
| --- | --- |
| String targetVarName | The target variable name of the assignment instruction. For example, in this code:  Variable <- 2  Then targetVariableName would be “Variable” |
| Any value | The value which will be placed into the the variable. For example, in this code:  Variable <- 2  Then value would be 2. |

**Instruction implementation code:**

|  |
| --- |
| Swift |
| if case .assignment(let targetVarName, let value) = match {  Program.shared!.variables[targetVarName] = value  } |
| Pseudocode |
| IF match IS OF CASE assignment THEN  targetVarName <- match.associatedValue1  value <- match.associatedValue2  Program.shared.variables[targetVarName] <- value  END IF |

This code will simply set the value of the target variable name in the shared Program object’s variables dictionary.

## ForInstruction Class

|  |
| --- |
| **ForInstruction** |
| + instructions: [Instruction] |
| + override run() |

The ForInstruction subclass contains extra properties which none of the other subclasses so far have had. It contains a property called instructions of type [Instruction] which is an array of all the instructions contained inside the scope of the for instruction. This property can be thought of as a pointer to a subroutine which is called for each iteration of the loop.

**Associated Values:**

|  |  |
| --- | --- |
| String indexVarName | The target variable name of the index inside the for loop instruction. For example, in this code:  FOR index <- 0 TO 100  Then indexVarName would be “index” |
| int startIndex | The value of the first index to start the for loop from. For example, in this code:  FOR index <- 0 TO 100  Then startIndex would be 0. |
| int endIndex | The value of the last index the for loop would reach. For example, in this code:  FOR index <- 0 TO 100  Then endIndex would be 100. |

**Instruction implementation code:**

|  |
| --- |
| Swift |
| if case .forLoop(let indexVarName, let startIndex, let endIndex) = match {  for index in startIndex...endIndex {  Program.shared!variables[indexVarName] = index  for instruction in instructions {  instruction.run()  }  }  Program.shared?.variables[indexVarName] = nil  } |
| Pseudocode |
| IF match IS OF CASE forLoop THEN  FOR index <- startIndex TO endIndex  Program.shared.variables[indexVarName] <- index  FOREACH instruction IN instructions  instruction.run()  NEXT  NEXT  Program.shared.variables[indexVarName] <- null  END IF |

This will perform a loop in Swift beginning at the startIndex and ending at the endIndex variable. Before the code block inside the for loop is run each time, it temporarily sets the variable with name stored in indexVarName and sets it to the current index. Then after the for loop has completed execution, it will remove the index from the shared Program’s variables so that it can only be available to code within that scope.

## EndForInstruction Class

There is also a class called EndForInstruction, which is a subclass of Instruction. This class does not contain any custom implementation but is used as a marker so that the Program can identify the start and end of a for loop when “interpreting” the code in the creation of the program.

|  |
| --- |
| Swift |
| class EndForInstruction: Instruction {} |
| Pseudocode |
| CLASS EndForInstruction INHERITS Instruction  END CLASS |

Any other classes further on that are simply used as markers are also defined exactly like this.

## IfInstruction Class

|  |
| --- |
| **IfInstruction** |
| + trueBlock: [Instruction]  + falseBlock: [Instruction] |
| + override run() |

The IfInstruction class contains two extra variables, a trueBlock and a falseBlock. Both of these are of type Instruction array and are called whether the Boolean expression in the code evaluates to true or not.

**Associated Values:**

|  |  |
| --- | --- |
| Bool boolVal | The result of the Boolean expression inside the if statement. For example:  IF 6 < 10 THEN  END IF  IF 1 <> 10 THEN  END IF  For both of these examples, boolVal would result in true. |

**Instruction implementation code:**

|  |
| --- |
| Swift |
| if case let .ifStatement(let boolVal) = match {  let instructions = boolVal ? trueBlock : falseBlock  for instruction in instructions {  instruction.run()  }  } |
| Pseudocode |
| IF match IS OF CASE ifStatement THEN  boolVal <- match.associatedValue  IF boolVal THEN  instructions <- trueBlock  ELSE  instructions <- falseBlock  END IF  FOREACH instruction in instructions  instruction.run()  NEXT  END IF |

This will use the boolVal associated value and set the local instructions variable to the property trueBlock or falseBlock accordingly. It will then loop through the chosen array and run that block of code.

## ElseInstruction & EndIfInstruction Classes

There are also two extra classes called ElseInstruction and EndIfInstruction which do not have any code. They are also used as markers, the same as the EndForInstruction class was used.

## RepeatInstruction Class

The RepeatInstruction class acts as the marker here where RepeatUntilInstruction contains the functionality, since the code should be run first, then evaluated whether it should run again or not.

## RepeatUntilInstruction Class

|  |
| --- |
| **RepeatUntilInstruction** |
| + instructions: [Instruction] |
| + override run()  - performRepeatUntil(String)  - evaluate(String) -> Bool |

The RepeatUntilInstruction class contains one extra variable, instructions which is of type [Instruction] and are called whether the Boolean expression in the code evaluates to true or not.

**Associated Values:**

|  |  |
| --- | --- |
| Bool boolVal | The result of the Boolean expression inside the if statement. For example:  REPEAT  // code  UNTIL 9 < 10  In this example, boolVal would result in true. |

**Instruction implementation code:**

|  |
| --- |
| Swift |
| if case let .repeatUntil(boolExpr) = match {  performRepeatUntil(boolExpr)  }  fileprivate func performRepeatUntil(usingBoolExpr boolExpr: String) {  for instruction in instructions {  instruction.run()  }  if !evaluate(boolExpr: boolExpr) {  performRepeatUntil(usingBoolExpr: boolExpr)  }  }  fileprivate func evaluate(boolExpr: String) -> Bool {  let substituted = RegexSubstituter(command: boolExpr, variables: Program.shared?.variables).substitute()  return BooleanExpressionParser(expr: substituted)?.calculate() ?? false  } |
| Pseudocode |
| IF match IS OF CASE repeatUntil THEN  boolExpr <- match.associatedValue  performRepeatUntil(boolExpr)  END IF  FUNCTION performRepeatUntil(String boolExpr)  FOREACH instruction IN instructions  instruction.run()  NEXT  IF !evaluate(boolExpr) THEN  performRepeatUntil(boolExpr)  END IF  END FUNCTION  FUNCTION evaluate(String boolExpr) RETURNS Boolean  service <- RegexSubstituter(boolExpr, Program.shared.variables)  substituted <- service.substitute()  value <- BooleanExpressionParser(substituted).calculate()  RETURN value  END FUNCTION |

This will perform a repeat until loop by running each of the instructions first. If the Boolean value evaluates to true after this, then the block is recursively run again.

## WhileInstruction

|  |
| --- |
| **WhileInstruction** |
| + instructions: [Instruction] |
| + override run()  - performWhile(String)  - evaluate(String) -> Bool |

The WhileInstruction class contains one extra variable, instructions which is of type [Instruction] and are called whether the Boolean expression in the code evaluates to true or not.

**Associated Values:**

|  |  |
| --- | --- |
| Bool boolVal | The result of the Boolean expression inside the if statement. For example:  WHILE true  // code  END WHILE  In this example, boolVal would always result in true. |

**Instruction implementation code:**

|  |
| --- |
| Swift |
| if case let .whileStart(boolExpr) = match {  performWhile(usingBoolExpr: boolExpr)  }    fileprivate func performWhile(usingBoolExpr boolExpr: String) {  if evaluate(boolExpr: boolExpr) {  for instruction in instructions {  instruction.run()  }  performWhile(usingBoolExpr: boolExpr)  }  }    fileprivate func evaluate(boolExpr: String) -> Bool {  let substituted = RegexSubstituter(command: boolExpr, variables: Program.shared?.variables).substitute()  return BooleanExpressionParser(expr: substituted)?.calculate() ?? false  } |
| Pseudocode |
| IF match IS OF CASE whileStart THEN  boolExpr <- match.associatedValue  performWhile(boolExpr)  END IF  FUNCTION performWhile(String boolExpr)  IF evaluate(boolExpr) THEN  FOREACH instruction IN instructions  instruction.run()  END FOREACH  performWhile(boolExpr)  END IF  END FUNCTION  FUNCTION evaluate(String boolExpr) RETURNS Boolean  service <- RegexSubstituter(boolExpr, Program.shared.variables)  substituted <- service.substitute()  value <- BooleanExpressionParser(substituted).calculate()  RETURN value  END FUNCTION |

This will evaluate the Boolean value before running, unlike the Repeat Until instruction. If the value is true, then it will run the block and then recursively call the perform method again. It will re-evaluate the expression and continue if appropriate.

# CommandParser Class

The CommandParser class will parse a string statement of code during “runtime” of the program and pass back information about variables and type of Instruction to run about that command. It works using the RegexExtractor class to compare patterns and extract values.

|  |
| --- |
| **CommandParser** |
| + type: CommandType?  + match: CommandMatch?  - command: String  - variables: [String : Any] |
| + init(String, [String : Any]?)  + createValue(String) : Any?  - textMatches(String) : Bool  - substituteVariables(String) : String  - inputMatch() : CommandMatch  - outputMatch() : CommandMatch?  - assignmentMatch() : CommandMatch?  - forLoopMatch() : CommandMatch?  - ifStatementMatch() : CommandMatch?  - repeatUntilMatch() : CommandMatch?  - whileMatch() : CommandMatch? |

The two most important properties of the command parser class are the computed properties of type and match.

The type property returns a enum of type CommandType. This will tell the program what type of command this string is describing and can therefore be matched to a subclass of Instruction in the program, without the need for final variable values. The code for deciding the type of the string is a series of if statements that compares the string against predefined Regex patterns. If none of the if statements pass true, then the property will return nil. An example of one of these if statements is below for the input instruction:

if textMatches(regex: CommandRegexPattern.input) {

return .input

}

The textMatches(regex:) method uses the RegexExtractor class to compare the class’s command property against the parameter of the regex pattern passed in. CommandRegexPattern is a struct which contains a set of static predefined values for each of the regex patterns for each of the types of statement supported by the application.

The match property is the next step after variable values have been decided during the simulated “runtime” of the program and can relate the string line of code to a type and also give all the variables so that the instruction can be executed. Below are the definitions for CommandType and CommandMatch. The associated values for each case of CommandMatch were explained in the section for each subclass of Instruction.

|  |  |
| --- | --- |
| **CommandType** | **CommandMatch** |
| enum CommandType {  case input  case output  case assignment  case forLoop  case endForLoop  case ifStatement  case elseStatement  case endIfStatement  case repeatStart  case repeatUntil  case whileStart,  case endWhile  } | enum CommandMatch {  case input(String)  case output(Any)  case assignment(String, Any)  case forLoop(String, Int, Int)  case endForLoop  case ifStatement(Bool)  case elseStatement  case endIfStatement  case repeatStart  case repeatUntil(String)  case whileStart(String)  case endWhile  } |

The match property is a computed variable. It first accesses the type property of itself and evaluates it using a switch statement to return the correct case along with calculating the associated values to be passed with the enum.

If a CommandMatch case does not have any associated values (endForLoop, elseStatement, endIfStatement), then it will return the enum case straight away.

For all other statement types, it will call a specific private method which will extract the values for each case and return the enum type along with their associated values.

Each of the “match methods” are defined in the class definition and are specific to each case but all perform similar algorithms for the manipulation of the string to extract the associated values. Here is an example of what happens to obtain the associated values for the OUTPUT command:

|  |
| --- |
| Swift |
| fileprivate func outputMatch() -> CommandMatch? {  let expr = substituteVariables(intoExpr: command)  let results = RegexExtractor.matches(forRegex: CommandRegexPattern.output, in: expr)  let valueExpr = results[1]    if let value = CommandParser.createValue(fromExpression: valueExpr) {  return .output(value)  }    return nil  } |
| Pseudocode |
| FUNCTION outputMatch() RETURNS CommandMatch?  expr <- substituteVariables(command)  results <- RegexExtractor.matches(CommandRegexPattern.output, expr)  valueExpr <- results[0]  value <- CommandParser.createValue(valueExpr)  if value != null THEN  return CommandMatch.output(value)  END IF  RETURN null  END FUNCTION |

The substituteVariables(intoExpr:) method will substitute all current program variables into the statement using the RegexSubstituter class. The RegexSubstituter class was previously used in the ProgramInterpreter class to validate the code. I discuss how this class works later on in this document.

The RegexExtractor.matches(forRegex:) will return an array with all captured values from the regex pattern, starting with the first item being the whole string and then the rest being each of the captured values. The value of an output command will therefore be in results[1] (the second item in the array).

The CommandParser.createValue(fromExpression:) will then return a value of type Any returning the type of value that is in valueExpr, if it is valid. It will then return the CommandMatch type with the value if it was valid, or return nil if it was not valid.

The flow of how these manipulate the original string is shown in the following example.

|  |
| --- |
| **Example – CommandParser Output Match** |
| For this example, there are already existing values in the program of:  valueOne <- 20  valueTwo <- 40  The original line of code is:  OUTPUT valueOne + valueTwo  **substituteVariables(intoExpr:)** will turn this into:  OUTPUT 20 + 40  **RegexExtractor.matches(forRegex:)** will extract the capture values and return the array:  [“OUTPUT 20 + 40”, “20 + 40”]  **CommandParser.createValue(fromExpression:)** will take the “20 + 40” string and return the value 60 as an Int.  We then return the CommandMatch case output with associated value 60. |

The createValue(fromExpression:) is an important feature of the CommandParser class. The inputted parameter is a substring of whatever statement is being analysed (for example the “20 + 40” string in the above example).

It will compare the substring against the regex patterns for an integer expression, Boolean value, Boolean expression or a string expression else it will return nil:

* In the case of an integer expression, it will use the ShuntingYardParser class to calculate the integer result of the string expression.
* In the case of a Boolean value, it will compare the string expression against the strings “TRUE”, “true”, “FALSE” or “false” and return the Boolean value accordingly.
* In the case of a Boolean expression, it will use the BooleanExpressionParser class to calculate the Boolean result of the string expression.
* In the case of a string expression, it will use the StringExpressionParser class to calculate the string result of the string expression.

These classes will transform complex expressions into one value. For example it will turn a complex math expression into a single integer value (e.g. “20\*((30+5)/(10-3))” into 100). The same applies for Boolean expressions or string expressions.

As mentioned, all the other match methods that are in the class definition for CommandParser use similar methods to obtain the associated values. For example the ifStatementMatch() method uses the BooleanExpressionParser to determine the value of the Boolean expression inside the statement. The forLoopMatch() method uses the ShuntingYardParser class to determine the values of the startIndex and endIndex of the loop.

# CodeErrorHandler Class

|  |
| --- |
| **CodeErrorHandler** |
| + provideErrors(String) : [Error]  - errorCodes(String) : [Int]  - error(Int) : Error |

This class is one used by ProgramInterpreter to provide detailed errors for a line of code.

The static method provideErrors(String) will return an array of type [Error]. Where Error is a struct defined as:

|  |
| --- |
| **Error** |
| + code: Int  + message: String  + link: URL? |
| + init(Int, String, String?) |

An error object contains an error code, an error message and an optional link which the user should be directed to where he/she can find out more about the syntax error that they have.

The provideErrors method will call the private errorCodes method which performs a series of checks against the string line of code that is provided. Due to time restraints, I only provided to specific types of error which are outlined below:

## Incorrect Case Syntax Error

This error occurs when a user uses the wrong case for a keyword. For example, Output instead of OUTPUT. It does this by capitalizing the whole line and comparing whether the CommandParser will accept this capitalized line of code instead.

|  |
| --- |
| Swift |
| let parser = CommandParser(command: line)  var codes: [Int] = []    if parser.type == nil {  let uppercase = line.uppercased()  let newType = CommandParser(command: uppercase).type  if newType != nil {  codes.append(1)  }  } |
| Pseudocode |
| parser <- CommandParser(line)  codes <- []  IF parser.type = null THEN  uppercase <- line.toUppercase()  newType = CommandParser(uppercase).type  IF newType <> null THEN  codes.Add(1)  END IF  END IF |

## Unterminated String Literal Syntax Error

This error occurs when a user starts a string literal by using an apostrophe but forgetting to close the string literal by using an ending apostrophe. This was one of the most common errors for GCSE students according to my client.

This code will count the number of apostrophes and check whether it is even or odd. If it is odd, there is an unterminated string literal somewhere.

The parser and codes variable from the previous code snippet are in the same scope as the following code snippet.

|  |
| --- |
| Swift |
| var numberOfApostrophes = 0  line.forEach({ numberOfApostrophes += $0 == "'" ? 1 : 0 })  if numberOfApostrophes % 2 > 0 {  codes.append(2)  } |
| Pseudocode |
| numberOfApostrophes <- 0  FOREACH character IN line  IF character = “’” THEN  numberOfApostrophes <- numberOfApostrophes + 1  END IF  END FOREACH  IF (numberOfApostrophes MOD 2) > 0 THEN  codes.Add(2)  END IF |

## Other Syntax Errors

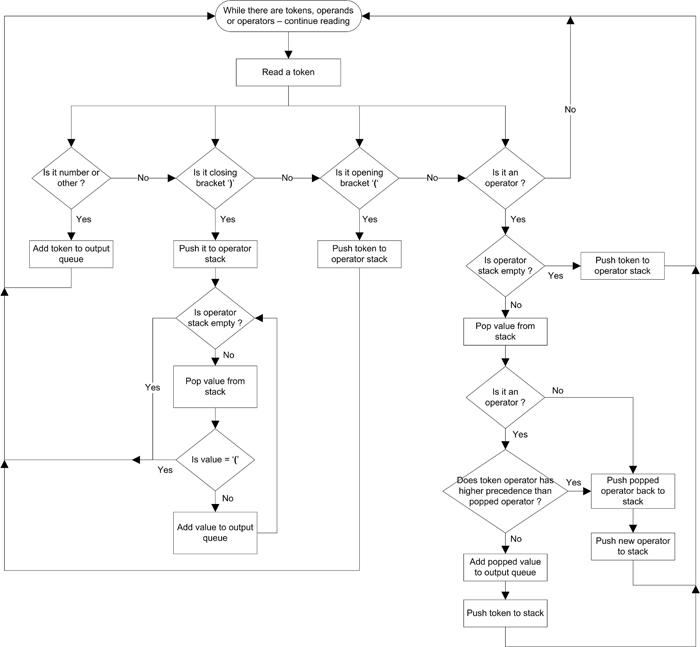
If the codes array is empty after all these checks, then the code 0 is entered. The error(Int) will provide an Error object which contains a message and an optional link for each error code.

Due to time constraints, I was not able to provide any other error checks, but I have left the design modular so more additional checks can be added at any point, with custom messages and links for each type of error.

# Shunting Yard Parser

This class will transform mathematical expressions that are described in infix notation into a single value and is stack based rather than other algorithms for math expression parsing. It uses the Shunting Yard algorithm that was invented by Edsger Dijkstra and named the "shunting yard" algorithm because its operation resembles that of a railroad shunting yard.

Below is a flowchart describing the algorithm of the shunting yard algorithm.



Here is the pseudo code which outlines the algorithm for Shunting Yard:

|  |
| --- |
| **while** there are tokens to be read:  read a token.  **if** the token is a number, **then** push it to the output queue.  **if** the token is an operator, **then**:  **while** ((there is an operator at the top of the operator stack with greater precedence)  **or** (the operator at the top of the operator stack has equal precedence and the operator is left associative))  **and** (the operator at the top of the stack is not a left bracket):  **pop** operators from the operator stack, onto the output queue.  **push** the read operator onto the operator stack.  **if** the token is a left bracket (i.e. "("), **then**:  **push** it onto the operator stack.  **if** the token is a right bracket (i.e. ")"), then:  **while** the operator at the top of the operator stack is not a left bracket:  **pop** operators from the operator stack onto the output queue.  **pop** the left bracket from the stack.  */\* if the stack runs out without finding a left bracket, then there are mismatched parentheses. \*/*  **if** there are no more tokens to read:  **while** there are still operator tokens on the stack:  */\* if the operator token on the top of the stack is a bracket, then there are mismatched parentheses. \*/*  **pop** the operator onto the output queue.  **exit.** |

*(Source: wikipedia.com)*

Here is the class definition for the ShuntingYardParser class:

|  |
| --- |
| **ShuntingYardParser** |
| + infix: String  + postfix: String  - input: Queue<String>  - tokens: Queue<String>  - operators: Queue<Operator>  - output: Queue<String>  - validOperators: [String] |
| + init(String)  + calculate() : Int  - parse(String)  - convertToPostFix() |

The Shunting Yard parser also uses the following classes:

## Operator Class

|  |
| --- |
| **Operator** |
| - op: String |
| + init(String)  + precedence() : Int |

Operator is a class which calculates the precedence/priority for BIDMAS order of an inputted operator out of the selection: ( ) + - / \*.

* The ( operator has precedence 1
* + and – has precedence 2
* \* and / has precedence
* All other strings have precedence -1.

The higher the number, the higher the priority of that operator.

## Queue Class

Queue conforms to the Sequence protocol. This means that it adopts the characteristics of a Sequence which is an iOS defined term for a sequence.

The <T> will define the class that the Queue uses. For example, if we define an object as a Queue<String>, it will mean that the queue uses String for each of the elements within its sequence. The T can be referenced as a type anywhere within its class definition.

|  |
| --- |
| **Queue<T> : Sequence** |
| + N: Int  + isEmpty: Bool  - first: Node?  - last: Node? |
| + enqueue(T)  + dequeue() : T?  *Below is a method that is required by the Sequence protocol where QueueIterator is a defined class which conforms to the IteratorProtocol protocol and is also required by the Sequence protocol.*  + makeIterator() -> QueueIterator |

|  |
| --- |
| **Node** |
| + item: T  + next: Node? |
| + init(T) |

Queue is a custom implementation of a queue with first in, first out order.

## Stack

|  |
| --- |
| **Stack<T> : Sequence** |
| + isEmpty: Bool  + size: Int  - N: Int  - last: Node? |
| + peek() : T?  + push(T)  + pop() : T?  *Below is a method that is required by the Sequence protocol where StackIterator is a defined class which conforms to the IteratorProtocol protocol and is also required by the Sequence protocol.*  + makeIterator() -> StackIterator |

Stack is a custom implementation of a stack with first in, last out order.

## Polish

|  |
| --- |
| **Polish** |
| + stringValue: String |
| + init(Queue<String>)  + calculate() : Int  - complainNaNError() |

Polish is a class which will convert an expression of polish notation from the Shunting Yard expression into the final value.

It puts every item (numbers and operators) of the Polish expression into a queue. It will then perform a for loop through each item in the stack.

* If the item is an integer value, it will push it to a stack which we’ll call A.
* If the item is an operator, it will pop the stack A and put the value into a variable called left and then pop stack A again and put the value into a variable called right.

The calculation based on the operator (+, -, / or \*) will then be performed on the two numbers (left and right). The resulting value will then be pushed onto stack A.

The final value of the expression will then be returned by popping the last value from stack A if it exists, else returning 0 if there was an error.

The method complainNaNError() will be called the user tries to divide a value by 0. Normally, if I tried to perform this calculation, my application would crash. To prevent this, I return 0 from this calculation and print to the console this message:

*“NaN error! Returning 0 to prevent crash.”*

# Boolean Expression Parser

|  |
| --- |
| **BooleanExpressionParser** |
| - type: ParserType  - lhs: Any  - rhs: Any  - op: String |
| - init([String], ParserType)  + init?(String)  + calculate() : Bool?  - calculateIntegerValue() : Bool?  - calculateStringValue() : Bool?  - calculateBooleanValue : Bool? |

ParserType is an enum with cases integer, string and boolean. There are no associated values for any of these cases.

The BooleanExpressionParser class works mainly using regex patterns and the expression parsers of other data types, since it is comparing values. The regex pattern always takes the form: *LHS OPERATOR RHS*. The LHS and RHS must be either String, Integer or Boolean and must be the same data type. The choice of operators are equals to (=), less than (<), greater than (>), less than or equal to (<=), greater than or equal to (>=) or not equal to (<>). However, the available choice of operators is different for each data type:

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Possible Operators** | **Regex Pattern** |
| **Integer** | ==, <, >, <=, >=, <> | ((?:[0123456789()+-/\*\\s])+) ?(=|<|>|<=|>=|<>) ?((?:[0123456789()+-/\*\\s])+) |
| **String** | ==, <> | (^(?:\'.+\' \\+ )\*(?:\'.+\')$) ?(=|<>) ?(^(?:\'.+\' \\+ )\*(?:\'.+\')$) |
| **Boolean** | ==, <> | (TRUE|FALSE) ?(=|<>) ?(TRUE|FALSE) |

The BooleanExpressionParser class takes an initializer with parameter String which is the Boolean expression to be analysed. It will then compare the string against each of the regex patterns for integer, string and boolean and determine which type it is and save it as the type property. It will assign the left hand (LHS) and right hand (RHS) properties extracted from the 1st and 3rd captured values of the regex pattern and assign the operator as the 2nd extracted capture value from the regex pattern.

When calculate() is called, it will call one of three private methods:

1. calculateIntegerValue()
2. calculateStringValue()
3. calculateBooleanValue()

based on the type property of the class. In each method, a switch on the operator which is of type String is performed and the calculation to determine the final Boolean value is performed and returned.

# RegexExtractor Class

|  |
| --- |
| **RegexExtractor** |
| + textMatches(String, String) : Bool  + matches(String, String) : [String] |

The RegexExtractor class has been has been used on numerous occasions within the application so far.

There are two main methods which are detailed below:

## Text Matches function

|  |
| --- |
| Swift |
| static func textMatches(\_ text: String, forRegex regex: String) -> Bool |
| Pseudocode |
| STATIC FUNCTION textMatches(String text, String regex) RETURNS Bool |

This method will check the parameter text and compare it against the parameter regex and return a Boolean to see whether it is a valid expression for the regex pattern. It uses a class called NSPredicate which is part of the standard iOS framework provided with the system. NSPredicate is defined in Apple’s documentation as “a definition of logical conditions used to constrain a search either for a fetch or for in-memory filtering.”

NSPredicate uses its own syntax to compare values but it can compare regex too. By passing the string “SELF MATCHES X” (where X is the regex pattern) into the initializer for an NSPredicate object. Then, using the method evaluate(with:) which takes a String parameter (which is the text to be evaluated). This method returns a Boolean which indicates whether the match was successful or not.

## Matches function

|  |
| --- |
| Swift |
| static func matches(forRegex regex: String, in text: String) -> [String] |
| Pseudocode |
| STATIC FUNCTION matches(String regex, String text) RETURNS [String] |

This method will take in the text as a parameter text to be analysed again and the regex pattern to compare against again as a parameter regex but will this time return an array of all capture values that were included in the regex pattern. As mentioned previously, the first value of the array is the whole string and each of the following strings are each of the regex patterns (due to the way the NSRegularExpression class in the iOS framework works). For example:

If the following string “20 == 40” with the regex pattern “(.) == (.)”, then the returned array would be: [“20 == 40”, “20”, “40”].

This method uses the NSRegularExpression class which is part of the standard iOS framework provided with the system. A NSRegularExpression object is initialized with the regex pattern as a parameter. NSRegularExpression has a method which will return a NSTextCheckingResult which can be used to determine the ranges of each of the matches in the string which can then be used to extract each of the captured values from the inputted string. The method for retrieving ranges returns the whole string as the first range and is why my method in RegexExtractor behaves similarly.

There is a struct named CommandRegexPattern (has been discussed previously) which includes all the regex patterns which are used throughout the program as static String members. Below is a list of all of the regex patterns which have been used at some point throughout the program:

|  |  |
| --- | --- |
| **Type** | **Regex Pattern** |
| **Integer Expression** | (?:[0123456789()+-/\*\\s])+ |
| **Boolean Value** | TRUE|FALSE |
| **String Expression** | ^(?:'.+' \\+ )\*(?:'.+')$ |
| **Variable Name** | [a-zA-Z\_$][a-zA-Z\_$0-9]\* |
| **Input** | INPUT (\(variable)) |
| **Output** | OUTPUT (.+) |
| **Assignment** | (\(variable)) <- (.+) |
| **For Loop** | FOR (\(variable)) <- (.+) TO (.+) |
| **End For Loop** | NEXT |
| **If Statement** | IF (.+) THEN |
| **Else Statement** | ELSE |
| **End If Statement** | END IF |

I constructed these all myself, using the website [www.regexr.com](http://www.regexr.com) as an aid.

Some of these regex patterns include substitutions for other regex patterns using the Swift string literal syntax for combining strings.

So for example, in the input regex pattern, instead of:

“INPUT (\(variable))"

the actual resulting string would be:

“INPUT ([a-zA-Z\_$][a-zA-Z\_$0-9]\*)”

# RegexSubstituter Class

|  |
| --- |
| RegexSubstituter |
| - command: String  - varNames: [String]?  - variables: [String : Any]? |
| + init(String, [String]?, [String : Any]?)  + substitute() : String  - substituteExpression(String) : String  - substituteOutputExpression() : String  - substituteAssignmentExpression() : String  - substituteForLoopExpression() : String  - substituteIfStatement() : String  - substituteValues([String : Any], String) : String  - substituteVarNames([String], String, String) : String  - stringDescribing(Any) : String  - replaceOccurrences(String, String, String) : String |

The RegexSubstituter class is used to substitute values and variables into an existing code string and its use has already been demonstrated many times previously in this document. It is initialized with a code string, an optional array [String]? of variable names and an optional dictionary [String : Any]? of variables.

When the substitute() method on a RegexSubstituter object is called, it will substitute the values and return a modified string of the original string which was passed in the initializer.

The CommandParser class is first used to determine the type of command it is.

Only some types of command require value substitution, therefore a switch is called on the type. For commands of type output, assignment, forLoop and ifStatement, the string is modified and returned. Otherwise, the original code string is returned.

In each case of a modified command, if the variableNames array passed in at initialization is not empty, then each value is replaced with a 0. Else, if the variables dictionary passed in at initialization is not empty, then the values in that dictionary are replaced by the keys containing them. The reason for this was explained earlier in the Program Interpreter section when using the RegexSubstituter for the program validation or the program execution.

Below I will go through the above types of command that need to be modified and explain how it is modified:

## Output Substitution

An output command will look like:

OUTPUT X

The value of X is the only value in this expression and is the only one which will be substituted.

## Assignment Substitution

An assignment command will look like:

A <- B

There are two variables here, A and B. A cannot be substituted because it is the final destination for copying whatever value was in B. The value of A can already be have created or not so it is irrelevant for this expression. However, the value of B must be determined beforehand because otherwise there would be an unidentified identifier error in the program, since the value of a variable which has not been initialized is being used.

## For Loop Substitution

A for loop command will look like:

FOR A <- B TO C

The variables are A, B and C. Similar to the previous example, A is the new value for and therefore is not substituted. However, B and C must be determined so that the program knows what to loop from and to.

## If Substitution

An if command will look like:

IF A THEN

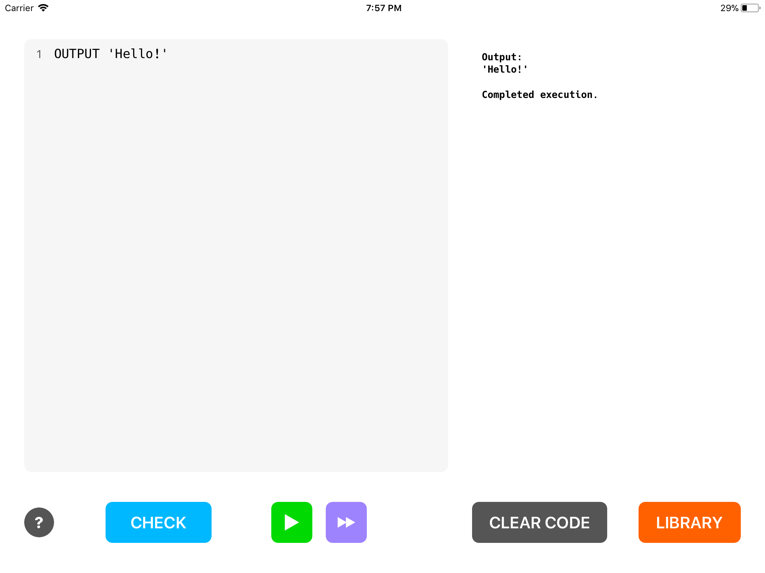
The only variable here is A and must be substituted. The expression in place of A may have been previously more complex but ultimately it will result as one definite Boolean value.

# iPad App Interface

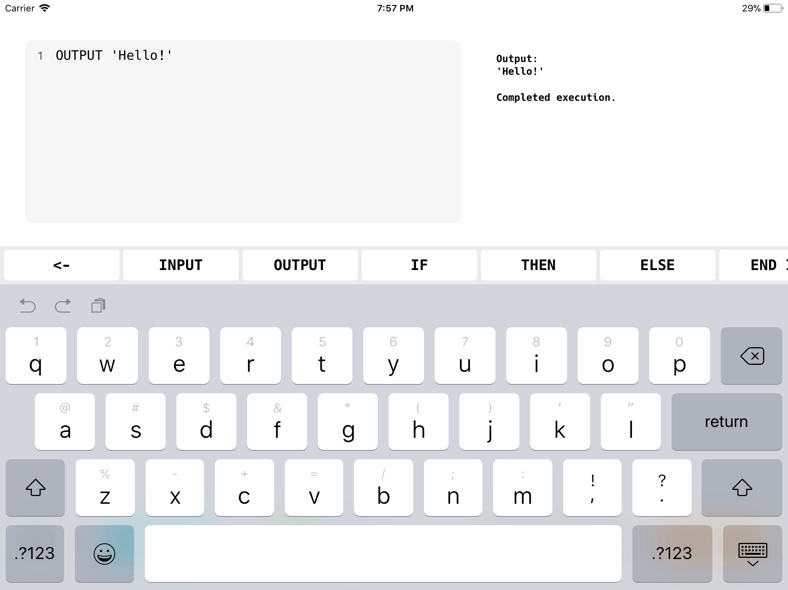
The iPad app consists of four main screens – the development screen, the input screen, the popup errors screen and the library screen, some of which were described in the analysis section.

## Development Screen

The development screen contains the editor for inputting the pseudo code to be run and the console where the output will be printed. It also contains simple controls to validate the program, skip lines of the program and to run the program.



The code for the development screen is contained in the ViewController class. This class inherits from UIViewController which is the base class for all view interfaces in iOS.

The editor is a series of views stacked on top of each other vertically and contained in and organised by a scroll interface. The whole view is organised by a table view which is represented by the class UITableView and each view is a row in the table view and is represented by the class UITableViewCell.

The editor UITableView is a private property of the view controller. This property makes use of the @IBOutlet tag before the property declaration which indicates that it connects to a specific user interface element within the interface builder file:

**@IBOutlet** fileprivate var inputTableView: UITableView!

An interface file in iOS is a custom Apple file type with extension .xib. Underneath, it is essentially a .xml file with properties for the appearance, layout and appearance of items. Xcode provides an interface builder interface where I can interactively change edit the interface, so the xml is rarely ever edited directly.

Alternatively, there is a .storyboard file. A xib file is used for laying out the user interface elements for just one type of view (one view controller, one table view cell, etc.). A new feature to Xcode was the introduction of storyboard files, where multiple view controllers and elements within those view controllers could be designed in one file and their transitions between screens be defined as segues. A storyboard file is also an xml file under the hood.

### UITableView

A UITableView is a subclass of UIScrollView. A UIScrollView class organises sub views inside a content area which is bigger than the frame of the view on the screen and allows the user to scroll around by touch and drag gestures. UITableView expands on this functionality of UIScrollView to present data in the form of a table separated into sections and table view cell rows within each section. A UITableView can scroll only vertically.

UITableView makes use of two protocols to determine the data which is to be displayed in the table view – UITableViewDataSource and UITableViewDelegate. UITableViewDataSource is used for determining the data which will be displayed on each table view cell and UITableViewDelegate is for other actions and options (such as the height of a table view cell row or what actions should be performed when a user taps on a table view cell row).

The code editor in this screen makes use of a UITableView where each table view cell is an inputted line of code in the program. To organise my code more clearly, I have created a new class called InputTableViewHandler which will be assigned as the delegate and data source of the table view in the ViewController class.

I first instantiate the InputTableViewHandler object as a property (not a local variable since automatic reference counting [ARC] will release it from memory) of the ViewController class and then I assign it as the delegate and the data source of the inputTableView property.

Inside the InputTableViewHandler class are 3 important methods which are required by the UITableViewDataSource protocol in order to implement a working UITableView. I will go through each of the methods one by one and explain exactly what each one does.

#### Sections

|  |
| --- |
| Swift |
| func numberOfSections(in tableView: UITableView) -> Int {  return 1  } |
| Pseudocode |
| FUNCTION numberOfSections(UITableView tableView) RETURNS Int  RETURN 1  END FUNCTION |

This will return how many separate sections in the table view there are. Normally sections are separated by a header and would mark a separate piece of data. However, in my case, all the table view cells are similar and should not be separated by any header, therefore we return 1.

#### Rows

|  |
| --- |
| Swift |
| func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {  return max(code.count, 1)  } |
| Pseudocode |
| FUNCTION tableView(UITableView tableView, Int section) RETURNS Int  RETURN Max(code.length, 1)  END FUNCTION |

This will return how many table view cells are in each section. This is called n number of times where n was the number of sections returned in the previous method. The index of the section is passed in as the parameter section. Since we know that we have only 1 section, we can ignore this parameter.

code is a private property of the InputTableViewHandler class which is an array of String for each inputted line of code. If the user has not inputted any code in yet, we want an empty line of code to be available to start typing. Therefore, we return the largest number between the length of the code array and the number 1.

#### Cell for Row

|  |
| --- |
| Swift |
| func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {  var line: String?  if code.count > indexPath.row {  line = code[indexPath.row]  }    let cell = tableView.dequeueReusableCell(withIdentifier: "InputCell", for: indexPath) as! InputLineTableViewCell  var errorCount = 0  if errors.count > indexPath.row {  errorCount = errors[indexPath.row].count  }  cell.configure(withLine: line, errors: errorCount, indexPath: indexPath)  cell.setScope(scopeIndexes?[indexPath.row])  cell.delegate = self  return cell  } |
| Pseudocode |
| FUNCTION tableView(UITableView tableView , IndexPath indexPath) RETURNS UITableViewCell  IF code.length > indexPath.row THEN  line <- code[indexPath.row]  END IF  cell <- tableView.dequeueReusableCell(“InputCell”, indexPath)  errorCount <- 0  IF errors.length > indexPath.row THEN  errorCount <- errors[indexPath.row].length  END IF  cell.configure(line, errorCount, indexPath)  cell.setScope(scopeIndexes[indexPath.row])  cell.delegate <- self  RETURN cell  END FUNCTION |

This is the most important method in the data source of a table view. It will return a UITableViewCell which is a subclass of UIView. UIView is the iOS class tha represents the most basic type of viewable interface item. In order to create custom views, you must return an object of type UITableViewCell, which in this case is the class InputLineTableViewCell (a subclass of UITableViewCell). The methods called on the cell variable will adjust the view and appearance of the table view cell.

This method will check the property code and see if there is currently some inputted code by the user for that line of code. If not, the local string variable line will be left as null.

Then we create the table view cell using a method on the table view object called dequeueReusableCell(withIdentifier:, for:) and force unwrap it as type InputLineTableViewCell.

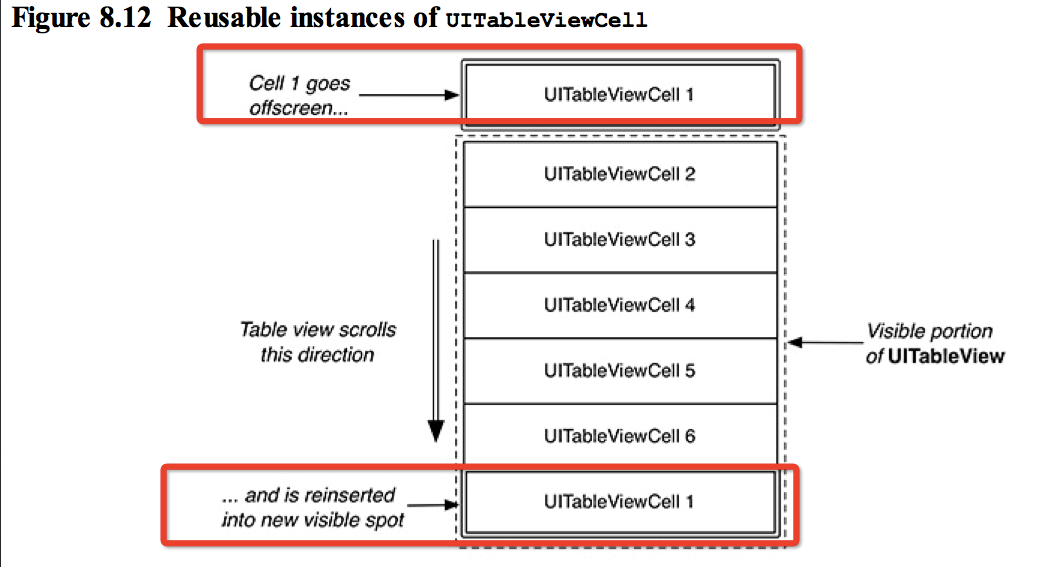
#### Reusable Cells

Table views in iOS work in a sophisticated way, where there are a limited number of table view cell objects created in memory, even though there may be thousands of cells that are requested in the data source methods.

The table view class will create enough cells for each identifier to fill the frame of the table view, plus a few on each side. If the user is scrolling down for example, the table view cell at the top will go out of the view of the table view and be reused as a new cell appears from the bottom of the view of the table view. The dequeue reusable cell method will return a UITableViewCell which either has been reused or created new.

The identifier is any string which can be used to identify each type of cell which can be reused in a table view. If your cell was designed using the interface builder, which mine was, then the string which is passed into the dequeue reusable cell method must be exactly the same as set in the interface builder options for that cell. I know that my cell is always going to be of type InputLineTableViewCell so I can safely force unwrap it as this class type without causing a crash.

The diagram below demonstrates how table view cells work in a table view cell:

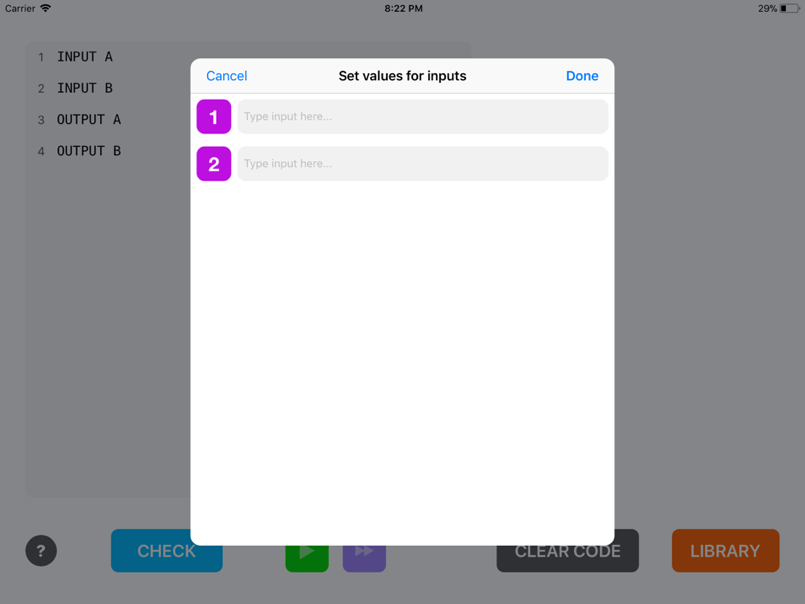


### Console and Controls

The console in the development screen is of type UITextField. This is the iOS view for showing long sections of text and is scrollable.

Each of the buttons along the bottom of the screen is of type UIButton (the iOS view for a button view). Each button is connected to the ViewController class through the use of the @IBAction tag. This tag works in a similar way to the @IBOutlet tag. An @IBOutlet will connect a view in the interface builder to a property whereas the @IBAction tag will connect a view in the interface builder to a method that should be called when a specific gesture is applied onto that view.

## Input Screen



The input screen’s purpose is to ask the user for a specified number of inputs and use those in the program when check is pressed.

The input screen is represented by the InputsTableViewController class. It inherits from UITableViewController. UITableViewController is a subclass of UIViewController which extends its functionality by providing a default implementation for a table view which fills the whole view controller view and automatically conforms to the delegate UITableViewDelegate and UITableViewDataSource.

The input screen also contains a UITableView with cells of type ProgramInputTableViewCell. Each table view cell contains a text field.

When the ‘Done’ button is pressed in the top right hand corner of the input screen, the private method fetchInputs() is called which will capture all the values from the text fields in each cell. The method allInputsAreValid() will check that none of the inputs were left empty.

Since the text from a text field will always be of type String, the method processInputs() will process the methods into either a number or a string with apostrophes surrounding the value.

If all inputs are present and valid, then the variable inputsReturned will be called. The variable is of a function type. This variable is set before the input view controller is initialized and will mean that the data for the inputs can be sent back to the ViewController class. The inputs array will be passed into the initializer for the ProgramInterpreter object as the inputs of the pseudo code program.

The type of inputsReturned is ([Any]) -> () which denotes a function which has an array of Any as a parameter and no return parameters.

If the inputs are not present or not valid, then a dialog box appears to say that there was an error.

## ../../../Simulator%20Screen%20Shot%20-%20iPad%20Air%202%20-%202018-04-23%20at%2022.07.23.pngError Detail Screen

The error details screen provides more information about whatever syntax error occurred in the user’s code.

Multiple errors are supported and web links to more information specifically for each type of error are supported. The errors are fetched from the errors property of the ProgramInterpreter class.

This view controller is represented by the class ErrorsTableViewController which inherits from UITableViewController. The view controller class has a property of type ErrorDetailTableViewHandler which is the delegate and data source for the table view. This will replace the ErrorsTableViewController using itself as the delegate and data source due to the default implementation of a subclass of UITableViewController.

The text with the red background which shows the contents of the line of code is created as a UIView and set to the header property of the table view. The header will show above all other items in the table view and is not affected by the data source. This is assigned in the addHeader() method in the view controller.

The errors are passed in from the ViewController class and displayed on this screen. Each error is of type Error, the message displayed is the message property, and if an arrow appears next to the table view cell, it means that there is a link. Clicking on the table view cell will take the user to the iOS browser which will open that link automatically.

## ../../../Simulator%20Screen%20Shot%20-%20iPad%20Air%202%20-%202018-04-23%20at%2022.30.40.pngLibrary Screen

This screen is represented by the LibraryTableViewController class, which inherits from UITableViewController. It uses itself as the delegate and data source for the table view, which is the default implementation.

This screen also uses a UITableView with 2 sections. The first section provides an option to save the code that is currently in the editor into memory. The second section provides a list of all previously saved code snippets.

Clicking the ‘Save current code’ row will show a pop-up which will ask the user to input a title which the code will be saved under. It will then save the code into memory under the key of the title.

The user can then select one of the options from the second section which will dismiss this screen and populate the editor with that saved code. A user can swipe from right to left on any of the title’s and show the option to delete, as pictured.

Saving the code makes use of the class UserDefaults. It is a permanent memory storage for as long as an app is installed and is generally used for small pieces of data, such as user preferences to change the behaviour of an app or for storing other small amounts of data. It works in the same way as a dictionary, where a value is set to a key. The key here is the title that the user chooses and the key is the array of lines of code.

Technical

# Interpreter

|  |
| --- |
| **Program.swift** |
| import Foundation  class Program {    // This is the singleton variable that all other classes will reference during  // the simulated runtime of the program.  static fileprivate(set) var shared: Program? = nil    // These are public properties  var outputs: [Any] = []  let instructions: [Instruction]  var variables: [String : Any]    // These are private properties that are needed for the execution of the program  fileprivate var inputs: [Any]  fileprivate var inputIndex = 0  fileprivate(set) var completedExecution: Bool  fileprivate(set) var runIndex = 0    // This will create a new Program object and set the singleton to this new object  // - @discardableResult means that the compiler will not show an error if the return value of this  // function is not used  @discardableResult static func resetProgram(withInterpreter interpreter: ProgramInterpreter) -> Program? {  let program = Program(interpreter: interpreter)  Program.shared = program  return program  }    init?(interpreter: ProgramInterpreter) {  guard interpreter.code.count > 0 && interpreter.canRun else {  return nil  }    var instructions = [Instruction]()  var queueParent: Instruction?  var queue = [Instruction]()  var addToQueue = false    for line in interpreter.code {  if let instruction = Instruction.createInstruction(fromLine: line) {    // Here, I organise each line of code into scopes according to the type of instruction they are.  // For example, all code within a for loop will fall within one scope and should be stored under  // the for instruction variable 'instructions'.  switch instruction {  case is ForInstruction:  addToQueue = true  queueParent = instruction  case is EndForInstruction:  addToQueue = false  if let queueParent = queueParent as? ForInstruction {  queueParent.instructions = queue  queue = []  instructions.append(queueParent)  }  case is IfInstruction:  addToQueue = true  queueParent = instruction  case is ElseInstruction:  addToQueue = true  (queueParent as? IfInstruction)?.trueBlock = queue  queue = []  case is EndIfInstruction:  addToQueue = false  if let queueParent = queueParent as? IfInstruction {  if queueParent.trueBlock.isEmpty {  queueParent.trueBlock = queue  } else {  queueParent.falseBlock = queue  }  queue = []  instructions.append(queueParent)  instructions.append(instruction)  }  case is RepeatInstruction:  addToQueue = true  instructions.append(instruction)  case is RepeatUntilInstruction:  addToQueue = false  (instruction as? RepeatUntilInstruction)?.instructions = queue  queue = []  instructions.append(instruction)  case is WhileInstruction:  addToQueue = true  queueParent = instruction  case is EndWhileInstruction:  if let queueParent = queueParent as? WhileInstruction {  queueParent.instructions = queue  instructions.append(queueParent)  }  queue = []  instructions.append(instruction)  default:  if addToQueue {  queue.append(instruction)  } else {  instructions.append(instruction)  }  }  }  }  self.instructions = instructions  self.inputs = interpreter.inputs  self.variables = [:]  self.runIndex = 0  self.completedExecution = false  }    func run() {  // This will protect the user pressing run again, or pressing step over line if they  // have already run the program.  guard !completedExecution else {  return  }    completedExecution = true  runIndex = 0  for instruction in instructions {  instruction.run()  }  outputs.append("\nCompleted execution.")  completedExecution = true  }    // This is the step by line method which will run each instruction one at a time.  func runNext() {  guard !completedExecution else {  return  }  if runIndex < instructions.count {  instructions[runIndex].run()  runIndex += 1  } else {  outputs.append("\nCompleted execution.")  completedExecution = false  }  }    // This will be called when the interpreter comes across an 'INPUT' command.  // If there are no more inputs available, it will return value: -1  func requestInput() -> Any {  if inputs.count > inputIndex {  let i = inputIndex  inputIndex += 1  return inputs[i]  }  return -1  }    } |

|  |
| --- |
| **ProgramInterpreter.swift** |
| import Foundation  class ProgramInterpreter {    // Private variables  fileprivate(set) var code: [String]  fileprivate(set) var inputs: [Any]    fileprivate(set) var variableNames: [String] = []  fileprivate(set) var errors: [[Error]]  fileprivate(set) var scopes: [Int] = []  fileprivate var nextScopeIndex: Int?    // A value that indicates whether the program is allowed to run or not  // depending on whether the code is valid or not  var canRun: Bool {  var errorCount = 0  errors.forEach({ errorCount += $0.count })  return errorCount == 0  }    init(code: [String], inputs: [Any]) {  self.code = code  self.inputs = inputs  self.errors = Array(repeating: [], count: code.count)  analyseCode()  }    // MARK: Identify Variables  // The following methods will parse through the string lines of code and detect/recognise certain  // types of command and value to determine whether the line of code is valid or not.    static func numberOfInputsRequired(code: [String]) -> Int {  var count = 0  for line in code {  if RegexExtractor.textMatches(line, forRegex: CommandRegexPattern.input) {  count += 1  }  }  return count  }    fileprivate func parseForInputs(line: String) {  if let varName = collectVariableName(forLine: line, regexPattern: CommandRegexPattern.input) {  appendVariableName(varName)  }  }    fileprivate func parseForVariableNames(line: String) {  if let varName = collectVariableName(forLine: line, regexPattern: CommandRegexPattern.assignment) {  appendVariableName(varName)  } else if let varName = collectVariableName(forLine: line, regexPattern: CommandRegexPattern.forLoop) {  appendVariableName(varName)  }  }    fileprivate func collectVariableName(forLine line: String, regexPattern: String) -> String? {  guard RegexExtractor.textMatches(line, forRegex: regexPattern) else {  return nil  }  let results = RegexExtractor.matches(forRegex: regexPattern, in: line)  return results[1]  }    fileprivate func appendVariableName(\_ varName: String) {  if !variableNames.contains(varName) {  variableNames.append(varName)  }  }    // MARK: Analyse lines    fileprivate func analyseCode() {  scopes = []  var index = 0  for line in code {  parseForInputs(line: line)  parseForVariableNames(line: line)    let expr = RegexSubstituter(command: line, varNames: variableNames).substitute()  validateLine(line: expr, index: index)  addScopeIndex(forLine: line)  index += 1  }  }    fileprivate func validateLine(line: String, index: Int) {  if let \_ = CommandParser(command: line).match {  return  }  let errs = errors(forLine: line)  errors[index] = errs  }    fileprivate func addScopeIndex(forLine line: String) {  if let nextScopeIndex = nextScopeIndex {  scopes.append(nextScopeIndex)  self.nextScopeIndex = nil  return  }    let scope = scopes.last ?? 0  guard let type = CommandParser(command: line).type else {  return  }  switch type {  case .ifStatement, .forLoop, .repeatStart, .whileStart:  nextScopeIndex = scope + 1  scopes.append(scope)  case .endForLoop, .endIfStatement, .repeatUntil, .endWhile:  scopes.append(scope - 1)  case .elseStatement:  nextScopeIndex = scope  scopes.append(scope - 1)  default:  scopes.append(scope)  }  }    fileprivate func errors(forLine line: String) -> [Error] {  return CodeErrorHandler.provideError(line: line)  }  } |

## Instructions

|  |
| --- |
| **Instruction.swift** |
| import Foundation  class Instruction {    var line: String    init(line: String) {  self.line = line  }    static func createInstruction(fromLine line: String) -> Instruction? {  guard let match = CommandParser(command: line).type else {  return nil  }    // This switch command determines the link between the CommandType enum  // and the type of Instruction subclass that is required.  switch match {  case .input: return InputInstruction(line: line)  case .output: return OutputInstruction(line: line)  case .assignment: return AssignmentInstruction(line: line)  case .forLoop: return ForInstruction(line: line)  case .endForLoop: return EndForInstruction(line: line)  case .ifStatement: return IfInstruction(line: line)  case .elseStatement: return ElseInstruction(line: line)  case .endIfStatement: return EndIfInstruction(line: line)  case .repeatStart: return RepeatInstruction(line: line)  case .repeatUntil: return RepeatUntilInstruction(line: line)  case .whileStart: return WhileInstruction(line: line)  case .endWhile: return EndWhileInstruction(line: line)  }  }    func run() {}    } |

|  |
| --- |
| **AssignmentInstruction.swift** |
| import Foundation  class AssignmentInstruction: Instruction {    // This class represents an instruction in the form:  // variable <- value    override func run() {  guard let match = CommandParser(command: line).match else {  return  }    if case .assignment(let targetVariableName, let value) = match {  run(targetVarName: targetVariableName, value: value)  }  }    fileprivate func run(targetVarName: String, value: Any) {  Program.shared!.variables[targetVarName] = value  }  } |

|  |
| --- |
| **InputInstruction.swift** |
| import Foundation  class InputInstruction: Instruction {    // This class represents an instruction in the form:  // INPUT variable    override func run() {  guard let match = CommandParser(command: line).match else {  return  }    if case let .input(variableName) = match {  run(withVariableName: variableName)  }  }    fileprivate func run(withVariableName varName: String) {  let input = Program.shared!.requestInput()  Program.shared!.variables[varName] = input  }  } |

|  |
| --- |
| **OutputInstruction.swift** |
| import Foundation  class OutputInstruction: Instruction {    // This class represents an instruction in the form:  // OUTPUT value    override func run() {  guard let match = CommandParser(command: line).match else {  return  }    if case let .output(value) = match {  Program.shared!.outputs.append(value)  }  }  } |

|  |
| --- |
| **ForInstruction.swift** |
| import Foundation  class ForInstruction: Instruction {    // This class represents an instruction in the form:  // FOR indexVariable <- startValue TO endValue  // code  // NEXT    var instructions: [Instruction] = []    override func run() {  guard let match = CommandParser(command: line).match else {  return  }    if case let .forLoop(indexVarName, startIndex, endIndex) = match {  for index in startIndex...endIndex {  Program.shared?.variables[indexVarName] = index  runBlock()  }  Program.shared?.variables[indexVarName] = nil  }  }    fileprivate func runBlock() {  for instruction in instructions {  instruction.run()  }  }  }  class EndForInstruction: Instruction {} |

|  |
| --- |
| **IfInstruction.swift** |
| import Foundation  // This class represents an instruction in the form:  // IF booleanValue THEN  // code  // END IF  class IfInstruction: Instruction {    var trueBlock: [Instruction] = []  var falseBlock: [Instruction] = []    override func run() {  guard let match = CommandParser(command: line).match else {  return  }    if case let .ifStatement(boolVal) = match {  let instructions = boolVal ? trueBlock : falseBlock  for instruction in instructions {  instruction.run()  }  }  }  }  class ElseInstruction: Instruction {}  class EndIfInstruction: Instruction {} |

|  |
| --- |
| **RepeatUntilInstruction.swift** |
| import UIKit  // This class represents an instruction in the form:  // REPEAT  // code  // UNTIL boolVal  class RepeatInstruction: Instruction {}  class RepeatUntilInstruction: Instruction {    var instructions: [Instruction] = []    override func run() {  guard let match = CommandParser(command: line).match else {  return  }    if case let .repeatUntil(boolExpr) = match {  performRepeatUntil(usingBoolExpr: boolExpr)  }  }    fileprivate func performRepeatUntil(usingBoolExpr boolExpr: String) {  for instruction in instructions {  instruction.run()  }  if !evaluate(boolExpr: boolExpr) {  performRepeatUntil(usingBoolExpr: boolExpr)  }  }    fileprivate func evaluate(boolExpr: String) -> Bool {  let substituted = RegexSubstituter(command: boolExpr, variables: Program.shared?.variables).substitute()  return BooleanExpressionParser(expr: substituted)?.calculate() ?? false  }  } |

|  |
| --- |
| **WhileInstruction.swift** |
| import UIKit  // This class represents an instruction in the form:  // WHILE boolVal  // code  // END WHILE  class WhileInstruction: Instruction {    var instructions: [Instruction] = []    override func run() {  guard let match = CommandParser(command: line).match else {  return  }    if case let .whileStart(boolExpr) = match {  performWhile(usingBoolExpr: boolExpr)  }  }    fileprivate func performWhile(usingBoolExpr boolExpr: String) {  if evaluate(boolExpr: boolExpr) {  for instruction in instructions {  instruction.run()  }  performWhile(usingBoolExpr: boolExpr)  }  }    fileprivate func evaluate(boolExpr: String) -> Bool {  let substituted = RegexSubstituter(command: boolExpr, variables: Program.shared?.variables).substitute()  return BooleanExpressionParser(expr: substituted)?.calculate() ?? false  }    }  class EndWhileInstruction: Instruction {} |

## Parsers

|  |
| --- |
| **CommandParser.swift** |
| import Foundation  enum CommandType {  case input, output  case assignment  case forLoop, endForLoop  case ifStatement, elseStatement, endIfStatement  case repeatStart, repeatUntil  case whileStart, endWhile  }  enum CommandMatch {  case input(String)  case output(Any)  case assignment(String, Any)  case forLoop(String, Int, Int)  case endForLoop  case ifStatement(Bool)  case elseStatement  case endIfStatement  case repeatStart  case repeatUntil(String)  case whileStart(String)  case endWhile  }  class CommandParser {    fileprivate let command: String  fileprivate let variables: [String : Any]    init(command: String, variables: [String : Any]? = nil) {  self.command = command  self.variables = variables ?? Program.shared?.variables ?? [:]  }    // This will return the CommandType enum type of the inputted string line of code  var type: CommandType? {  if textMatches(regex: CommandRegexPattern.input) { return .input }  if textMatches(regex: CommandRegexPattern.output) { return .output }  if textMatches(regex: CommandRegexPattern.assignment) { return .assignment }  if textMatches(regex: CommandRegexPattern.forLoop) { return .forLoop }  if textMatches(regex: CommandRegexPattern.endForLoop) { return .endForLoop }  if textMatches(regex: CommandRegexPattern.ifStatement) { return .ifStatement }  if textMatches(regex: CommandRegexPattern.elseStatement) { return .elseStatement }  if textMatches(regex: CommandRegexPattern.endIfStatement) { return .endIfStatement }  if textMatches(regex: CommandRegexPattern.repeatStatement) { return .repeatStart }  if textMatches(regex: CommandRegexPattern.repeatUntilStatement) { return .repeatUntil }  if textMatches(regex: CommandRegexPattern.whileStart) { return .whileStart }  if textMatches(regex: CommandRegexPattern.endWhile) { return .endWhile }  return nil  }    // This will return the CommandMatch enum type of the inputted string line of code  // which includes associated values for data inside the command.  var match: CommandMatch? {  guard let type = type else {  return nil  }  switch type {  case .input: return inputMatch()  case .output: return outputMatch()  case .assignment: return assignmentMatch()  case .forLoop: return forLoopMatch()  case .endForLoop: return .endForLoop  case .ifStatement: return ifStatementMatch()  case .elseStatement: return .elseStatement  case .endIfStatement: return .endIfStatement  case .repeatStart: return .repeatStart  case .repeatUntil: return repeatUntilMatch()  case .whileStart: return whileMatch()  case .endWhile: return .endWhile  }  }    // Below are standard helper functions which use regex to either recognise  // or substitute strings using the inputted line of code.    fileprivate func textMatches(regex: String) -> Bool {  return RegexExtractor.textMatches(command, forRegex: regex)  }    fileprivate func substituteVariables(intoExpr expr: String) -> String {  let substituted = RegexSubstituter(command: expr, variables: variables).substitute()  return substituted  }    // Uses either the ShuntingYardParser, StringExpressionParser or BooleanExpressionParser  // class to create a value for a substring which should represent a value of type either  // Int, String or Boolean  static func createValue(fromExpression expr: String) -> Any? {  if RegexExtractor.textMatches(expr, forRegex: CommandRegexPattern.integerExpression) {  let value = ShuntingYardParser(expr: expr).calculate()  return value  } else if RegexExtractor.textMatches(expr, forRegex: CommandRegexPattern.booleanValue) {  if expr == "TRUE" || expr == "true" { return true }  if expr == "FALSE" || expr == "false" { return false }  } else if RegexExtractor.textMatches(expr, forRegex: CommandRegexPattern.stringExpression) {  return StringExpressionParser(input: expr).evaluatedValue  } else if let parser = BooleanExpressionParser(expr: expr) {  return parser.calculate()  }  return nil  }    // MARK: Types  // Below are all the custom string manipulation methods to extract values from each type of command    fileprivate func inputMatch() -> CommandMatch {  let results = RegexExtractor.matches(forRegex: CommandRegexPattern.input, in: command)  let varName = results[1]  return .input(varName)  }    fileprivate func outputMatch() -> CommandMatch? {  let expr = substituteVariables(intoExpr: command)  let results = RegexExtractor.matches(forRegex: CommandRegexPattern.output, in: expr)  let valueExpr = results[1]    if let value = CommandParser.createValue(fromExpression: valueExpr) {  return .output(value)  }    return nil  }    fileprivate func assignmentMatch() -> CommandMatch? {  let expr = substituteVariables(intoExpr: command)  let results = RegexExtractor.matches(forRegex: CommandRegexPattern.assignment, in: expr)  let targetVarName = results[1]  let valueExpr = results[2]    if let value = CommandParser.createValue(fromExpression: valueExpr) {  return .assignment(targetVarName, value)  }    return nil  }    fileprivate func forLoopMatch() -> CommandMatch? {  let expr = substituteVariables(intoExpr: command)  let results = RegexExtractor.matches(forRegex: CommandRegexPattern.forLoop, in: expr)  let indexVarName = results[1]  let startIndexExpr = results[2]  let endIndexExpr = results[3]    let startIndexValid = RegexExtractor.textMatches(startIndexExpr, forRegex: CommandRegexPattern.integerExpression)  let endIndexValid = RegexExtractor.textMatches(endIndexExpr, forRegex: CommandRegexPattern.integerExpression)    guard startIndexValid && endIndexValid else {  return nil  }    let startIndex = ShuntingYardParser(expr: startIndexExpr).calculate()  let endIndex = ShuntingYardParser(expr: endIndexExpr).calculate()  return .forLoop(indexVarName, startIndex, endIndex)  }    fileprivate func ifStatementMatch() -> CommandMatch? {  let expr = substituteVariables(intoExpr: command)  let results = RegexExtractor.matches(forRegex: CommandRegexPattern.ifStatement, in: expr)  let boolExpr = results[1]  guard let boolVal = CommandParser.createValue(fromExpression: boolExpr) as? Bool else {  return nil  }  return .ifStatement(boolVal)  }    fileprivate func repeatUntilMatch() -> CommandMatch? {  let results = RegexExtractor.matches(forRegex: CommandRegexPattern.repeatUntilStatement, in: command)  let boolExpr = results[1]  return .repeatUntil(boolExpr)  }    fileprivate func whileMatch() -> CommandMatch? {  let results = RegexExtractor.matches(forRegex: CommandRegexPattern.whileStart, in: command)  let boolExpr = results[1]  return .whileStart(boolExpr)  }    } |

### Math Expression Parser

|  |
| --- |
| **ShuntingYardParser.swift** |
| import Foundation  // This class represents the working of the Shunting Yard algorithm  class ShuntingYardParser {    fileprivate let input = Queue<String>()  fileprivate let tokens = Queue<String>()  fileprivate let operators = Stack<Operator>()  fileprivate let output = Queue<String>()  fileprivate let validOperators = ["+", "-", "\*", "/", "%", "(", ")"]    var infix: String {  var expr = ""  input.forEach({ expr += $0 + " " })  return expr  }    var postfix: String {  return Polish(q: output).stringValue  }    init(expr: String) {  var expr = expr  if validOperators.contains(String(expr[expr.startIndex])) {  expr = "0" + expr  }  parse(expr: expr)  convertToPostfix()  }    func calculate() -> Int {  return Polish(q: output).calculate()  }    // MARK: Setup    fileprivate func parse(expr: String) {  var previousToken = ""  for character in expr {  let token = String(character)    if validOperators.contains(token) {  if !previousToken.isEmpty {  tokens.enqueue(item: previousToken)  }  tokens.enqueue(item: token)  previousToken = ""  } else if Int(token) != nil {  previousToken += token  }  }    if !previousToken.isEmpty {  tokens.enqueue(item: previousToken)  }    for token in tokens {  input.enqueue(item: token)  }  }    fileprivate func convertToPostfix() {  while !tokens.isEmpty {  let token = tokens.dequeue()!    if Int(token) != nil {  output.enqueue(item: token)  } else if validOperators.contains(token) {  let op = Operator(op: token)  var topOperator = operators.peek()    if token == ")" {  while topOperator != nil && topOperator!.op != "(" {  output.enqueue(item: topOperator!.op)  operators.pop()  topOperator = operators.peek()  }  operators.pop()  } else if token == "(" {  operators.push(item: op)  } else {  while topOperator != nil && topOperator!.precendence() > op.precendence() {  output.enqueue(item: topOperator!.op)  operators.pop()  topOperator = operators.peek()  }  operators.push(item: op)  }  }  }    while (!operators.isEmpty) {  if let op = operators.pop() {  output.enqueue(item: op.op)  }  }  }  } |

|  |
| --- |
| **Stack.swift** |
| // A custom implementation for a Stack  import Foundation  class Stack<T>: NSObject, Sequence {  typealias Element = T  typealias Iterator = StackIterator    fileprivate var last: Node?  fileprivate var N = 0    fileprivate class Node {  var item: T  var next: Node?  init(item: T, next: Node?) {  self.item = item  self.next = next  }  }    override var description: String {  var output = "Stack<\(T.self)>\n"  forEach({ output += String(describing: $0) + " " })  return output  }    var isEmpty: Bool {  return N < 1  }    var size: Int {  return N  }    func peek() -> T? {  if let last = last {  return last.item  }  return nil  }    func push(item: T) {  let oldLast = last  last = Node(item: item, next: oldLast)  N += 1  }    @discardableResult  func pop() -> T? {  let oldLast = last  last = oldLast?.next  N -= 1  return oldLast?.item  }    func makeIterator() -> Stack<T>.Iterator {  return StackIterator(last: last, N: N)  }    struct StackIterator: IteratorProtocol {  typealias Element = T  private var nextNode: Node?  private var i: Int    fileprivate init(last: Node?, N: Int) {  self.nextNode = last  self.i = N  }    mutating func next() -> T? {  guard i > 0 else {  return nil  }  let oldNext = nextNode  nextNode = oldNext?.next  i -= 1  return oldNext?.item  }    }  } |

|  |
| --- |
| **Operator.swift** |
| /\* A class which represents an Operator, possible types being:    (  )  +  -  \*  /  %    \*/  class Operator: NSObject {    fileprivate(set) var op: String    init(op: String) {  self.op = op  }    override var description: String {  if ["(", ")", "+", "-", "\*", "/", "%"].contains(op) {  return op  }  return "Unknown\_Operator"  }    func precendence() -> Int {  if op == "(" {  return 1  } else if op == "+" || op == "-" {  return 2  } else if op == "\*" || op == "/" || op == "%" {  return 3  }  return -1  }  } |

|  |
| --- |
| **Queue.swift** |
| import Foundation  // A custom implementation for a Queue  class Queue<T>: NSObject, Sequence {  typealias Element = T  typealias Iterator = QueueIterator    fileprivate var first: Node?  fileprivate var last: Node?  fileprivate(set) var N = 0    fileprivate class Node {  var item: T  var next: Node?  init(item: T) {  self.item = item  }  }    override var description: String {  var output = "Queue<\(T.self)>\n"  forEach({ output += String(describing: $0) + " " })  return output  }    var isEmpty: Bool {  return N == 0  }    func enqueue(item: T) {  let oldLast = last  last = Node(item: item)    if isEmpty {  first = last  } else {  oldLast?.next = last  }    N += 1  }    func dequeue() -> T? {  guard let first = first else {  return nil  }  let item = first.item  self.first = first.next  N -= 1    if isEmpty {  last = nil  }  return item  }    func makeIterator() -> Queue<T>.Iterator {  return QueueIterator(firstNode: first, N: N)  }    struct QueueIterator: IteratorProtocol {  typealias Element = T  private var firstNode: Node?  private var i: Int    fileprivate init(firstNode: Node?, N: Int) {  self.firstNode = firstNode  self.i = N  }    mutating func next() -> T? {  guard i > 0 else {  return nil  }  let oldFirst = firstNode  firstNode = firstNode?.next  i -= 1  return oldFirst?.item  }    }    } |

|  |
| --- |
| **Polish.swift** |
| import Foundation  class Polish {    fileprivate var stack = Stack<Int>()  fileprivate var queue: Queue<String>    init(q: Queue<String>) {  self.queue = q  }    var stringValue: String {  var str = ""  queue.forEach { str += $0 + " " }  return str  }    func calculate() -> Int {  for item in queue {  if let n = Int(item) {  stack.push(item: n)  } else {  if let right = stack.pop(), let left = stack.pop() {  switch item {  case "+": stack.push(item: left + right)  case "-": stack.push(item: left - right)  case "\*": stack.push(item: left \* right)  case "/":  if right == 0 {  complainNaNError()  stack.push(item: 0)  } else {  stack.push(item: left / right)  }  case "%":  if right == 0 {  complainNaNError()  stack.push(item: 0)  } else {  stack.push(item: left % right)  }  default: break  }  }  }  }    return stack.pop() ?? 0  }    fileprivate func complainNaNError() {  Program.shared?.outputs.append("NaN error! Returning 0 to prevent crash.")  }  } |

### String Expression Parser

|  |
| --- |
| **StringExpressionParser.swift** |
| import UIKit  class StringExpressionParser {    var input: String    init(input: String) {  self.input = input  }    var evaluatedValue: String? {  if let evaluated = evaluate(input) {  return "'\(evaluated)'"  }  return nil  }    fileprivate func evaluate(\_ input: String) -> String? {  let split = input.split(separator: "'")  if split.count % 2 == 0 {  // String addition should have an even number of ' characters for starting and ending strings.  return nil  }    var shouldAdd = true  var output = ""    for str in split {  let shortened = str.replacingOccurrences(of: " ", with: "")  switch shortened {  case "+":  shouldAdd = true  default:  if shouldAdd {  output += str  shouldAdd = false  } else {  return nil  }  }  }  return output  }  } |

### Boolean Expression Parser

|  |
| --- |
| **BooleanExpressionParser.swift** |
| import Foundation  fileprivate struct BooleanExprRegexPatterns {    static var integer: String {  return expression(value: CommandRegexPattern.integerExpression, operators: ["=","<",">","<=",">=","<>"])  }    static var string: String {  return expression(value: CommandRegexPattern.stringExpression, operators: ["=","<>"])  }    static var boolean: String {  return expression(value: CommandRegexPattern.booleanValue, operators: ["=","<>"])  }    private static func expression(value: String, operators: [String]) -> String {  let ops = operators.joined(separator: "|")  return "(\(value)) ?(\(ops)) ?(\(value))"  }    }  class BooleanExpressionParser {    fileprivate enum ParserType {  case integer, string, boolean  }    fileprivate var type: ParserType  fileprivate var lhs: Any  fileprivate var rhs: Any  fileprivate var op: String    fileprivate init?(regexResults results: [String], type: ParserType) {  guard let lhs = CommandParser.createValue(fromExpression: results[1]), let rhs = CommandParser.createValue(fromExpression: results[3]) else {  return nil  }  self.type = type  self.lhs = lhs  self.rhs = rhs  self.op = results[2]  }    convenience init?(expr: String) {  if RegexExtractor.textMatches(expr, forRegex: BooleanExprRegexPatterns.integer) {  self.init(regexResults: RegexExtractor.matches(forRegex: BooleanExprRegexPatterns.integer, in: expr), type: .integer)  return  }  if RegexExtractor.textMatches(expr, forRegex: BooleanExprRegexPatterns.string) {  self.init(regexResults: RegexExtractor.matches(forRegex: BooleanExprRegexPatterns.string, in: expr), type: .string)  return  }  if RegexExtractor.textMatches(expr, forRegex: BooleanExprRegexPatterns.boolean) {  self.init(regexResults: RegexExtractor.matches(forRegex: BooleanExprRegexPatterns.boolean, in: expr), type: .boolean)  return  }  return nil  }    func calculate() -> Bool? {  switch type {  case .integer: return calculateIntegerValue()  case .string: return calculateStringValue()  case .boolean: return calculateBooleanValue()  }  }    fileprivate func calculateIntegerValue() -> Bool? {  guard let lhs = lhs as? Int, let rhs = rhs as? Int else {  return nil  }  switch op {  case "=": return lhs == rhs  case "<": return lhs < rhs  case ">": return lhs > rhs  case "<=": return lhs <= rhs  case ">=": return lhs >= rhs  case "<>": return lhs != rhs  default: return nil  }  }    fileprivate func calculateStringValue() -> Bool? {  guard let lhs = lhs as? String, let rhs = rhs as? String else {  return nil  }  switch op {  case "==": return lhs == rhs  case "<>": return lhs != rhs  default: return nil  }  }    fileprivate func calculateBooleanValue() -> Bool? {  guard let lhs = lhs as? Bool, let rhs = rhs as? Bool else {  return nil  }  switch op {  case "==": return lhs == rhs  case "<>": return lhs != rhs  default: return nil  }  }  } |

## Regex

|  |
| --- |
| **RegexExtractor.swift** |
| import Foundation  struct CommandRegexPattern {  static var integerExpression = "(?:[0123456789()+-/\*%\\s])+"  static var booleanValue = "TRUE|FALSE"  static var stringExpression = "^(?:'.+' \\+ )\*(?:'.+')$"  static var variable = "[a-zA-Z\_$][a-zA-Z\_$0-9]\*"  static var input = "INPUT (\(variable))"  static var output = "OUTPUT (.+)"  static var assignment = "(\(variable)) <- (.+)"  static var forLoop = "FOR (\(variable)) <- (.+) TO (.+)"  static var endForLoop = "NEXT"  static var ifStatement = "IF (.+) THEN"  static var elseStatement = "ELSE"  static var endIfStatement = "END IF"  static var repeatStatement = "REPEAT"  static var repeatUntilStatement = "UNTIL (.+)"  static var whileStart = "WHILE (.+)"  static var endWhile = "END WHILE"  }  class RegexExtractor {    // Uses the iOS Framework class NSPredicate to determine whether a String (text) matches  // a regex pattern (regex)  static func textMatches(\_ text: String, forRegex regex: String) -> Bool {  return NSPredicate(format: "SELF MATCHES %@", regex).evaluate(with: text)  }    // Will return all the capture values from the regex pattern (regex) in the string (text).  // The first value will always be the entire string, and each capture value in order following that.  static func matches(forRegex regex: String, in text: String) -> [String] {  do {  let regex = try NSRegularExpression(pattern: regex)  let nsString = text as NSString  let results = regex.matches(in: text, range: NSRange(location: 0, length: nsString.length))  var a = [String]()  for i in 0..<results[0].numberOfRanges {  a.append(nsString.substring(with: results[0].range(at: i)))  }  return a  } catch let error {  print("Invalid regex: \(error.localizedDescription)")  return []  }  }    } |

|  |
| --- |
| **RegexSubstituter.swift** |
| import Foundation  class RegexSubstituter {    fileprivate let command: String  fileprivate let varNames: [String]?  fileprivate let variables: [String : Any]?    init(command: String, varNames: [String]? = nil, variables: [String : Any]? = nil) {  self.command = command  self.varNames = varNames  self.variables = variables  }    // This will substitute variables into string line of code commands.  // For example, the second line of code here:  // MyVariable <- "Hello, world!"  // OUTPUT MyVariable  // will be transformed into:  // OUTPUT "Hello, world!"    func substitute() -> String {  guard let type = CommandParser(command: command).type else {  return substituteExpression(command)  }    // If there were a variable in memory called OUTPUT, then that would be substituted automatically.  // Therefore for certain commands, we only substitute certain substrings of the line of code in order not  // to completely change the line of code.  switch type {  case .output: return substituteOutputExpression()  case .assignment: return substituteAssignmentExpression()  case .forLoop: return substituteForLoopExpression()  case .ifStatement: return substituteIfStatement()  default: return command  }  }    }  // MARK: - Substitutions  // Before are custom string manipulations using regex on each specific command type to substitute  // variables into commands.  extension RegexSubstituter {    fileprivate func substituteExpression(\_ expr: String) -> String {  var rhs = expr  if let varNames = varNames {  rhs = substituteVarNames(varNames, withValue: "0", intoExpression: rhs)  } else if let variables = variables {  rhs = substituteValues(variables, intoExpression: rhs)  }  return rhs  }    fileprivate func substituteOutputExpression() -> String {  let results = RegexExtractor.matches(forRegex: CommandRegexPattern.output, in: command)  let rhs = substituteExpression(results[1])  return command.replacingOccurrences(of: results[1], with: rhs)  }    fileprivate func substituteAssignmentExpression() -> String {  let results = RegexExtractor.matches(forRegex: CommandRegexPattern.assignment, in: command)  let rhs = substituteExpression(results[2])  return command.replacingOccurrences(of: results[2], with: rhs)  }    fileprivate func substituteForLoopExpression() -> String {  let results = RegexExtractor.matches(forRegex: CommandRegexPattern.forLoop, in: command)  var newExpr = command  newExpr = newExpr.replacingOccurrences(of: results[2], with: substituteExpression(results[2]))  newExpr = newExpr.replacingOccurrences(of: results[3], with: substituteExpression(results[3]))  return newExpr  }    fileprivate func substituteIfStatement() -> String {  let results = RegexExtractor.matches(forRegex: CommandRegexPattern.ifStatement, in: command)  let boolExpr = substituteExpression(results[1])  return command.replacingOccurrences(of: results[1], with: boolExpr)  }    }    // MARK: - Helpers  extension RegexSubstituter {    fileprivate func substituteValues(\_ values: [String : Any], intoExpression line: String) -> String {  var expr = line  values.forEach({ expr = RegexSubstituter.replaceOccurrences(of: $0.key, with: string(describing: $0.value), in: expr) })  return expr  }    fileprivate func substituteVarNames(\_ varNames: [String], withValue value: String, intoExpression line: String) -> String {  var expr = line  varNames.forEach({ expr = RegexSubstituter.replaceOccurrences(of: $0, with: string(describing: value), in: expr) })  return expr  }    fileprivate func string(describing value: Any) -> String {  var str = String(describing: value)  if ["true", "false"].contains(str) {  str = str.uppercased()  }  return str  }    fileprivate static func replaceOccurrences(of key: String, with replacement: String, in text: String) -> String {  do {  let pattern = "\\b\(key)\\b"  let regex = try NSRegularExpression(pattern: pattern)  return regex.stringByReplacingMatches(in: text, options: [], range: NSRange(location: 0, length: text.count), withTemplate: replacement)  } catch let error {  print("Invalid regex: \(error.localizedDescription)")  return text  }  }  } |

# Views

## Home

|  |
| --- |
| **ViewController.swift** |
| import UIKit  class ViewController: UIViewController {    // Identifiers that link to transitions between screens in the Main.storyboard file  fileprivate let errorDetailSegueIdentifier = "ErrorDetailSegue"  fileprivate let inputDetailSegueIdentifier = "InputDetailSegue"  fileprivate let librarySegueIdentifier = "LibrarySegue"    // Used to transfer data to error detail screen when selected  fileprivate var selectedIndex: Int?  fileprivate var selectedErrors: [Error]?    // Connections to user interface elements in the Main.storyboard file  @IBOutlet fileprivate var inputContainer: UIView!  @IBOutlet fileprivate var inputTableView: UITableView!  @IBOutlet fileprivate var outputTextView: UITextView!  @IBOutlet fileprivate var checkButton: UIButton!  @IBOutlet fileprivate var runButton: UIButton!  @IBOutlet fileprivate var nextButton: UIButton!  @IBOutlet fileprivate var clearButton: UIButton!  @IBOutlet fileprivate var libraryButton: UIButton!  @IBOutlet fileprivate var helpButton: UIButton!  @IBOutlet fileprivate var keyboardConstraint: NSLayoutConstraint!    // The delegate and data source for the table view  fileprivate var inputTableViewHandler: InputTableViewHandler!    // The program interpreter object which runs the functionality of the inputted pseudo code  fileprivate var programInterpreter: ProgramInterpreter?  // This method is called on UIViewController when the view is first loaded into memory  override func viewDidLoad() {  super.viewDidLoad()  inputTableViewHandler = InputTableViewHandler(tableView: inputTableView)  inputTableViewHandler.delegate = self  inputTableView.delegate = inputTableViewHandler  inputTableView.dataSource = inputTableViewHandler  styling()  setupKeyboard()  setProgramAbleToRun(false)  }    fileprivate func styling() {  inputContainer.layer.cornerRadius = 10  inputContainer.layer.masksToBounds = true  for btn in [checkButton, runButton, nextButton, clearButton, libraryButton, helpButton] {  btn!.layer.cornerRadius = 9  btn!.layer.masksToBounds = true  }  helpButton.layer.cornerRadius = helpButton.frame.size.height/2  }    // This method is called on UIViewController when it is about to transition to another screen  override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  // This segue identifier is the same as the segue identifiers which are defined at the start of this file  guard let segueIdentifier = segue.identifier else {  return  }  let vc = (segue.destination as? UINavigationController)?.viewControllers.first  switch segueIdentifier {  case errorDetailSegueIdentifier:  guard let vc = vc as? ErrorsTableViewController, let index = selectedIndex else { return }  vc.code = inputTableViewHandler.code[index]  vc.line = index  vc.errors = selectedErrors  selectedIndex = nil  selectedErrors = nil  case inputDetailSegueIdentifier:  guard let vc = vc as? InputsTableViewController else { return }  vc.numberOfInputs = ProgramInterpreter.numberOfInputsRequired(code: inputTableViewHandler.code)  vc.inputsReturned = inputsSuccessfullyReturned(inputs:)  case librarySegueIdentifier:  guard let vc = vc as? LibraryTableViewController else { return }  vc.currentCode = inputTableViewHandler.code  vc.loadCode = loadCode  default: break  }  }    fileprivate func setupKeyboard() {  // These closures will be called everytime the keyboard is shown/hidden so that the other views on the screen can be adjusted  NotificationCenter.default.addObserver(forName: NSNotification.Name.UIKeyboardWillShow, object: nil, queue: nil) { notification in  guard let keyboardSize = (notification.userInfo?[UIKeyboardFrameEndUserInfoKey] as? NSValue)?.cgRectValue else { return }  let mainScreenSize = UIScreen.main.bounds  // Sets the spacing between the views on the screen and the bottom of the screen  self.keyboardConstraint.constant = (mainScreenSize.height - keyboardSize.origin.y+30)  UIView.animate(withDuration: 0.25) {  self.view.layoutIfNeeded()  }  }  NotificationCenter.default.addObserver(forName: NSNotification.Name.UIKeyboardWillHide, object: nil, queue: nil) { notification in  self.keyboardConstraint.constant = 135  UIView.animate(withDuration: 0.25) { self.view.layoutIfNeeded() }  }  }    // MARK: Actions    fileprivate func setProgramAbleToRun(\_ able: Bool) {  runButton.isEnabled = able  nextButton.isEnabled = able  }    // Links to the 'CHECK' button in the Main.storyboard file, called when the button is tapped  @IBAction fileprivate func checkPressed() {  inputTableViewHandler.removeEmptyLines()  let numberOfInputs = ProgramInterpreter.numberOfInputsRequired(code: inputTableViewHandler.code)  if numberOfInputs > 0 {  performSegue(withIdentifier: inputDetailSegueIdentifier, sender: self)  } else {  inputsSuccessfullyReturned(inputs: [])  }  }    fileprivate func inputsSuccessfullyReturned(inputs: [Any]) {  let code = inputTableViewHandler.code  runProgram(code: code, inputs: inputs)  }    fileprivate func runProgram(code: [String], inputs: [Any]) {  let programInterpreter = ProgramInterpreter(code: code, inputs: inputs)  setProgramAbleToRun(programInterpreter.canRun)  self.programInterpreter = programInterpreter  if programInterpreter.canRun {  inputTableViewHandler.scopeIndexes = programInterpreter.scopes  inputTableView.reloadData()  logMessage("Program is ready to run.")  Program.resetProgram(withInterpreter: programInterpreter)  inputTableViewHandler.selectedIndex = 0  } else {  inputTableViewHandler.errors = programInterpreter.errors  inputTableView.reloadData()  logErrorMessage()  }  }    // Links to the 'RUN' button in the Main.storyboard file, called when the button is tapped  @IBAction fileprivate func runPressed() {  if let interpreter = programInterpreter {  inputTableViewHandler.selectedIndex = -1  Program.resetProgram(withInterpreter: interpreter)  Program.shared?.run()  logOutput(Program.shared!.outputs)  inputTableViewHandler.selectedIndex = -1  }  }    // Links to the 'NEXT' button in the Main.storyboard file, called when the button is tapped  @IBAction fileprivate func nextPressed() {  Program.shared?.runNext()  logOutput(Program.shared!.outputs)  inputTableViewHandler.selectedIndex += 1  if Program.shared?.completedExecution == true {  inputTableViewHandler.selectedIndex = -1  }  }    // Links to the 'CLEAR' button in the Main.storyboard file, called when the button is tapped  @IBAction fileprivate func clearPressed() {  inputTableViewHandler.selectedIndex = -1  inputTableViewHandler.clear()  }    // Links to the 'LIBRARY' button in the Main.storyboard file, called when the button is tapped  @IBAction fileprivate func libraryPressed() {  performSegue(withIdentifier: librarySegueIdentifier, sender: self)  }    // Links to the '?' button in the Main.storyboard file, called when the button is tapped  @IBAction fileprivate func helpPressed() {  let link = URL(string: "http://www.ocr.org.uk/Images/202654-pseudocode-guide.pdf")!  UIApplication.shared.open(link, options: [:], completionHandler: nil)  }    // MARK: - Save/load code    fileprivate func loadCode(code: [String]) {  inputTableViewHandler.clear()  inputTableViewHandler.code = code  inputTableView.reloadData()  }    // MARK: - Log To Console    // Will print a message to the on-screen console  fileprivate func logMessage(\_ message: String) {  outputTextView.text = message  outputTextView.flashScrollIndicators()  }    // Will print specifically an error message to the on-screen console, in the same format each time  fileprivate func logErrorMessage(\_ errorMessage: String? = nil) {  let extra = errorMessage == nil ? "" : "\n\n\(errorMessage!)"  logMessage("Error: Cannot run program!" + extra)  }    // Will log all of the outputs which are returned from running the program.  fileprivate func logOutput(\_ outputs: [Any]) {  var output = ""  for line in outputs {  output += String(describing: line)  output += "\n"  }  logMessage("Output:\n\(output)")  }  }  extension ViewController: InputTableViewHandlerDelegate {    // Is called everytime the text changes at a certain line of code at index 'index'  func textDidChange(atIndex index: Int) {  setProgramAbleToRun(false)  programInterpreter = nil  logMessage("")  }    // Is called when the error number button is pressed on a line of code  func didSelectErrors(atIndex index: Int) {  guard let errors = programInterpreter?.errors[index] else {  return  }  selectedIndex = index  selectedErrors = errors  performSegue(withIdentifier: "ErrorDetailSegue", sender: self)  }    } |

|  |
| --- |
| **InputTableViewHandler.swift** |
| import UIKit  protocol InputTableViewHandlerDelegate {  func textDidChange(atIndex index: Int)  func didSelectErrors(atIndex index: Int)  }  class InputTableViewHandler: NSObject {    // ViewController conforms to this delegate, so anytime a method is sent to this delegate, the ViewController instance will receive  var delegate: InputTableViewHandlerDelegate?    var scopeIndexes: [Int]?  var errors: [[Error]] = [[]]  var code: [String] = []  fileprivate var tableView: UITableView    // Identifies the currently highlighted line of code on screen in purple, if the user chooses to step by the program by line  // A -1 indicates no line should be selected  var selectedIndex: Int = -1 {  didSet {  tableView.reloadData()  }  }    init(tableView: UITableView) {  self.tableView = tableView  }    // Clears all the lines of code  func clear() {  code = []  scopeIndexes = nil  errors = [[]]  tableView.reloadData()  }    // Fixes a bug where when the keyboard is dismissed and there are empty lines of code  // and the user presses 'RUN' then there will be a compiler error for that line of code.  // Therefore all empty lines are removed.  func removeEmptyLines() {  var newCode = [String]()  for line in code {  if !line.isEmpty {  newCode.append(line)  }  }  code = newCode  tableView.reloadData()  }  }  extension InputTableViewHandler: UITableViewDelegate, UITableViewDataSource {    // All of the methods required by UITableView to function are below.    func numberOfSections(in tableView: UITableView) -> Int {  return 1  }    func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {  return max(code.count, 1)  }    func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {  var line: String?  if code.count > indexPath.row {  line = code[indexPath.row]  }    let cell = tableView.dequeueReusableCell(withIdentifier: "InputCell", for: indexPath) as! InputLineTableViewCell  var errorCount = 0  if errors.count > indexPath.row {  errorCount = errors[indexPath.row].count  }  cell.configure(withLine: line, errors: errorCount, indexPath: indexPath, selectedLine: selectedIndex == indexPath.row)  cell.setScope(scopeIndexes?[indexPath.row])  cell.delegate = self  return cell  }    func tableView(\_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {  return 40.0  }    func tableView(\_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {  tableView.deselectRow(at: indexPath, animated: true)  if errors.count > indexPath.row && errors[indexPath.row].count > 0 {  // Calls a delegate method which will be received by ViewController.  // This is when the user presses on one of the lines of code to show the errors for that line.  delegate?.didSelectErrors(atIndex: indexPath.row)  }  }    }  extension InputTableViewHandler: InputLineTableViewCellDelegate {    // This is called when the user presses return on the keyboard, indicating that he wishes to type in the next line of code.  // If the next line of code exists, it will be selected, else it will create a new one and recursively call this method again.  func inputLineDidReturn(atIndex index: Int) {  errors = [[]]  scopeIndexes = nil  if code.count > index+1 {  if let cell = tableView.cellForRow(at: IndexPath(row: index+1, section: 0)) as? InputLineTableViewCell {  cell.selectField()  }  } else {  code.append("")  tableView.reloadData()  inputLineDidReturn(atIndex: index)  }  }    // This is called anytime the text is changed at a line of code.  func inputLineDidChangeText(atIndex index: Int, text: String) {  errors = [[]]  scopeIndexes = nil  // Calls a delegate method which will be received by ViewController  delegate?.textDidChange(atIndex: index)  if code.count <= index {  code.append("")  inputLineDidChangeText(atIndex: index, text: text)  return  }  code[index] = text  }    } |

|  |
| --- |
| **InputLineTableViewCell.swift** |
| import UIKit  protocol InputLineTableViewCellDelegate {  func inputLineDidReturn(atIndex index: Int)  func inputLineDidChangeText(atIndex index: Int, text: String)  }  class InputLineTableViewCell: UITableViewCell {    // The ViewController class conforms to InputLineTableViewDelegate and will respond to all  // methods called from this delegate  var delegate: InputLineTableViewCellDelegate?    // Connections to user interface elements in the Main.storyboard file  @IBOutlet fileprivate var leftScopeConstraint: NSLayoutConstraint!  @IBOutlet fileprivate var lineNumberLabel: UILabel!  @IBOutlet fileprivate var textField: UITextField!  @IBOutlet fileprivate var errorContainer: UIView!  @IBOutlet fileprivate var errorLabel: UILabel!  fileprivate var indexPath: IndexPath!    // This method is called when the view has finished loading from the Main.storyboard file  override func awakeFromNib() {  super.awakeFromNib()  selectionStyle = .none  textField.delegate = self  textField.addTarget(self, action: #selector(textFieldDidChangeText(\_:)), for: .editingChanged)  errorContainer.layer.cornerRadius = errorContainer.frame.size.height/2    let accessory = UINib(nibName: "InputCodeKeyboardAccessory", bundle: nil).instantiate(withOwner: self, options: nil).first as! InputCodeKeyboardAccessory  accessory.frame = CGRect(x: 0, y: 0, width: UIScreen.main.bounds.size.width, height: 50.0)  accessory.didSelectWord = { word in  self.textField.text = self.textField.text! + word  self.textFieldDidChangeText(self.textField)  }  textField.inputAccessoryView = accessory  }    // The following methods adjust the appearance of the table view cell for the data provided.    func configure(withLine line: String?, errors: Int = 0, indexPath: IndexPath, selectedLine: Bool = false) {  self.indexPath = indexPath  backgroundColor = backgroundColor(forErrors: errors, selectedLine: selectedLine)  lineNumberLabel.text = "\(indexPath.row + 1)"  textField.text = line  errorLabel.text = "\(errors)"  errorContainer.isHidden = errors < 1  }    func setScope(\_ scope: Int?) {  if let scope = scope {  let multiplier: CGFloat = 20.0  leftScopeConstraint.constant = multiplier \* CGFloat(scope)  } else {  leftScopeConstraint.constant = 0.0  }  }    func selectField() {  textField.becomeFirstResponder()  }    fileprivate func backgroundColor(forErrors errors: Int, selectedLine: Bool) -> UIColor {  if errors > 0 {  return UIColor(red: 1, green: 0, blue: 0, alpha: 0.3)  } else if selectedLine {  return UIColor(red: 0, green: 0, blue: 255, alpha: 0.2)  } else {  return .clear  }  }    }  extension InputLineTableViewCell: UITextFieldDelegate {    // This delegate method of UITextFieldDelegate is called when the value of the text of the  // UITextField is changed  @objc func textFieldDidChangeText(\_ textField: UITextField) {  let text = textField.text ?? ""  // This delegate here is connected to the ViewController class  delegate?.inputLineDidChangeText(atIndex: indexPath.row, text: text)  }    // This delegate method of UITextFieldDelegate will be called everytime the user presses  // the return button on the keyboard.  func textFieldShouldReturn(\_ textField: UITextField) -> Bool {  guard textField.text != nil else {  return true  }  textField.resignFirstResponder()  if !textField.text!.isEmpty {  // This delegate here is connected to the ViewController class  delegate?.inputLineDidReturn(atIndex: indexPath.row)  }  return true  }    } |

|  |
| --- |
| **InputCodeKeyboardAccessory.swift** |
| import UIKit  class InputCodeKeyboardAccessory: UIView {    fileprivate let words = ["<-", "INPUT", "OUTPUT", "IF", "THEN", "ELSE", "END IF", "FOR", "TO", "NEXT", "WHILE", "END WHILE", "REPEAT", "UNTIL", "TRUE", "FALSE"]    // Connections to user interface elements in the Main.storyboard file  @IBOutlet fileprivate var collectionView: UICollectionView!    // This function is a property of the keyboard accessory class and is assigned  // to a closure in the InputCodeKeyboardAccessory class to respond to changes in  // the keyboard accessory  var didSelectWord: ((String) -> ())?    // This method is called when the view has finished loading from the Main.storyboard file  override func awakeFromNib() {  super.awakeFromNib()  collectionView.delegate = self  collectionView.dataSource = self  let nib = UINib(nibName: "InputCodeCollectionViewCell", bundle: Bundle.main)  collectionView.register(nib, forCellWithReuseIdentifier: "WordCell")  }    }  extension InputCodeKeyboardAccessory: UICollectionViewDelegate, UICollectionViewDataSource, UICollectionViewDelegateFlowLayout {    // A collection view works similar to a UITableView in that the data methods are similar but the appearance  // is free as opposed to constrained to vertical, screen-width rows.    func numberOfSections(in collectionView: UICollectionView) -> Int {  return 1  }    func collectionView(\_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {  return words.count  }    func collectionView(\_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {  let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "WordCell", for: indexPath) as! InputCodeCollectionViewCell  cell.configure(words[indexPath.row])  return cell  }    func collectionView(\_ collectionView: UICollectionView, didSelectItemAt indexPath: IndexPath) {  didSelectWord?(words[indexPath.row])  }    // This will determine the size of a cell in the collection view cell  func collectionView(\_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, sizeForItemAt indexPath: IndexPath) -> CGSize {  return CGSize(width: 150.0, height: 40.0)  }    // This will determine the horizontal spacing in between items in a section  func collectionView(\_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, minimumLineSpacingForSectionAt section: Int) -> CGFloat {  return 5.0  }    // This will determine the insets in all directions for a section in the collection view  func collectionView(\_ collectionView: UICollectionView, layout collectionViewLayout: UICollectionViewLayout, insetForSectionAt section: Int) -> UIEdgeInsets {  return UIEdgeInsetsMake(5, 5, 5, 5)  }    } |

|  |
| --- |
| **InputCodeCollectionViewCell.swift** |
| import UIKit  class InputCodeCollectionViewCell: UICollectionViewCell {    @IBOutlet fileprivate var container: UIView!  @IBOutlet fileprivate var label: UILabel!    // This method is called when the view has finished loading from the Main.storyboard file  override func awakeFromNib() {  super.awakeFromNib()  layer.masksToBounds = false  backgroundColor = .clear  container.layer.cornerRadius = 4  container.layer.masksToBounds = true  }    // Changes the appearance of the table view cell  func configure(\_ text: String) {  self.label.text = text  }    } |

## Input

|  |
| --- |
| **InputsTableViewController.swift** |
| import UIKit  class InputsTableViewController: UITableViewController {    var numberOfInputs: Int!  var inputsReturned: (([Any]) -> ())?    fileprivate func fetchInputs() -> [String] {  var inputs = [String]()  for i in 0..<tableView.numberOfRows(inSection: 0) {  let cell = tableView.cellForRow(at: IndexPath(row: i, section: 0)) as! ProgramInputTableViewCell  inputs.append(cell.inputtedText)  }  return inputs  }    fileprivate func allInputsAreValid() -> Bool {  let inputs = fetchInputs()  for input in inputs {  if input.isEmpty {  return false  }  }  return true  }    fileprivate func processInputs() -> [Any] {  let strInputs = fetchInputs()  var inputs = [Any]()  for input in strInputs {  if let integer = Int(input) {  inputs.append(integer)  } else {  inputs.append("'\(input)'")  }  }  return inputs  }    // This is connected to the 'CANCEL' button in the Main.storyboard and will be called when  // the button is tapped  @IBAction fileprivate func cancelPressed() {  dismiss(animated: true, completion: nil)  }    // This is connected to the 'DONE' button in the Main.storyboard and will be called when  // the button is tapped  @IBAction fileprivate func donePressed() {  if allInputsAreValid() {  inputsReturned?(processInputs())  dismiss(animated: true, completion: nil)  } else {  let alert = UIAlertController(title: "Error!", message: "Make sure you have entered valid inputs in all fields.", preferredStyle: .alert)  alert.addAction(UIAlertAction(title: "Okay", style: .default, handler: { (\_) in  alert.dismiss(animated: true, completion: nil)  }))  present(alert, animated: true, completion: nil)  }  }    // MARK: - Table view data source    override func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {  return numberOfInputs  }    override func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {  let cell = tableView.dequeueReusableCell(withIdentifier: "InputVariableCell", for: indexPath) as! ProgramInputTableViewCell  cell.configure(indexPath)  return cell  }    override func tableView(\_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {  return 60.0  }    } |

|  |
| --- |
| **PorgramInputTableViewCell.swift** |
| import UIKit  class ProgramInputTableViewCell: UITableViewCell {    // These UI elements are connected to UI elements in the Main.storybaord file  @IBOutlet fileprivate var numberLabel: UILabel!  @IBOutlet fileprivate var fieldContainer: UIView!  @IBOutlet fileprivate var field: UITextField!    var inputtedText: String {  return field.text ?? ""  }    // This method is called when the view has finished loading from the Main.storyboard file  override func awakeFromNib() {  super.awakeFromNib()  selectionStyle = .none  for view in [numberLabel, fieldContainer] {  view?.layer.cornerRadius = 10  view?.layer.masksToBounds = true  }  }    func configure(\_ indexPath: IndexPath) {  numberLabel.text = "\(indexPath.row + 1)"  }    } |

## Errors

|  |
| --- |
| **ErrorsTableViewController.swift** |
| import UIKit  class ErrorsTableViewController: UITableViewController {    var code: String!  var line: Int!  var errors: [Error]!    // The delegate and data source of the tableView in this view controller  // are set to this tableViewHandler property  fileprivate var tableViewHandler: ErrorDetailTableViewHandler!    override func viewDidLoad() {  super.viewDidLoad()  title = "Errors on line \(line!+1)"  tableViewHandler = ErrorDetailTableViewHandler(errors: errors)  addHeader()  tableView.rowHeight = 88.0  tableView.delegate = tableViewHandler  tableView.dataSource = tableViewHandler  tableView.reloadData()  }    fileprivate func addHeader() {  let label = UILabel(frame: CGRect(x: 0, y: 0, width: 1024, height: 80))  label.backgroundColor = UIColor(red: 255/255, green: 169/255, blue: 169/255, alpha: 1.0)  label.font = UIFont(name: "Hack", size: 37.0)!  label.textAlignment = .center  label.text = code  tableView.tableHeaderView = label  }    // This is connected to the Close button in the Main.storyboard file and will  // be called when it is tapped  @IBAction fileprivate func closePressed() {  navigationController?.dismiss(animated: true, completion: nil)  }    } |

|  |
| --- |
| **ErrorDetailTableViewHandler.swift** |
| import UIKit  class ErrorDetailTableViewHandler: NSObject {    fileprivate let cellIdentifier = "ErrorCell"  fileprivate let errors: [Error]    init(errors: [Error]) {  self.errors = errors  }  }  extension ErrorDetailTableViewHandler: UITableViewDelegate, UITableViewDataSource {    func numberOfSections(in tableView: UITableView) -> Int {  return 1  }    func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {  return errors.count  }    func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {  let cell = tableView.dequeueReusableCell(withIdentifier: cellIdentifier, for: indexPath) as! ErrorDetailTableViewCell  cell.configure(withError: errors[indexPath.row], index: indexPath.row+1)  return cell  }    func tableView(\_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {  tableView.deselectRow(at: indexPath, animated: true)  guard let link = errors[indexPath.row].link else {  return  }    // This will open the URL in the variable 'link'  // This will cause iOS to switch apps to the browser and automatically load this link  if UIApplication.shared.canOpenURL(link) {  UIApplication.shared.open(link, options: [:], completionHandler: nil)  }  }    } |

|  |
| --- |
| **ErrorDetailTableViewCell.swift** |
| import UIKit  class ErrorDetailTableViewCell: UITableViewCell {    // These are UI elements connected to UI elements in the Main.storyboard file  @IBOutlet fileprivate var indexLabel: UILabel!  @IBOutlet fileprivate var errorLabel: UILabel!    // Will configure the UI elements to reflect the data  func configure(withError error: Error, index: Int) {  self.indexLabel.text = "\(index)"  let extra = error.link == nil ? "" : "Tap this message to see more information."  self.errorLabel.text = "\(error.message) \(extra)"  selectionStyle = error.link == nil ? .none : .default  accessoryType = error.link == nil ? .none : .disclosureIndicator  }    } |

## Library

|  |
| --- |
| **LibraryTableViewController.swift** |
| import UIKit  class LibraryTableViewController: UITableViewController {    var currentCode: [String]!  var loadCode: (([String]) -> ())!    // This is the key for the library in the UserDefaults memory dictionary  fileprivate let libraryKey = "CodeLibrary"  fileprivate var collections: [String : [String]] = [:]  fileprivate var keys: [String] = []    override func viewDidLoad() {  super.viewDidLoad()  getCollections()  }    fileprivate func getCollections() {  collections = (UserDefaults.standard.dictionary(forKey: libraryKey) as? [String : [String]]) ?? [:]  keys = Array(collections.keys).sorted()  tableView.reloadData()  }    fileprivate func setCollections(\_ collections: [String : [String]]) {  UserDefaults.standard.set(collections, forKey: libraryKey)  self.collections = collections  self.keys = Array(collections.keys).sorted()  }    fileprivate func currentCodeIsValid() -> Bool {  return (currentCode.count > 1) || (currentCode.count > 0 && !currentCode[0].isEmpty)  }    // This will show a popup dialog that will ask the user to type in a title  // to save their code snippet under.  fileprivate func showNewNameAlert(\_ callback: @escaping ((String) -> ())) {  let alert = UIAlertController(title: "Save code", message: "Select a unique title for your code snippet.", preferredStyle: .alert)  alert.addTextField { textField in  textField.placeholder = "Title..."  }  alert.addAction(UIAlertAction(title: "Save", style: .default, handler: { \_ in  if let text = alert.textFields?[0].text, !text.isEmpty {  callback(text)  }  alert.dismiss(animated: true, completion: nil)  }))  alert.addAction(UIAlertAction(title: "Cancel", style: .cancel, handler: { \_ in  alert.dismiss(animated: true, completion: nil)  }))  present(alert, animated: true, completion: nil)  }    // This is connected to the Close button in the Main.storyboard and will be  // called when it is tapped  @IBAction fileprivate func closePressed() {  navigationController?.dismiss(animated: true, completion: nil)  }    // MARK: - Table view data source    override func numberOfSections(in tableView: UITableView) -> Int {  return 2  }    override func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {  return section == 0 ? 1 : keys.count  }    override func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {  let cell = tableView.dequeueReusableCell(withIdentifier: "LibraryCell", for: indexPath)  cell.textLabel?.text = indexPath.section == 0 ? "Save current code" : keys[indexPath.row]  return cell  }    override func tableView(\_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {  tableView.deselectRow(at: indexPath, animated: true)  if indexPath.section == 0 && currentCodeIsValid() {  showNewNameAlert { name in  self.collections[name] = self.currentCode  self.setCollections(self.collections)  self.tableView.reloadData()  }  } else if indexPath.section == 1 {  if let code = collections[keys[indexPath.row]] {  loadCode(code)  navigationController?.dismiss(animated: true, completion: nil)  }  }  }    override func tableView(\_ tableView: UITableView, canEditRowAt indexPath: IndexPath) -> Bool {  return indexPath.section == 1  }    // This enables the default iOS functionality to swipe right to left on a table view cell  // to reveal a delete button. When the delete button is tapped, the following code will run.  override func tableView(\_ tableView: UITableView, commit editingStyle: UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {  if editingStyle == .delete {  collections.removeValue(forKey: keys[indexPath.row])  setCollections(collections)  tableView.deleteRows(at: [indexPath], with: .automatic)  }  }    } |

# User Interface Files

|  |
| --- |
| **Main.storyboard** |
| <?xml version="1.0" encoding="UTF-8"?>  <document type="com.apple.InterfaceBuilder3.CocoaTouch.Storyboard.XIB" version="3.0" toolsVersion="13771" targetRuntime="iOS.CocoaTouch" propertyAccessControl="none" useAutolayout="YES" useTraitCollections="YES" useSafeAreas="YES" colorMatched="YES" initialViewController="BYZ-38-t0r">  <device id="ipad9\_7" orientation="landscape">  <adaptation id="fullscreen"/>  </device>  <dependencies>  <deployment identifier="iOS"/>  <plugIn identifier="com.apple.InterfaceBuilder.IBCocoaTouchPlugin" version="13772"/>  <capability name="Aspect ratio constraints" minToolsVersion="5.1"/>  <capability name="Constraints to layout margins" minToolsVersion="6.0"/>  <capability name="Safe area layout guides" minToolsVersion="9.0"/>  <capability name="documents saved in the Xcode 8 format" minToolsVersion="8.0"/>  </dependencies>  <customFonts key="customFonts">  <array key="Hack-Bold.otf">  <string>Hack-Bold</string>  </array>  <array key="Hack-Regular.otf">  <string>Hack</string>  </array>  <array key="HelveticaNeue.ttc">  <string>HelveticaNeue-Bold</string>  </array>  </customFonts>  <scenes>  <!--View Controller-->  <scene sceneID="tne-QT-ifu">  <objects>  <viewController id="BYZ-38-t0r" customClass="ViewController" customModule="Pseudo\_Code" customModuleProvider="target" sceneMemberID="viewController">  <view key="view" contentMode="scaleToFill" id="8bC-Xf-vdC">  <rect key="frame" x="0.0" y="0.0" width="1024" height="768"/>  <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>  <subviews>  <view contentMode="scaleToFill" translatesAutoresizingMaskIntoConstraints="NO" id="XNI-CT-Hgv" userLabel="Input Area">  <rect key="frame" x="32" y="52" width="568" height="581"/>  <subviews>  <tableView clipsSubviews="YES" contentMode="scaleToFill" alwaysBounceVertical="YES" dataMode="prototypes" style="plain" separatorStyle="none" rowHeight="-1" estimatedRowHeight="-1" sectionHeaderHeight="28" sectionFooterHeight="28" translatesAutoresizingMaskIntoConstraints="NO" id="eJO-UW-sZh">  <rect key="frame" x="0.0" y="0.0" width="568" height="581"/>  <color key="backgroundColor" white="0.0" alpha="0.0" colorSpace="calibratedWhite"/>  <prototypes>  <tableViewCell clipsSubviews="YES" contentMode="scaleToFill" preservesSuperviewLayoutMargins="YES" selectionStyle="default" indentationWidth="10" reuseIdentifier="InputCell" id="fhd-Rx-eh7" customClass="InputLineTableViewCell" customModule="Pseudo\_Code" customModuleProvider="target">  <rect key="frame" x="0.0" y="28" width="568" height="44"/>  <autoresizingMask key="autoresizingMask"/>  <tableViewCellContentView key="contentView" opaque="NO" clipsSubviews="YES" multipleTouchEnabled="YES" contentMode="center" preservesSuperviewLayoutMargins="YES" insetsLayoutMarginsFromSafeArea="NO" tableViewCell="fhd-Rx-eh7" id="ca7-3K-1gf">  <rect key="frame" x="0.0" y="0.0" width="568" height="44"/>  <autoresizingMask key="autoresizingMask"/>  <subviews>  <view contentMode="scaleToFill" translatesAutoresizingMaskIntoConstraints="NO" id="GwQ-BT-kRy">  <rect key="frame" x="512" y="5" width="34" height="34"/>  <subviews>  <label opaque="NO" userInteractionEnabled="NO" contentMode="left" horizontalHuggingPriority="251" verticalHuggingPriority="251" text="1" textAlignment="center" lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="aDx-Ed-VI9">  <rect key="frame" x="9.5" y="0.0" width="15" height="34"/>  <fontDescription key="fontDescription" type="system" weight="semibold" pointSize="22"/>  <color key="textColor" white="1" alpha="1" colorSpace="calibratedWhite"/>  <nil key="highlightedColor"/>  </label>  </subviews>  <color key="backgroundColor" red="1" green="0.0" blue="0.0" alpha="1" colorSpace="calibratedRGB"/>  <constraints>  <constraint firstAttribute="width" relation="greaterThanOrEqual" constant="34" id="99k-ou-9f6"/>  <constraint firstItem="aDx-Ed-VI9" firstAttribute="leading" secondItem="GwQ-BT-kRy" secondAttribute="leading" constant="10" id="Bsc-7Y-B1I"/>  <constraint firstItem="aDx-Ed-VI9" firstAttribute="top" secondItem="GwQ-BT-kRy" secondAttribute="top" id="MCg-a2-iDY"/>  <constraint firstAttribute="height" constant="34" id="VHX-Ue-2b5"/>  <constraint firstAttribute="bottom" secondItem="aDx-Ed-VI9" secondAttribute="bottom" id="ibG-ZY-cKk"/>  <constraint firstAttribute="trailing" secondItem="aDx-Ed-VI9" secondAttribute="trailing" constant="10" id="maZ-hY-qLQ"/>  </constraints>  </view>  <textField opaque="NO" contentMode="scaleToFill" contentHorizontalAlignment="left" contentVerticalAlignment="center" text="asd" placeholder="Type code here..." minimumFontSize="17" translatesAutoresizingMaskIntoConstraints="NO" id="OgN-iX-0U9">  <rect key="frame" x="40" y="0.0" width="460" height="44"/>  <nil key="textColor"/>  <fontDescription key="fontDescription" name="Hack" family="Hack" pointSize="17"/>  <textInputTraits key="textInputTraits" autocorrectionType="no" spellCheckingType="no" smartDashesType="no" smartInsertDeleteType="no" smartQuotesType="no"/>  </textField>  <label opaque="NO" userInteractionEnabled="NO" contentMode="left" horizontalHuggingPriority="251" verticalHuggingPriority="251" text="1" textAlignment="center" lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="RMD-MH-rER">  <rect key="frame" x="0.0" y="0.0" width="40" height="44"/>  <constraints>  <constraint firstAttribute="width" constant="40" id="FTX-CL-qgs"/>  </constraints>  <fontDescription key="fontDescription" type="system" pointSize="15"/>  <color key="textColor" white="0.33333333333333331" alpha="1" colorSpace="custom" customColorSpace="genericGamma22GrayColorSpace"/>  <nil key="highlightedColor"/>  </label>  </subviews>  <constraints>  <constraint firstItem="RMD-MH-rER" firstAttribute="leading" secondItem="ca7-3K-1gf" secondAttribute="leading" id="1cv-R8-sUH"/>  <constraint firstAttribute="bottom" secondItem="OgN-iX-0U9" secondAttribute="bottom" id="5Dq-tX-8sL"/>  <constraint firstAttribute="trailing" secondItem="GwQ-BT-kRy" secondAttribute="trailing" constant="22" id="CBJ-Lg-PVI"/>  <constraint firstItem="OgN-iX-0U9" firstAttribute="top" secondItem="ca7-3K-1gf" secondAttribute="top" id="JbZ-SS-WC3"/>  <constraint firstItem="GwQ-BT-kRy" firstAttribute="leading" secondItem="OgN-iX-0U9" secondAttribute="trailing" constant="12" id="NIW-bI-nVw"/>  <constraint firstItem="GwQ-BT-kRy" firstAttribute="centerY" secondItem="ca7-3K-1gf" secondAttribute="centerY" id="no4-Zx-q02"/>  <constraint firstAttribute="bottom" secondItem="RMD-MH-rER" secondAttribute="bottom" id="ojN-d8-Rbs"/>  <constraint firstItem="RMD-MH-rER" firstAttribute="top" secondItem="ca7-3K-1gf" secondAttribute="top" id="sa4-dl-sM7"/>  <constraint firstItem="OgN-iX-0U9" firstAttribute="leading" secondItem="RMD-MH-rER" secondAttribute="trailing" id="yDn-uv-VeJ"/>  </constraints>  </tableViewCellContentView>  <color key="backgroundColor" red="1" green="0.0" blue="0.0" alpha="0.29704607664233579" colorSpace="calibratedRGB"/>  <connections>  <outlet property="errorContainer" destination="GwQ-BT-kRy" id="ctf-zj-NuD"/>  <outlet property="errorLabel" destination="aDx-Ed-VI9" id="jOb-63-JFp"/>  <outlet property="leftScopeConstraint" destination="yDn-uv-VeJ" id="ZYG-wa-chq"/>  <outlet property="lineNumberLabel" destination="RMD-MH-rER" id="fV5-qU-gUk"/>  <outlet property="textField" destination="OgN-iX-0U9" id="blN-Ae-kxH"/>  </connections>  </tableViewCell>  </prototypes>  </tableView>  </subviews>  <color key="backgroundColor" red="0.95686274509803915" green="0.95686274509803915" blue="0.95686274509803915" alpha="1" colorSpace="calibratedRGB"/>  <constraints>  <constraint firstItem="eJO-UW-sZh" firstAttribute="top" secondItem="XNI-CT-Hgv" secondAttribute="top" id="5lN-LN-QaR"/>  <constraint firstAttribute="bottom" secondItem="eJO-UW-sZh" secondAttribute="bottom" id="U6g-0P-uJ5"/>  <constraint firstItem="eJO-UW-sZh" firstAttribute="leading" secondItem="XNI-CT-Hgv" secondAttribute="leading" id="aL7-hM-AlS"/>  <constraint firstAttribute="trailing" secondItem="eJO-UW-sZh" secondAttribute="trailing" id="uL7-hZ-bdV"/>  </constraints>  </view>  <view contentMode="scaleToFill" translatesAutoresizingMaskIntoConstraints="NO" id="D3u-RX-EaT" userLabel="Controls">  <rect key="frame" x="141" y="673" width="350" height="55"/>  <subviews>  <button opaque="NO" contentMode="scaleToFill" contentHorizontalAlignment="center" contentVerticalAlignment="center" buttonType="roundedRect" lineBreakMode="middleTruncation" translatesAutoresizingMaskIntoConstraints="NO" id="GNg-Fx-2wi">  <rect key="frame" x="0.0" y="0.0" width="142" height="55"/>  <color key="backgroundColor" red="0.0" green="0.6588235294117647" blue="1" alpha="1" colorSpace="calibratedRGB"/>  <constraints>  <constraint firstAttribute="width" relation="greaterThanOrEqual" constant="142" id="2kR-a0-AbJ"/>  </constraints>  <fontDescription key="fontDescription" type="system" weight="semibold" pointSize="22"/>  <state key="normal" title="CHECK">  <color key="titleColor" white="1" alpha="1" colorSpace="calibratedWhite"/>  </state>  <connections>  <action selector="checkPressed" destination="BYZ-38-t0r" eventType="touchUpInside" id="yv6-VL-zsE"/>  </connections>  </button>  <button opaque="NO" contentMode="scaleToFill" contentHorizontalAlignment="center" contentVerticalAlignment="center" lineBreakMode="middleTruncation" translatesAutoresizingMaskIntoConstraints="NO" id="Fja-cM-Wsx">  <rect key="frame" x="295" y="0.0" width="55" height="55"/>  <color key="backgroundColor" red="0.54509803921568623" green="0.40784313725490196" blue="1" alpha="1" colorSpace="calibratedRGB"/>  <constraints>  <constraint firstAttribute="width" secondItem="Fja-cM-Wsx" secondAttribute="height" multiplier="1:1" id="YTm-nS-2Jo"/>  </constraints>  <state key="normal" image="fast\_forward"/>  <connections>  <action selector="nextPressed" destination="BYZ-38-t0r" eventType="touchUpInside" id="KMd-U2-19P"/>  </connections>  </button>  <button opaque="NO" contentMode="scaleToFill" contentHorizontalAlignment="center" contentVerticalAlignment="center" lineBreakMode="middleTruncation" translatesAutoresizingMaskIntoConstraints="NO" id="gIK-6S-Q79">  <rect key="frame" x="222" y="0.0" width="55" height="55"/>  <color key="backgroundColor" red="0.0" green="0.83921568627450982" blue="0.027450980392156862" alpha="1" colorSpace="calibratedRGB"/>  <constraints>  <constraint firstAttribute="width" secondItem="gIK-6S-Q79" secondAttribute="height" multiplier="1:1" id="4dD-zS-4qK"/>  </constraints>  <state key="normal" image="play"/>  <connections>  <action selector="runPressed" destination="BYZ-38-t0r" eventType="touchUpInside" id="iYm-AG-IfN"/>  </connections>  </button>  </subviews>  <color key="backgroundColor" white="1" alpha="1" colorSpace="calibratedWhite"/>  <constraints>  <constraint firstAttribute="bottom" secondItem="gIK-6S-Q79" secondAttribute="bottom" id="9wu-mY-0lc"/>  <constraint firstItem="gIK-6S-Q79" firstAttribute="leading" secondItem="GNg-Fx-2wi" secondAttribute="trailing" constant="80" id="WKt-kX-8Zh"/>  <constraint firstItem="Fja-cM-Wsx" firstAttribute="top" secondItem="D3u-RX-EaT" secondAttribute="top" id="Zec-J9-Beq"/>  <constraint firstAttribute="height" constant="55" id="b09-bi-n0j"/>  <constraint firstItem="GNg-Fx-2wi" firstAttribute="leading" secondItem="D3u-RX-EaT" secondAttribute="leading" id="eC2-VL-YKb"/>  <constraint firstAttribute="trailing" secondItem="Fja-cM-Wsx" secondAttribute="trailing" id="ftw-zX-STN"/>  <constraint firstAttribute="bottom" secondItem="Fja-cM-Wsx" secondAttribute="bottom" id="gHk-Ex-R4G"/>  <constraint firstItem="Fja-cM-Wsx" firstAttribute="leading" secondItem="gIK-6S-Q79" secondAttribute="trailing" constant="18" id="lWn-YT-xrU"/>  <constraint firstItem="gIK-6S-Q79" firstAttribute="top" secondItem="D3u-RX-EaT" secondAttribute="top" id="shF-jI-zce"/>  <constraint firstAttribute="bottom" secondItem="GNg-Fx-2wi" secondAttribute="bottom" id="shv-UC-ndm"/>  <constraint firstItem="GNg-Fx-2wi" firstAttribute="top" secondItem="D3u-RX-EaT" secondAttribute="top" id="xCz-qd-bIj"/>  </constraints>  </view>  <view clipsSubviews="YES" contentMode="scaleToFill" translatesAutoresizingMaskIntoConstraints="NO" id="xQd-KY-yJn" userLabel="Console">  <rect key="frame" x="632" y="52" width="360" height="581"/>  <subviews>  <textView clipsSubviews="YES" multipleTouchEnabled="YES" contentMode="scaleToFill" editable="NO" textAlignment="natural" selectable="NO" translatesAutoresizingMaskIntoConstraints="NO" id="fUg-Z6-sE3">  <rect key="frame" x="8" y="8" width="344" height="565"/>  <color key="backgroundColor" white="1" alpha="1" colorSpace="custom" customColorSpace="genericGamma22GrayColorSpace"/>  <fontDescription key="fontDescription" name="Hack-Bold" family="Hack" pointSize="13"/>  <textInputTraits key="textInputTraits" autocorrectionType="no" spellCheckingType="no" keyboardAppearance="light" smartDashesType="no" smartInsertDeleteType="no" smartQuotesType="no"/>  </textView>  </subviews>  <color key="backgroundColor" white="1" alpha="1" colorSpace="calibratedWhite"/>  <constraints>  <constraint firstAttribute="trailingMargin" secondItem="fUg-Z6-sE3" secondAttribute="trailing" id="Lda-db-EAj"/>  <constraint firstAttribute="width" constant="360" id="i3y-Ew-Z1f"/>  <constraint firstItem="fUg-Z6-sE3" firstAttribute="leading" secondItem="xQd-KY-yJn" secondAttribute="leadingMargin" id="jk9-9I-aOU"/>  <constraint firstItem="fUg-Z6-sE3" firstAttribute="top" secondItem="xQd-KY-yJn" secondAttribute="topMargin" id="nkJ-FM-cTK"/>  <constraint firstAttribute="bottomMargin" secondItem="fUg-Z6-sE3" secondAttribute="bottom" id="pzR-6d-XBr"/>  </constraints>  </view>  <button opaque="NO" contentMode="scaleToFill" contentHorizontalAlignment="center" contentVerticalAlignment="center" buttonType="roundedRect" lineBreakMode="middleTruncation" translatesAutoresizingMaskIntoConstraints="NO" id="fTl-Hy-O5A">  <rect key="frame" x="632" y="673" width="181" height="55"/>  <color key="backgroundColor" white="0.33333333333333331" alpha="1" colorSpace="custom" customColorSpace="genericGamma22GrayColorSpace"/>  <constraints>  <constraint firstAttribute="height" constant="55" id="O75-BH-lDt"/>  <constraint firstAttribute="width" constant="181" id="ocX-Y4-hY5"/>  </constraints>  <fontDescription key="fontDescription" type="system" weight="semibold" pointSize="22"/>  <state key="normal" title="CLEAR CODE">  <color key="titleColor" white="1" alpha="1" colorSpace="custom" customColorSpace="genericGamma22GrayColorSpace"/>  </state>  <connections>  <action selector="clearPressed" destination="BYZ-38-t0r" eventType="touchUpInside" id="bNf-Az-bsj"/>  </connections>  </button>  <button opaque="NO" contentMode="scaleToFill" contentHorizontalAlignment="center" contentVerticalAlignment="center" buttonType="roundedRect" lineBreakMode="middleTruncation" translatesAutoresizingMaskIntoConstraints="NO" id="NNh-Fq-4GK">  <rect key="frame" x="855" y="673" width="137" height="55"/>  <color key="backgroundColor" red="1" green="0.38150919430000002" blue="0.0" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>  <constraints>  <constraint firstAttribute="height" constant="55" id="ChT-PO-FmZ"/>  <constraint firstAttribute="width" constant="137" id="KjT-um-4gA"/>  </constraints>  <fontDescription key="fontDescription" type="system" weight="semibold" pointSize="22"/>  <state key="normal" title="LIBRARY">  <color key="titleColor" white="1" alpha="1" colorSpace="custom" customColorSpace="genericGamma22GrayColorSpace"/>  </state>  <connections>  <action selector="libraryPressed" destination="BYZ-38-t0r" eventType="touchUpInside" id="Gnu-V0-80X"/>  </connections>  </button>  <button opaque="NO" contentMode="scaleToFill" contentHorizontalAlignment="center" contentVerticalAlignment="center" buttonType="roundedRect" lineBreakMode="middleTruncation" translatesAutoresizingMaskIntoConstraints="NO" id="kGR-Dl-yZq">  <rect key="frame" x="32" y="680" width="40" height="40"/>  <color key="backgroundColor" red="0.33300891518592834" green="0.33328679203987122" blue="0.33305191993713379" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>  <constraints>  <constraint firstAttribute="width" constant="40" id="bwB-qH-uoy"/>  <constraint firstAttribute="height" constant="40" id="pvQ-Ip-Rc1"/>  </constraints>  <fontDescription key="fontDescription" type="system" weight="heavy" pointSize="21"/>  <state key="normal" title="?">  <color key="titleColor" white="1" alpha="1" colorSpace="custom" customColorSpace="genericGamma22GrayColorSpace"/>  </state>  <connections>  <action selector="helpPressed" destination="BYZ-38-t0r" eventType="touchUpInside" id="dUZ-rg-GVU"/>  </connections>  </button>  </subviews>  <color key="backgroundColor" red="1" green="1" blue="1" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>  <constraints>  <constraint firstItem="xQd-KY-yJn" firstAttribute="bottom" secondItem="eJO-UW-sZh" secondAttribute="bottom" id="0mo-BQ-Pd5"/>  <constraint firstItem="fTl-Hy-O5A" firstAttribute="leading" secondItem="xQd-KY-yJn" secondAttribute="leading" id="4Vc-8q-7QR"/>  <constraint firstItem="6Tk-OE-BBY" firstAttribute="bottom" secondItem="D3u-RX-EaT" secondAttribute="bottom" constant="40" id="7nG-6A-GFX"/>  <constraint firstItem="xQd-KY-yJn" firstAttribute="top" secondItem="6Tk-OE-BBY" secondAttribute="top" constant="32" id="8FW-lD-GYo"/>  <constraint firstItem="NNh-Fq-4GK" firstAttribute="centerY" secondItem="fTl-Hy-O5A" secondAttribute="centerY" id="C74-yx-dRl"/>  <constraint firstItem="XNI-CT-Hgv" firstAttribute="top" secondItem="6Tk-OE-BBY" secondAttribute="top" constant="32" id="Cn2-mY-89I"/>  <constraint firstItem="NNh-Fq-4GK" firstAttribute="trailing" secondItem="xQd-KY-yJn" secondAttribute="trailing" id="E3Y-vA-kP1"/>  <constraint firstItem="xQd-KY-yJn" firstAttribute="leading" secondItem="XNI-CT-Hgv" secondAttribute="trailing" constant="32" id="Hbw-gH-iqj"/>  <constraint firstItem="XNI-CT-Hgv" firstAttribute="leading" secondItem="6Tk-OE-BBY" secondAttribute="leading" constant="32" id="LmU-tL-wfl"/>  <constraint firstItem="kGR-Dl-yZq" firstAttribute="leading" secondItem="eJO-UW-sZh" secondAttribute="leading" id="QvW-ut-xdh"/>  <constraint firstItem="D3u-RX-EaT" firstAttribute="centerX" secondItem="XNI-CT-Hgv" secondAttribute="centerX" id="b30-m3-KAH"/>  <constraint firstItem="6Tk-OE-BBY" firstAttribute="bottom" secondItem="XNI-CT-Hgv" secondAttribute="bottom" constant="135" id="eG3-25-mfA"/>  <constraint firstItem="fTl-Hy-O5A" firstAttribute="centerY" secondItem="D3u-RX-EaT" secondAttribute="centerY" id="ii6-SL-oDu"/>  <constraint firstItem="6Tk-OE-BBY" firstAttribute="trailing" secondItem="xQd-KY-yJn" secondAttribute="trailing" constant="32" id="k75-6P-ZWy"/>  <constraint firstItem="kGR-Dl-yZq" firstAttribute="centerY" secondItem="D3u-RX-EaT" secondAttribute="centerY" id="xeg-vd-zgD"/>  </constraints>  <viewLayoutGuide key="safeArea" id="6Tk-OE-BBY"/>  </view>  <connections>  <outlet property="checkButton" destination="GNg-Fx-2wi" id="sje-AE-eV3"/>  <outlet property="clearButton" destination="fTl-Hy-O5A" id="Nf6-OJ-rQm"/>  <outlet property="helpButton" destination="kGR-Dl-yZq" id="lwR-PU-XVW"/>  <outlet property="inputContainer" destination="XNI-CT-Hgv" id="y3D-Oj-qJv"/>  <outlet property="inputTableView" destination="eJO-UW-sZh" id="jIl-Gn-n14"/>  <outlet property="keyboardConstraint" destination="eG3-25-mfA" id="OW7-iO-Lze"/>  <outlet property="libraryButton" destination="NNh-Fq-4GK" id="FEU-Ia-v7S"/>  <outlet property="nextButton" destination="Fja-cM-Wsx" id="6IH-f0-iwT"/>  <outlet property="outputTextView" destination="fUg-Z6-sE3" id="6S9-6c-wGB"/>  <outlet property="runButton" destination="gIK-6S-Q79" id="5yh-SZ-HtH"/>  <segue destination="8RR-ZM-3pp" kind="presentation" identifier="ErrorDetailSegue" modalPresentationStyle="formSheet" id="kDC-DH-HPM"/>  <segue destination="1al-l5-IB9" kind="presentation" identifier="InputDetailSegue" modalPresentationStyle="formSheet" id="oNf-ry-Mnc"/>  <segue destination="4ta-gw-4dp" kind="presentation" identifier="LibrarySegue" modalPresentationStyle="formSheet" id="zkY-bW-M1U"/>  </connections>  </viewController>  <placeholder placeholderIdentifier="IBFirstResponder" id="dkx-z0-nzr" sceneMemberID="firstResponder"/>  </objects>  <point key="canvasLocation" x="-565.4296875" y="132.8125"/>  </scene>  <!--Errors-->  <scene sceneID="9gY-zw-H6L">  <objects>  <tableViewController id="FLi-iA-KhU" customClass="ErrorsTableViewController" customModule="Pseudo\_Code" customModuleProvider="target" sceneMemberID="viewController">  <tableView key="view" clipsSubviews="YES" contentMode="scaleToFill" alwaysBounceVertical="YES" dataMode="prototypes" style="grouped" separatorStyle="default" rowHeight="-1" estimatedRowHeight="-1" sectionHeaderHeight="18" sectionFooterHeight="18" id="VR2-Yo-8pI">  <rect key="frame" x="0.0" y="0.0" width="540" height="620"/>  <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>  <color key="backgroundColor" cocoaTouchSystemColor="groupTableViewBackgroundColor"/>  <prototypes>  <tableViewCell contentMode="scaleToFill" selectionStyle="default" indentationWidth="10" reuseIdentifier="ErrorCell" id="cg6-W7-cd6" customClass="ErrorDetailTableViewCell" customModule="Pseudo\_Code" customModuleProvider="target">  <rect key="frame" x="0.0" y="55.5" width="540" height="44"/>  <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>  <tableViewCellContentView key="contentView" opaque="NO" clipsSubviews="YES" multipleTouchEnabled="YES" contentMode="center" tableViewCell="cg6-W7-cd6" id="H1p-VJ-UJC">  <rect key="frame" x="0.0" y="0.0" width="540" height="43.5"/>  <autoresizingMask key="autoresizingMask"/>  <subviews>  <label opaque="NO" userInteractionEnabled="NO" contentMode="left" horizontalHuggingPriority="252" verticalHuggingPriority="251" text="1." textAlignment="natural" lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="DM7-ZN-r9N">  <rect key="frame" x="16" y="0.0" width="13.5" height="43.5"/>  <fontDescription key="fontDescription" type="boldSystem" pointSize="17"/>  <nil key="textColor"/>  <nil key="highlightedColor"/>  </label>  <label opaque="NO" userInteractionEnabled="NO" contentMode="left" horizontalHuggingPriority="251" verticalHuggingPriority="251" text="Label" textAlignment="natural" lineBreakMode="tailTruncation" numberOfLines="0" baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="rAF-gQ-Cfc">  <rect key="frame" x="37.5" y="0.0" width="486.5" height="43.5"/>  <fontDescription key="fontDescription" type="system" weight="light" pointSize="17"/>  <nil key="textColor"/>  <nil key="highlightedColor"/>  </label>  </subviews>  <constraints>  <constraint firstAttribute="bottom" secondItem="DM7-ZN-r9N" secondAttribute="bottom" id="MfE-SG-gvo"/>  <constraint firstItem="rAF-gQ-Cfc" firstAttribute="leading" secondItem="DM7-ZN-r9N" secondAttribute="trailing" constant="8" id="N7q-bR-lxy"/>  <constraint firstAttribute="bottom" secondItem="rAF-gQ-Cfc" secondAttribute="bottom" id="eQN-aK-GAC"/>  <constraint firstItem="DM7-ZN-r9N" firstAttribute="top" secondItem="H1p-VJ-UJC" secondAttribute="top" id="enY-43-XtS"/>  <constraint firstItem="rAF-gQ-Cfc" firstAttribute="top" secondItem="H1p-VJ-UJC" secondAttribute="top" id="fac-Zt-Cv0"/>  <constraint firstItem="DM7-ZN-r9N" firstAttribute="leading" secondItem="H1p-VJ-UJC" secondAttribute="leading" constant="16" id="gXQ-Xe-ZQF"/>  <constraint firstAttribute="trailing" secondItem="rAF-gQ-Cfc" secondAttribute="trailing" constant="16" id="yUc-oB-3am"/>  </constraints>  </tableViewCellContentView>  <viewLayoutGuide key="safeArea" id="HcO-TI-UQF"/>  <connections>  <outlet property="errorLabel" destination="rAF-gQ-Cfc" id="GlO-N3-ujG"/>  <outlet property="indexLabel" destination="DM7-ZN-r9N" id="2vf-IN-Wdp"/>  </connections>  </tableViewCell>  </prototypes>  <connections>  <outlet property="dataSource" destination="FLi-iA-KhU" id="jQT-Ua-sIN"/>  <outlet property="delegate" destination="FLi-iA-KhU" id="I2h-Lw-9BB"/>  </connections>  </tableView>  <navigationItem key="navigationItem" title="Errors" id="mAe-En-sGY">  <barButtonItem key="rightBarButtonItem" title="Close" style="plain" id="ikw-AE-ObO">  <connections>  <action selector="closePressed" destination="FLi-iA-KhU" id="h7b-Ep-uke"/>  </connections>  </barButtonItem>  </navigationItem>  </tableViewController>  <placeholder placeholderIdentifier="IBFirstResponder" id="ykz-z3-rhT" userLabel="First Responder" sceneMemberID="firstResponder"/>  </objects>  <point key="canvasLocation" x="1086" y="133"/>  </scene>  <!--Navigation Controller-->  <scene sceneID="39n-Ty-IGk">  <objects>  <navigationController automaticallyAdjustsScrollViewInsets="NO" id="8RR-ZM-3pp" sceneMemberID="viewController">  <toolbarItems/>  <navigationBar key="navigationBar" contentMode="scaleToFill" insetsLayoutMarginsFromSafeArea="NO" id="ttg-bS-63D">  <rect key="frame" x="0.0" y="0.0" width="540" height="44"/>  <autoresizingMask key="autoresizingMask"/>  </navigationBar>  <nil name="viewControllers"/>  <connections>  <segue destination="FLi-iA-KhU" kind="relationship" relationship="rootViewController" id="Fqx-ld-n7h"/>  </connections>  </navigationController>  <placeholder placeholderIdentifier="IBFirstResponder" id="knT-6r-MdK" userLabel="First Responder" sceneMemberID="firstResponder"/>  </objects>  <point key="canvasLocation" x="268" y="133"/>  </scene>  <!--Set values for inputs-->  <scene sceneID="RfT-a6-CjN">  <objects>  <tableViewController id="C1O-zF-TvR" customClass="InputsTableViewController" customModule="Pseudo\_Code" customModuleProvider="target" sceneMemberID="viewController">  <tableView key="view" clipsSubviews="YES" contentMode="scaleToFill" alwaysBounceVertical="YES" dataMode="prototypes" style="plain" separatorStyle="none" rowHeight="-1" estimatedRowHeight="-1" sectionHeaderHeight="28" sectionFooterHeight="28" id="izC-M9-JuR">  <rect key="frame" x="0.0" y="0.0" width="540" height="620"/>  <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>  <color key="backgroundColor" white="1" alpha="1" colorSpace="custom" customColorSpace="genericGamma22GrayColorSpace"/>  <prototypes>  <tableViewCell contentMode="scaleToFill" selectionStyle="default" indentationWidth="10" reuseIdentifier="InputVariableCell" rowHeight="60" id="GRo-7X-1rN" customClass="ProgramInputTableViewCell" customModule="Pseudo\_Code" customModuleProvider="target">  <rect key="frame" x="0.0" y="28" width="540" height="60"/>  <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>  <tableViewCellContentView key="contentView" opaque="NO" clipsSubviews="YES" multipleTouchEnabled="YES" contentMode="center" tableViewCell="GRo-7X-1rN" id="LqL-9Q-Lc9">  <rect key="frame" x="0.0" y="0.0" width="540" height="60"/>  <autoresizingMask key="autoresizingMask"/>  <subviews>  <label opaque="NO" userInteractionEnabled="NO" contentMode="left" horizontalHuggingPriority="251" verticalHuggingPriority="251" text="1" textAlignment="center" lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="ibw-jo-b6B">  <rect key="frame" x="8" y="8" width="43.5" height="43.5"/>  <color key="backgroundColor" red="0.74117647060000003" green="0.062745098040000002" blue="0.87843137250000003" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>  <constraints>  <constraint firstAttribute="width" secondItem="ibw-jo-b6B" secondAttribute="height" multiplier="1:1" id="oDM-BY-Kmi"/>  </constraints>  <fontDescription key="fontDescription" name="HelveticaNeue-Bold" family="Helvetica Neue" pointSize="25"/>  <color key="textColor" white="1" alpha="1" colorSpace="custom" customColorSpace="genericGamma22GrayColorSpace"/>  <nil key="highlightedColor"/>  </label>  <view contentMode="scaleToFill" translatesAutoresizingMaskIntoConstraints="NO" id="Ki0-IW-Mom">  <rect key="frame" x="59.5" y="8" width="472.5" height="43.5"/>  <subviews>  <textField opaque="NO" contentMode="scaleToFill" contentHorizontalAlignment="left" contentVerticalAlignment="center" placeholder="Type input here..." textAlignment="natural" minimumFontSize="17" translatesAutoresizingMaskIntoConstraints="NO" id="Q2S-hX-URX">  <rect key="frame" x="8" y="0.0" width="456.5" height="43"/>  <color key="backgroundColor" white="0.0" alpha="0.0" colorSpace="custom" customColorSpace="genericGamma22GrayColorSpace"/>  <nil key="textColor"/>  <fontDescription key="fontDescription" type="system" pointSize="14"/>  <textInputTraits key="textInputTraits" autocorrectionType="no" spellCheckingType="no" smartDashesType="no" smartInsertDeleteType="no" smartQuotesType="no"/>  </textField>  </subviews>  <color key="backgroundColor" white="0.94667245369999997" alpha="1" colorSpace="custom" customColorSpace="genericGamma22GrayColorSpace"/>  <constraints>  <constraint firstItem="Q2S-hX-URX" firstAttribute="leading" secondItem="Ki0-IW-Mom" secondAttribute="leading" constant="8" id="FEu-LH-oIi"/>  <constraint firstItem="Q2S-hX-URX" firstAttribute="top" secondItem="Ki0-IW-Mom" secondAttribute="top" id="GPS-Qh-odB"/>  <constraint firstAttribute="bottom" secondItem="Q2S-hX-URX" secondAttribute="bottom" id="fHQ-kI-7VX"/>  <constraint firstAttribute="trailing" secondItem="Q2S-hX-URX" secondAttribute="trailing" constant="8" id="vPu-Ec-ACg"/>  </constraints>  </view>  </subviews>  <constraints>  <constraint firstAttribute="bottom" secondItem="Ki0-IW-Mom" secondAttribute="bottom" constant="8" id="5dJ-jb-0rm"/>  <constraint firstItem="ibw-jo-b6B" firstAttribute="top" secondItem="LqL-9Q-Lc9" secondAttribute="top" constant="8" id="8r6-bs-hcO"/>  <constraint firstItem="ibw-jo-b6B" firstAttribute="leading" secondItem="LqL-9Q-Lc9" secondAttribute="leading" constant="8" id="BAZ-Pe-u3k"/>  <constraint firstAttribute="trailing" secondItem="Ki0-IW-Mom" secondAttribute="trailing" constant="8" id="C7G-oz-Aob"/>  <constraint firstAttribute="bottom" secondItem="ibw-jo-b6B" secondAttribute="bottom" constant="8" id="Imm-DS-IEc"/>  <constraint firstItem="Ki0-IW-Mom" firstAttribute="leading" secondItem="ibw-jo-b6B" secondAttribute="trailing" constant="8" id="fog-7o-JwY"/>  <constraint firstItem="Ki0-IW-Mom" firstAttribute="top" secondItem="LqL-9Q-Lc9" secondAttribute="top" constant="8" id="mUE-eV-ekq"/>  </constraints>  </tableViewCellContentView>  <viewLayoutGuide key="safeArea" id="zgP-XU-ugY"/>  <connections>  <outlet property="field" destination="Q2S-hX-URX" id="y3u-DL-jnb"/>  <outlet property="fieldContainer" destination="Ki0-IW-Mom" id="vEb-p8-NI7"/>  <outlet property="numberLabel" destination="ibw-jo-b6B" id="PEJ-xn-d1g"/>  </connections>  </tableViewCell>  </prototypes>  <connections>  <outlet property="dataSource" destination="C1O-zF-TvR" id="zQz-3V-bX1"/>  <outlet property="delegate" destination="C1O-zF-TvR" id="heX-6d-Rp6"/>  </connections>  </tableView>  <navigationItem key="navigationItem" title="Set values for inputs" id="7fb-a3-Zy7">  <barButtonItem key="leftBarButtonItem" systemItem="cancel" id="M9p-sb-I68">  <connections>  <action selector="cancelPressed" destination="C1O-zF-TvR" id="FkA-yC-8wL"/>  </connections>  </barButtonItem>  <barButtonItem key="rightBarButtonItem" title="Done" style="done" id="t2H-19-UkH">  <connections>  <action selector="donePressed" destination="C1O-zF-TvR" id="haF-uF-zvq"/>  </connections>  </barButtonItem>  </navigationItem>  </tableViewController>  <placeholder placeholderIdentifier="IBFirstResponder" id="Qu5-KU-JUa" userLabel="First Responder" sceneMemberID="firstResponder"/>  </objects>  <point key="canvasLocation" x="1086" y="884"/>  </scene>  <!--Navigation Controller-->  <scene sceneID="bcp-AT-DR1">  <objects>  <navigationController automaticallyAdjustsScrollViewInsets="NO" id="1al-l5-IB9" sceneMemberID="viewController">  <toolbarItems/>  <navigationBar key="navigationBar" contentMode="scaleToFill" insetsLayoutMarginsFromSafeArea="NO" id="a2K-GY-2Ud">  <rect key="frame" x="0.0" y="0.0" width="540" height="44"/>  <autoresizingMask key="autoresizingMask"/>  </navigationBar>  <nil name="viewControllers"/>  <connections>  <segue destination="C1O-zF-TvR" kind="relationship" relationship="rootViewController" id="gbW-bu-PX2"/>  </connections>  </navigationController>  <placeholder placeholderIdentifier="IBFirstResponder" id="e7n-vv-o5z" userLabel="First Responder" sceneMemberID="firstResponder"/>  </objects>  <point key="canvasLocation" x="268" y="884"/>  </scene>  <!--Code Library-->  <scene sceneID="2Ir-3g-2wV">  <objects>  <tableViewController id="FUo-Y7-J9u" customClass="LibraryTableViewController" customModule="Pseudo\_Code" customModuleProvider="target" sceneMemberID="viewController">  <tableView key="view" clipsSubviews="YES" contentMode="scaleToFill" alwaysBounceVertical="YES" dataMode="prototypes" style="grouped" separatorStyle="default" rowHeight="-1" estimatedRowHeight="-1" sectionHeaderHeight="18" sectionFooterHeight="18" id="GQg-iA-1Kv">  <rect key="frame" x="0.0" y="0.0" width="540" height="620"/>  <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>  <color key="backgroundColor" cocoaTouchSystemColor="groupTableViewBackgroundColor"/>  <prototypes>  <tableViewCell clipsSubviews="YES" contentMode="scaleToFill" preservesSuperviewLayoutMargins="YES" selectionStyle="default" accessoryType="disclosureIndicator" indentationWidth="10" reuseIdentifier="LibraryCell" textLabel="KGs-ig-aes" style="IBUITableViewCellStyleDefault" id="jWT-zw-tqy">  <rect key="frame" x="0.0" y="55.5" width="540" height="44"/>  <autoresizingMask key="autoresizingMask"/>  <tableViewCellContentView key="contentView" opaque="NO" clipsSubviews="YES" multipleTouchEnabled="YES" contentMode="center" preservesSuperviewLayoutMargins="YES" insetsLayoutMarginsFromSafeArea="NO" tableViewCell="jWT-zw-tqy" id="R43-od-hc1">  <rect key="frame" x="0.0" y="0.0" width="502" height="43.5"/>  <autoresizingMask key="autoresizingMask"/>  <subviews>  <label opaque="NO" multipleTouchEnabled="YES" contentMode="left" insetsLayoutMarginsFromSafeArea="NO" text="Title" textAlignment="natural" lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO" id="KGs-ig-aes">  <rect key="frame" x="20" y="0.0" width="482" height="43.5"/>  <autoresizingMask key="autoresizingMask"/>  <fontDescription key="fontDescription" type="system" pointSize="17"/>  <nil key="textColor"/>  <nil key="highlightedColor"/>  </label>  </subviews>  </tableViewCellContentView>  </tableViewCell>  </prototypes>  <connections>  <outlet property="dataSource" destination="FUo-Y7-J9u" id="8el-cl-ePP"/>  <outlet property="delegate" destination="FUo-Y7-J9u" id="sTt-6K-7xw"/>  </connections>  </tableView>  <navigationItem key="navigationItem" title="Code Library" id="rqd-s8-39H">  <barButtonItem key="rightBarButtonItem" title="Close" id="Y3r-W6-xOp">  <connections>  <action selector="closePressed" destination="FUo-Y7-J9u" id="mdh-ad-XSB"/>  </connections>  </barButtonItem>  </navigationItem>  </tableViewController>  <placeholder placeholderIdentifier="IBFirstResponder" id="0bl-ZW-UtA" userLabel="First Responder" sceneMemberID="firstResponder"/>  </objects>  <point key="canvasLocation" x="1086" y="1677"/>  </scene>  <!--Navigation Controller-->  <scene sceneID="GwW-Qq-xv2">  <objects>  <navigationController automaticallyAdjustsScrollViewInsets="NO" id="4ta-gw-4dp" sceneMemberID="viewController">  <toolbarItems/>  <navigationBar key="navigationBar" contentMode="scaleToFill" insetsLayoutMarginsFromSafeArea="NO" id="i01-1N-MDi">  <rect key="frame" x="0.0" y="0.0" width="540" height="44"/>  <autoresizingMask key="autoresizingMask"/>  </navigationBar>  <nil name="viewControllers"/>  <connections>  <segue destination="FUo-Y7-J9u" kind="relationship" relationship="rootViewController" id="FUD-gT-NbG"/>  </connections>  </navigationController>  <placeholder placeholderIdentifier="IBFirstResponder" id="h4T-V7-hHu" userLabel="First Responder" sceneMemberID="firstResponder"/>  </objects>  <point key="canvasLocation" x="267.7734375" y="1676.5625"/>  </scene>  </scenes>  <resources>  <image name="fast\_forward" width="24" height="15"/>  <image name="play" width="20" height="23"/>  </resources>  </document> |

|  |
| --- |
| **InputCodeCollectionViewCell.xib** |
| <?xml version="1.0" encoding="UTF-8"?>  <document type="com.apple.InterfaceBuilder3.CocoaTouch.XIB" version="3.0" toolsVersion="13771" targetRuntime="iOS.CocoaTouch" propertyAccessControl="none" useAutolayout="YES" useTraitCollections="YES" useSafeAreas="YES" colorMatched="YES">  <device id="retina4\_7" orientation="portrait">  <adaptation id="fullscreen"/>  </device>  <dependencies>  <deployment identifier="iOS"/>  <plugIn identifier="com.apple.InterfaceBuilder.IBCocoaTouchPlugin" version="13772"/>  <capability name="Safe area layout guides" minToolsVersion="9.0"/>  <capability name="documents saved in the Xcode 8 format" minToolsVersion="8.0"/>  </dependencies>  <customFonts key="customFonts">  <array key="Hack-Bold.otf">  <string>Hack-Bold</string>  </array>  </customFonts>  <objects>  <placeholder placeholderIdentifier="IBFilesOwner" id="-1" userLabel="File's Owner"/>  <placeholder placeholderIdentifier="IBFirstResponder" id="-2" customClass="UIResponder"/>  <collectionViewCell opaque="NO" clipsSubviews="YES" multipleTouchEnabled="YES" contentMode="center" id="KRf-Si-lUF" customClass="InputCodeCollectionViewCell" customModule="Pseudo\_Code" customModuleProvider="target">  <rect key="frame" x="0.0" y="0.0" width="266" height="49"/>  <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>  <view key="contentView" opaque="NO" clipsSubviews="YES" multipleTouchEnabled="YES" contentMode="center" insetsLayoutMarginsFromSafeArea="NO">  <rect key="frame" x="0.0" y="0.0" width="266" height="49"/>  <autoresizingMask key="autoresizingMask"/>  <subviews>  <view contentMode="scaleToFill" translatesAutoresizingMaskIntoConstraints="NO" id="CnF-Xf-haT">  <rect key="frame" x="0.0" y="0.0" width="266" height="49"/>  <color key="backgroundColor" white="1" alpha="1" colorSpace="custom" customColorSpace="genericGamma22GrayColorSpace"/>  </view>  <label opaque="NO" userInteractionEnabled="NO" contentMode="left" horizontalHuggingPriority="251" verticalHuggingPriority="251" text="KEYWORD" textAlignment="center" lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="zuw-Jk-Z8h">  <rect key="frame" x="4" y="0.0" width="258" height="49"/>  <fontDescription key="fontDescription" name="Hack-Bold" family="Hack" pointSize="19"/>  <nil key="textColor"/>  <nil key="highlightedColor"/>  </label>  </subviews>  </view>  <color key="backgroundColor" white="1" alpha="1" colorSpace="custom" customColorSpace="genericGamma22GrayColorSpace"/>  <constraints>  <constraint firstAttribute="bottom" secondItem="CnF-Xf-haT" secondAttribute="bottom" id="1Yc-N3-szL"/>  <constraint firstItem="CnF-Xf-haT" firstAttribute="top" secondItem="KRf-Si-lUF" secondAttribute="top" id="2nj-6a-0XI"/>  <constraint firstAttribute="trailing" secondItem="CnF-Xf-haT" secondAttribute="trailing" id="Ohk-P1-QGB"/>  <constraint firstAttribute="trailing" secondItem="zuw-Jk-Z8h" secondAttribute="trailing" constant="4" id="l2V-Yj-Ulb"/>  <constraint firstItem="zuw-Jk-Z8h" firstAttribute="leading" secondItem="KRf-Si-lUF" secondAttribute="leading" constant="4" id="pkC-FM-N66"/>  <constraint firstItem="zuw-Jk-Z8h" firstAttribute="top" secondItem="KRf-Si-lUF" secondAttribute="top" id="puo-Co-Bou"/>  <constraint firstItem="CnF-Xf-haT" firstAttribute="leading" secondItem="KRf-Si-lUF" secondAttribute="leading" id="rDJ-ey-EkD"/>  <constraint firstAttribute="bottom" secondItem="zuw-Jk-Z8h" secondAttribute="bottom" id="sfc-P1-JWx"/>  </constraints>  <viewLayoutGuide key="safeArea" id="ksK-ha-cIv"/>  <size key="customSize" width="266" height="49"/>  <connections>  <outlet property="container" destination="CnF-Xf-haT" id="nG9-pW-GlT"/>  <outlet property="label" destination="zuw-Jk-Z8h" id="8ns-Ow-yxZ"/>  </connections>  <point key="canvasLocation" x="135" y="-65.5"/>  </collectionViewCell>  </objects>  </document> |

|  |
| --- |
| **InputCodeKeyboardAccessory.xib** |
| <?xml version="1.0" encoding="UTF-8"?>  <document type="com.apple.InterfaceBuilder3.CocoaTouch.XIB" version="3.0" toolsVersion="13771" targetRuntime="iOS.CocoaTouch" propertyAccessControl="none" useAutolayout="YES" useTraitCollections="YES" useSafeAreas="YES" colorMatched="YES">  <device id="retina4\_7" orientation="portrait">  <adaptation id="fullscreen"/>  </device>  <dependencies>  <deployment identifier="iOS"/>  <plugIn identifier="com.apple.InterfaceBuilder.IBCocoaTouchPlugin" version="13772"/>  <capability name="Safe area layout guides" minToolsVersion="9.0"/>  <capability name="documents saved in the Xcode 8 format" minToolsVersion="8.0"/>  </dependencies>  <objects>  <placeholder placeholderIdentifier="IBFilesOwner" id="-1" userLabel="File's Owner"/>  <placeholder placeholderIdentifier="IBFirstResponder" id="-2" customClass="UIResponder"/>  <view contentMode="scaleToFill" id="OAh-Yg-ZSB" customClass="InputCodeKeyboardAccessory" customModule="Pseudo\_Code" customModuleProvider="target">  <rect key="frame" x="0.0" y="0.0" width="375" height="50"/>  <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>  <subviews>  <collectionView clipsSubviews="YES" multipleTouchEnabled="YES" contentMode="scaleToFill" showsHorizontalScrollIndicator="NO" showsVerticalScrollIndicator="NO" dataMode="prototypes" translatesAutoresizingMaskIntoConstraints="NO" id="LRg-Jg-ph9">  <rect key="frame" x="0.0" y="0.0" width="375" height="50"/>  <color key="backgroundColor" white="0.0" alpha="0.0" colorSpace="custom" customColorSpace="genericGamma22GrayColorSpace"/>  <collectionViewFlowLayout key="collectionViewLayout" scrollDirection="horizontal" minimumLineSpacing="10" minimumInteritemSpacing="10" id="fxL-iS-Lsj">  <size key="itemSize" width="50" height="50"/>  <size key="headerReferenceSize" width="0.0" height="0.0"/>  <size key="footerReferenceSize" width="0.0" height="0.0"/>  <inset key="sectionInset" minX="0.0" minY="0.0" maxX="0.0" maxY="0.0"/>  </collectionViewFlowLayout>  <cells/>  </collectionView>  </subviews>  <color key="backgroundColor" red="0.92124956849999995" green="0.92194873089999996" blue="0.92135781049999999" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>  <constraints>  <constraint firstItem="LRg-Jg-ph9" firstAttribute="top" secondItem="OAh-Yg-ZSB" secondAttribute="top" id="0zt-tA-pgk"/>  <constraint firstItem="LRg-Jg-ph9" firstAttribute="leading" secondItem="6Y6-ud-dhL" secondAttribute="leading" id="3k0-6Y-zlq"/>  <constraint firstAttribute="bottom" secondItem="LRg-Jg-ph9" secondAttribute="bottom" id="y4k-sF-4Ts"/>  <constraint firstAttribute="trailing" secondItem="LRg-Jg-ph9" secondAttribute="trailing" id="yh0-aY-0PL"/>  </constraints>  <freeformSimulatedSizeMetrics key="simulatedDestinationMetrics"/>  <viewLayoutGuide key="safeArea" id="6Y6-ud-dhL"/>  <connections>  <outlet property="collectionView" destination="LRg-Jg-ph9" id="Tel-5A-gOv"/>  </connections>  </view>  </objects>  </document> |

Testing

For my project to be successful, pseudo code programs need to run successfully. During my testing, I will test every single feature that was included in my objectives using normal, boundary and erroneous data as appropriate.

# Testing Strategy

According to my objectives, here are the tests of the application interface which I believe should be carried out to ensure quality and that my application works:

|  |  |
| --- | --- |
| The user should be able to input a program successfully. | **PASS** |
| The user should be able to see the output of the program successfully. | **PASS** |
| The buttons on top of the keyboard should allow the user to input pseudo code keywords into the input area successfully. | **PASS** |
| The app should clearly show where errors are, what the errors are and how to fix that error. | **PASS** |
| The app should be able to clear the code currently in the editor. | **PASS** |
| The app should be able to save and load code snippets from the library. | **PASS** |
| The app should be able to take the user to documentation for the syntax of the pseudo code. | **PASS** |

The above tests will be to do with the application and the user’s interaction with it. In my opinion, those tests were more suitable to show in a video. The link for the video is below, along with times where each of the tests are performed.

# Application Testing

|  |  |
| --- | --- |
| [**https://youtu.be/XsqWH-gLiDI**](https://youtu.be/XsqWH-gLiDI) | |
| **0.00** | Here, I input the sample program that was provided to me by my client in the analysis stage of research for my project.  The user can easily input code using the onscreen keyboard and the additional row of buttons including keywords and operators for pseudo code that I have included. |
| **1.41** | Now, I will select check which will validate the syntax of the code. Since the syntax of the code was valid, the app will now ask me to provide inputs for the code. In line 1 of this program, I wrote the line of code INPUT HowMany which is what the 1st input on the input pop up screen refers to. |
| **1.47** | I have written valid pseudo code and successfully provided the inputs for my program. The pop up dismisses and the console indicates that the program is ready to run. |
| **1.50** | I click the green Run button and the program is executed. The output has one line: “Goodbye” and a message says “Completed execution.”  Since my input was 5, the value in the Total variable was 1+2+3+4 = 10.  Since 10 is not less than 10, the else statement was reached and “Goodbye” was printed to the console. |
| **1.55** | I am saving the code that I have just written to my library. I click the Library button and a Code Library pop up appears.  I click Save current code and specify a name for my code snippet.  My code snippet then successfully appears alongside my other code snippets as “Test video”. |
| **2.09** | I then make use of the Clear Code button which will clear all code from the editor.  I then reopen the code library and select the code snippet I have just saved. This shows that the code was successfully saved and that I can now reopen it. |
| **2.21** | Here, I am creating an error in my code. In line 1, I am changing the keyword INPUT to lowercase - input (which is not valid syntax). |
| **2.30** | I click Check again and the line which I have changed is highlighted indicating there is 1 error on that line.  Clicking the 1 symbol will show the error screen. It shows a specific error message for the syntax error that I have demonstrated.  Clicking on that error message will take me to a specific webpage to tell me more information about my mistake. *For now, it will take me to the CIE Pseudo code documentation for their examinations. This could easily be changed to any other web page specifically for each type of error.* |
| **2.53** | Clicking on the question mark in the lower left hand corner will open up a guide to the syntax of the pseudo code used in this application. This is a webpage for the CIE Pseudo code documentation for their examinations. This could easily be changed to any other web page. |
| **3.03** | Here, I am inputting a new simple program with a number of inputs, a mathematical expression into an assignment, and a number of outputs. |
| **3.35** | I click the Check button and it is valid, so I am asked to enter the inputs. |
| **3.45** | Instead of running the code this time, I step through the code line by line. The line which is about to run is highlighted in purple. After stepping over each of the OUTPUT statements, the output is added to the Console output. |

This video shows the application running and the interaction with the user. It shows that each of the tests that were mentioned above **have passed.**

# Pseudo Code Testing

Also according to my objectives, below are some of the tests that will test the working of the interpreter in my application. For each test, I will provide:

1. An explanation of the test being carried out.
2. The inputted pseudo code into the program.
3. The equivalent of the pseudo code in C#.
4. The output by my application of the pseudo code.
5. The output by Visual Studio of the C# program.
6. A description of the result of the test.

By providing both pseudo code and C# for each test carried out, I can ensure that each test was absolutely successful with more confidence.

The tests I am going to carry out will demonstrate all of the features of the program and the latter tests will demonstrate many of these features working together in more complex algorithms which would have more useful applications.

Here are the tests of each of the features I will perform and the results of each of the tests:

|  |  |
| --- | --- |
| 1. A test to show INPUT, assignment and OUTPUT. | **PASS** |
| 1. A test to show mathematical expressions and results. | **PASS** |
| 1. A test to show string expressions and results, including string concatenation. | **PASS** |
| 1. A test to show various Boolean expressions and results. | **PASS** |
| 1. A test to show an if and else statement. | **PASS** |
| 1. A test to show a for next loop. | **PASS** |
| 1. A test to show a repeat until loop. | **PASS** |
| 1. A test to show a while loop. | **PASS** |
| 1. A test to show the NaN error message. | **PASS** |

For some more complex tests to perform, I used the example program that my client had given me which is included in the Analysis document and a past paper question from a CIE GCSE Computer Science Examination:

|  |  |
| --- | --- |
| 1. Example program given to me by my teacher during the analysis. | **PASS** |
| 1. CIE GCSE Computer Science 2015 Specimen Paper – Question 2 | **PASS** |
| 1. Add and average numbers inside a for loop. | **PASS** |

## INPUT, Assignment and OUTPUT Test

This test will demonstrate the ability of a user to input a value from the user, assign values to different variables and output values to the console. For this test only, I will show each screenshots of each stage of the test, afterwards I will only show the result.

|  |  |
| --- | --- |
| ../../../Simulator%20Screen%20Shot%20-%20iPad%20Air%202%20-%202018-03-25%20at%2018.42.27.png | ../../../Simulator%20Screen%20Shot%20-%20iPad%20Air%202%20-%202018-03-25%20at%2018.42.35.png |
| ../../../Simulator%20Screen%20Shot%20-%20iPad%20Air%202%20-%202018-03-25%20at%2018.42.48.png | I have inputted the program into the panel on the left. I have clicked CHECK to validate the program. The popup then appears showing the form to input variables where I inputted the values 20 and 30 for A and B. I then clicked DONE and then I clicked RUN.  The output appeared in the console on the right. |

|  |  |  |
| --- | --- | --- |
|  | Pseudocode | C# |
| CODE | ../../../Screen%20Shot%202018-03-25%20at%2019.20.54.png | ../../../Screen%20Shot%202018-03-25%20at%2019.20.12.png |
| Console OUTPUT | ../../../Screen%20Shot%202018-03-25%20at%2019.20.57.png | ../../../Screen%20Shot%202018-03-25%20at%2019.09.02.png  *(The 20 and 30 on lines 1 and 2 are the inputs into the console in this screenshot)* |

A was equal to 20 and B was equal to 30, as inputted. C was correctly equal to the sum of A and B – 50. All outputs were correct. **Therefore, this test passed.**

## Mathematical Expressions Test

This test will check complex mathematical expressions in my application. The expressions should follow the rules/order of BIDMAS and should give the correct result.

|  |  |
| --- | --- |
| Pseudocode | C# |
| ../../../Screen%20Shot%202018-03-25%20at%2019.31.03.png | ../../../Screen%20Shot%202018-03-25%20at%2019.30.52.png |
| ../../../Screen%20Shot%202018-03-25%20at%2019.32.54.png | ../../../Screen%20Shot%202018-03-25%20at%2019.30.59.png |

BIDMAS order is brackets, indices, division, multiplication, addition and then subtraction. The code to find the result of A solves a complex mathematical expression. The result of A (6) is correct;

B then shows the use of the MOD function which is symbolised by the % operator. The remained when dividing 6 by 4 is 2. Therefore, the result of B (2) is correct.

The output for the pseudocode and the C# match, therefore **this test passed.**

## String Expressions Test

This test will demonstrate the ability to add strings together in the application, both from stored variables and using string literals. In the example below, we will create a greeting using a name stored in a name variable which was inputted from the user and a string literal that was coded into the program.

|  |  |
| --- | --- |
| Pseudocode | C# |
| ../../../Screen%20Shot%202018-03-25%20at%2019.43.03.png | ../../../Screen%20Shot%202018-03-25%20at%2019.42.59.png |
| ../../../Screen%20Shot%202018-03-25%20at%2019.43.05.png | ../../../Screen%20Shot%202018-03-25%20at%2019.42.55.png  *(The ‘Jake’ on line 1 was the input to the Console)* |

In this example, the string “Jake” was inputted into the application for the input in both examples.

The output to the console was the expected value (“Hello, Jake”) in both examples. The output was the same in both examples. Therefore, **this test passed.**

## Boolean Expressions Test

This test will demonstrate each of the Boolean operators available in my application: less/greater than, less/greater than or equal to, equal to, not equal to.

|  |  |
| --- | --- |
| Pseudocode | C# |
| ../../../Screen%20Shot%202018-03-25%20at%2019.51.25.png | ../../../Screen%20Shot%202018-03-25%20at%2019.52.53.png |
| ../../../Screen%20Shot%202018-03-25%20at%2019.50.06.png | ../../../Screen%20Shot%202018-03-25%20at%2019.52.45.png |

This code shows the ability to compare values either stored in a value or expressed explicitly in the code. It also demonstrates the ability to store a Boolean value in a variable. It also demonstrates the ability to print a Boolean value to the Console.

Each of the Boolean expressions outputted the correct value, and the pseudocode output was matching the C# output. Therefore, **this test passed.**

## If And Else Statement Test

This test will show the ability to conditionally perform one block of code or another based on a Boolean value passed into the if statement.

|  |  |
| --- | --- |
| Pseudocode | C# |
| ../../../Screen%20Shot%202018-03-25%20at%2020.04.20.png | ../../../Screen%20Shot%202018-03-25%20at%2020.04.34.png |
| ../../../Screen%20Shot%202018-03-25%20at%2020.04.24.png | ../../../Screen%20Shot%202018-03-25%20at%2020.04.47.png |

This test shows the evaluation of a Boolean expression within an if statement (including mathematical expressions). It will calculate if A times 5 (50) is equal to B times 5 (25), which is false. Since it is false, only the else block should be run.

The expected result occurred, where only the else block ran. The output from my application matched the output from the C# console, therefore **this test passed.**

## For Next Loop Test

This test will show the ability to perform a block of code a certain number of times, as specified within a for next loop structure.

|  |  |
| --- | --- |
| Pseudocode | C# |
| ../../../Screen%20Shot%202018-03-25%20at%2020.08.05.png | ../../../Screen%20Shot%202018-03-25%20at%2020.09.36.png |
| ../../../Screen%20Shot%202018-03-25%20at%2020.08.08.png | ../../../Screen%20Shot%202018-03-25%20at%2020.09.25.png  *(The 10 on line 1 was the input to the Console)* |

This program demonstrates the ability to perform a for loop a number of times (could be an explicitly stated number, a number for a variable or a number calculated in the for loop). It also demonstrates the ability to use the variable within only the scope of the for loop of the index of the iteration the for loop is currently on.

This loop was expected to print every number from 1 to 10 inclusive, which was correctly outputted to the Console. The C# console output matched the output of my application, therefore **this test passed.**

## Repeat Until Loop Test

This test will demonstrate the ability to perform a repeat until loop, where the code is run and then the Boolean expression is evaluated as to whether the loop should run again.

|  |  |
| --- | --- |
| Pseudocode | C# |
| ../../../Screen%20Shot%202018-03-25%20at%2020.19.08.png | ../../../Screen%20Shot%202018-03-25%20at%2020.19.23.png |
| ../../../Screen%20Shot%202018-03-25%20at%2020.19.37.png | ../../../Screen%20Shot%202018-03-25%20at%2020.17.49.png  *(The 5 in Line 1 is the input to the Console)* |

In C#, there is no REPEAT UNTIL structure. The closest thing available is a DO WHILE structure, where the Boolean expressions needs to be the opposite of a REPEAT UNTIL structure. In the pseudocode, we run until A >= 5 but in the C# we run while A < 5.

This test demonstrates the ability to perform a REPEAT UNTIL loop and increment the value of a variable within a loop.

The pseudocode test outputted the expected result, and both languages returned the same output. Therefore, **this test passed.**

## While Loop Test

This test demonstrates the ability to perform a block of code conditionally in a WHILE, END WHILE loop.

|  |  |
| --- | --- |
| Pseudocode | C# |
| ../../../Screen%20Shot%202018-03-25%20at%2020.24.19.png | ../../../Screen%20Shot%202018-03-25%20at%2020.24.14.png |
| ../../../Screen%20Shot%202018-03-25%20at%2020.23.39.png | ../../../Screen%20Shot%202018-03-25%20at%2020.24.56.png |

This test demonstrated a WHILE, END WHILE loop with the expected result and both languages produced the same output. Therefore, **this test passed.**

## NaN Error Message Test

|  |  |
| --- | --- |
| Pseudocode | C# |
| ../../../Screen%20Shot%202018-03-26%20at%2022.57.20.png | ../../../Screen%20Shot%202018-03-26%20at%2022.54.51.png |
| ../../../Screen%20Shot%202018-03-26%20at%2022.57.24.png | *../../../Screen%20Shot%202018-03-26%20at%2022.54.59.png*  *The C# program crashed and the following error message was shown on Line 3 in the editor.* |

Returning this error message was the expected result, therefore **this test passed.**

## Example Program from Analysis Document

This was the program that my client gave me during the research stage of this project. He said that he needed the application to be able to run programs of this level of complexity.

|  |  |
| --- | --- |
| Pseudocode | C# |
| ../../../Screen%20Shot%202018-03-25%20at%2020.31.17.png | ../../../Screen%20Shot%202018-03-25%20at%2020.33.27.png |
| ../../../Screen%20Shot%202018-03-25%20at%2020.31.22.png | ../../../Screen%20Shot%202018-03-25%20at%2020.33.45.png  *(The 5 on Line 1 was the input to the Console).* |

This test demonstrates the use of inputs, assignment, mathematical expressions, for loops, if statements, else statements, Boolean expressions and output to the console. It demonstrates nearly all of the features I have included in my program.

The value of Total after the for loop will be 0+1+2+3+4 = 10, which is not less than 10, therefore the Goodbye is printed from the else block. The pseudocode output and the C# output are the same, therefore **this test passed.**

## Add and Average Numbers in a Loop

This is another program which uses many of the features of my program which I created. It is similar in complexity to the previous example and demonstrates an algorithm which is very common.

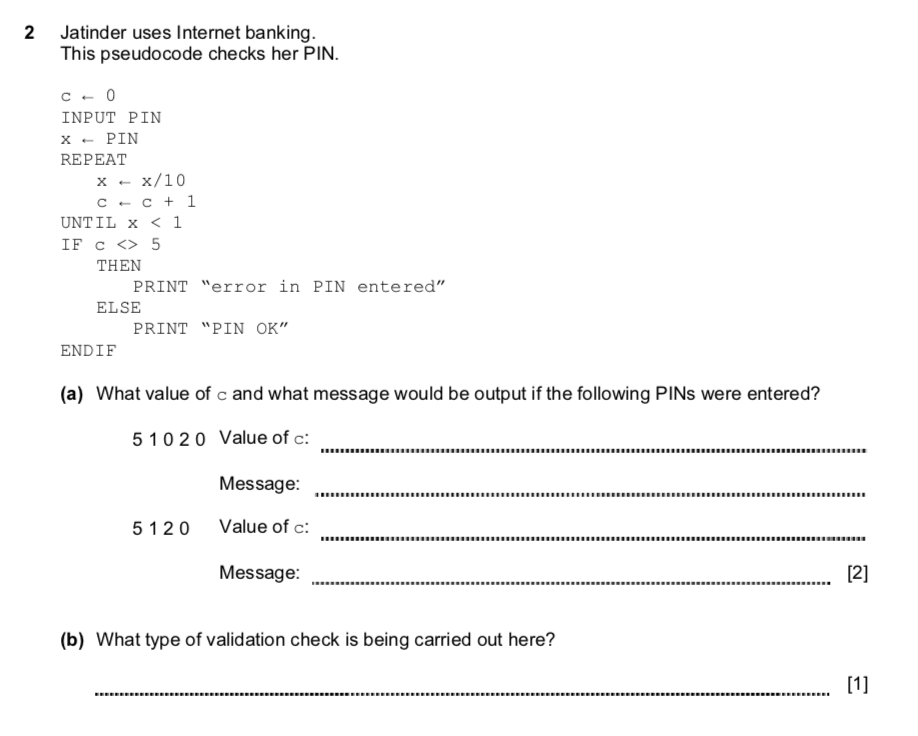
|  |  |
| --- | --- |
| Pseudocode | C# |
| ../../../Screen%20Shot%202018-03-25%20at%2020.37.56.png | ../../../Screen%20Shot%202018-03-25%20at%2020.39.18.png |
| ../../../Screen%20Shot%202018-03-25%20at%2020.37.59.png | ../../../Screen%20Shot%202018-03-25%20at%2020.39.26.png  *(The 5 on Line 1 is the user input to the Console)* |

This test demonstrated INPUT, assignment, for loops, mathematical expressions and output to the console in an algorithm which is very common.

It produced the expected result with 5 being the input for both programs. The total resulted in 15 and the average resulted in 3. The pseudocode output matched the C# output, therefore **this test passed.**

## 2015 Specimen Paper Question 2 Test

The following is a question from a past paper for Computer Science GCSE. I will run the pseudocode using my program with a number of different inputs and try to answer the questions after the exercise.



Below is the code inputted into Pseudocode and C#:

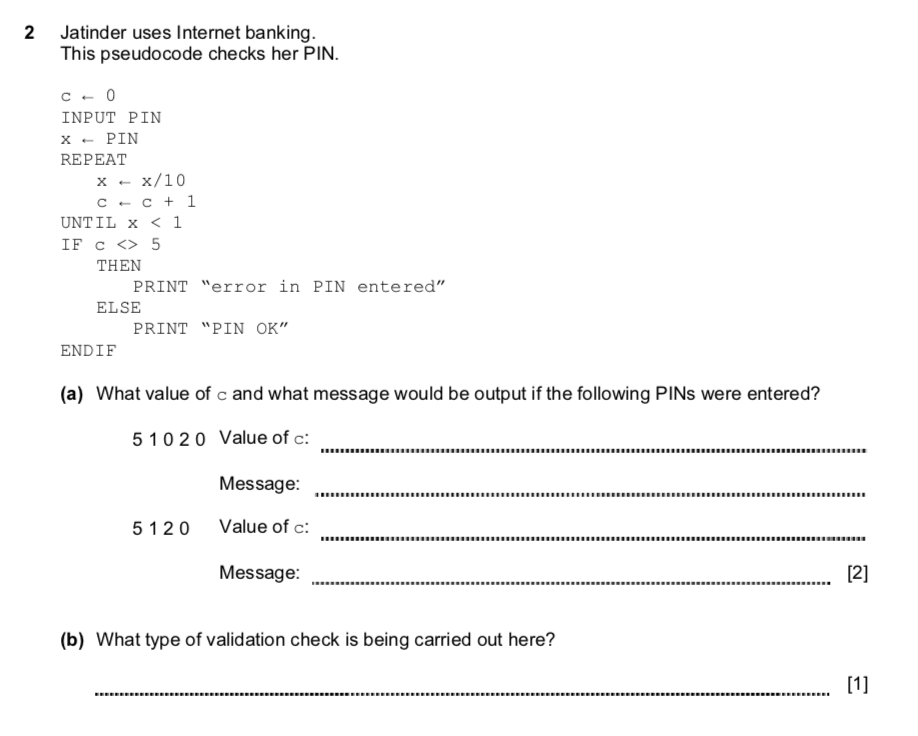
|  |  |
| --- | --- |
| Pseudocode | C# |
| ../../../Screen%20Shot%202018-03-25%20at%2020.46.13.png | ../../../Screen%20Shot%202018-03-25%20at%2020.46.57.png |

Now I will test the program for both languages with a range of inputs. *(The first line in the output of each C# test is the input from the user to the Console).*

|  |  |  |
| --- | --- | --- |
| Input | Pseudocode output | C# output |
| 0 | ../../../Screen%20Shot%202018-03-25%20at%2020.49.51.png | ../../../Screen%20Shot%202018-03-25%20at%2020.50.56.png |
| 51020 | ../../../Screen%20Shot%202018-03-25%20at%2020.50.25.png | ../../../Screen%20Shot%202018-03-25%20at%2020.54.17.png |
| 5102 | ../../../Screen%20Shot%202018-03-25%20at%2020.49.51.png | ../../../Screen%20Shot%202018-03-25%20at%2020.54.40.png |
| 1238761287 | ../../../Screen%20Shot%202018-03-25%20at%2020.49.51.png | ../../../Screen%20Shot%202018-03-25%20at%2020.51.50.png |

Both languages performed the same outputs, which means **my test passed** for functionality of the application.

Now, let’s look at the questions following the psedocode in the paper:



Using the inputs that I have already performed, we can answer part (a) straight away.

For **51020**, the **value of c is 5** and the message is **“PIN OK”.**

For **5120**, the **value of c is 4** and the message is **“error in PIN entered”.**

From using my application on a number of inputs, we can answer part (b) as:

**A check to see if the length of the PIN entered is 5 digits long.**

# Error Testing

The application allows for the checking of syntax errors within the program.

Due to a lack of time, I was only able to include a few error messages into the app. However, I did add a place in the code where more checks could easily be added. Adding specific error messages is specific to each type of error and would be very time intensive. I decided to focus my time on other aspects of the application.

If a check is not supported however and there is a syntax error, the app will provide the message “Syntax error”.

Below are tests to show each of the syntax error messages supported:

## Unsupported syntax error message

In this test, the keyword PRINT is not supported. The correct keyword for printing to the Console is OUTPUT. Since there is no check for this error, the message “Syntax error” will be shown.

|  |  |
| --- | --- |
| Input | ../../../Screen%20Shot%202018-03-25%20at%2021.00.53.png |
| Error Message | ../../../Screen%20Shot%202018-03-25%20at%2021.00.59.png |

## Unterminated String Literal error message

One of the most common errors when writing pseudocode according to my client was incorrect usage of apostrophes when defining strings.

This test for syntax error will check whether there is a string that has been opened with an apostrophe, but never closed.

|  |  |
| --- | --- |
| Input | ../../../Screen%20Shot%202018-03-25%20at%2021.03.23.png |
| Error Message | ../../../Screen%20Shot%202018-03-25%20at%2021.03.31.png |

## Incorrect case of keywords error message

Another of the most common errors when writing pseudo code is writing a keyword in the wrong case.

This test for syntax error will check if a line of code is valid, except for a keyword which is not in the correct case (all keywords should be CAPITALS).

|  |  |
| --- | --- |
| Input | ../../../Screen%20Shot%202018-03-25%20at%2021.06.36.png |
| Error Message | ../../../Screen%20Shot%202018-03-25%20at%2021.06.31.png |

Here, the correct keyword should be “OUTPUT” rather than “output”. Therefore, the message tells the user that one of the keywords is not the correct case.

For more specific cases, I have provided the functionality to go to a specific web page for more information about that error. However, due a lack of time, this web page will direct the user to the CIE documentation about pseudo code syntax for their examinations.

Evaluation

My client’s original statement was this:

What I’m looking for is a teaching aid for the Divisions and Fifth form. For their GCSE, they have to master Pseudo code which allows them to master a programming language. I would like something that allows them to experiment with pseudo code, something that helps them to understand how it works and something that helps me to mark it. Those are the three main advantages.

He wanted this application to:

1. Help students experiment with pseudo code.
2. Help students understand how code works.
3. Help teachers mark pseudo code.

Through my demonstrations, I feel that I have successfully reached these three criteria. During my testing, I showed a range of features of pseudo code that are possible with the application which allows students to experiment. I have shown the endless possibilities of code that can be performed, which helps students to understand. And I have shown with a real past paper question that teachers/students can understand and mark pseudo code exercises better than previous methods.

# Objectives

Below are my objectives that I defined in the analysis section of this document:

1. **The product should be in the form of an iPad application where a user can run a pseudo code program:**
   1. The app should have an input panel where the user can input a program.
   2. The app should have an output panel, displaying the output of the program typed into the input panel.
2. **The user should have clear controls on how to input the program:**
   1. There should be buttons alongside the keyboard to input keywords into the program.
   2. There should be provided instructions on the correct syntax for the pseudo code being used.
3. **The user should be informed of any errors in the program:**
   1. The app should highlight exactly which line is incorrect.
   2. The app should provide reason(s) why the line is incorrect.
   3. The app should show ways to correct the code in detail.
4. **The program should use and enforce specific into inputted code:**
   1. Enforce the use of the “<-“ arrow for assignment.
   2. INPUT, OUTPUT commands must be supported.
   3. IF, ELSE, END IF conditionals must be supported.
   4. Not equals to <> command must be supported.
   5. FOR NEXT, REPEAT UNTIL loops must be supported.
   6. WHILE and END WHILE loops should be supported.
   7. Conversion to INT must be supported.
   8. Arithmetic operators must be supported, including the MOD command.
5. **The program should be simple, clear and easy to use:**
   1. This app is aimed at GCSE students, aged 14 to 16, it should be designed in a friendly way.
   2. The app should be clearly laid out, with very obvious instructions and prompts of how to use.
   3. The app should follow iPad and iOS conventions.

I will go through each of these objectives and discuss how successfully they were completed.

## The product should be in the form of an iPad application where a user can run a pseudo code program

|  |  |
| --- | --- |
| **The app should have an input panel where the user can input a program.** | |
| There is an input panel which is easy to input code into. | ✓ | |
| **The app should have an output panel, displaying the output of the program typed into the input panel.** | |
| There is an output panel which provides information about whether a program is valid or not, whether the code is ready to run, when the code has completed execution and most importantly, the outputs of the program. | ✓ | |

## The user should have clear controls on how to input the program

|  |  |
| --- | --- |
| **There should be buttons alongside the keyboard to input keywords into the program.** | |
| There are buttons on top of the keyboard to easily input keywords into the program. The following options are available:  <-, INPUT, OUTPUT, IF, THEN, ELSE, END IF, FOR, TO, NEXT, WHILE, END WHILE, REPEAT, UNTIL, TRUE, FALSE | ✓ |
| **There should be provided instructions on the correct syntax for the pseudo code being used.** | |
| The question mark button in the lower left hand corner directs the user to a web page with information about the correct syntax. This can easily be changed to any other web page. | ✓ |

## The user should be informed of any errors in the program

|  |  |
| --- | --- |
| **The app should highlight exactly which line is incorrect.** | |
| When a line has invalid syntax, the program will highlight it red and display a number on the right hand side with the number of syntax errors identified in that line. | ✓ |
| **The app should provide reason(s) why the line is incorrect.** | |
| Due to a lack of time, my app currently only supports two specific types of error:   1. Unterminated string literal 2. Incorrect case of keywords   Otherwise, the message “syntax error” will be shown. However, I have provided an easy way in my code for additional error messages to be added in the future.  The task of identifying error messages is very specific to each type of error and requires a specific check on the String line of code to identify it. I felt that this task was very time intensive and my time was better spent on improving other parts of the program.  Therefore, I have marked this objective as yellow – **incomplete**. | ? |
| **The app should show ways to correct the code in detail.** | |
| For this task, I implemented a URL to be assigned to each type of error that was identified. The user can select the error and be directed to that URL in the iPad browser if he/she wants.  Since every error check that I included had a URL assigned to it if necessary, then I will mark this objective as **complete**. | ✓ |

## The program should use and enforce specific into inputted code

|  |  |  |
| --- | --- | --- |
| **Enforce the use of the “<-“ arrow for assignment.** | ✓ | |
| **INPUT, OUTPUT commands must be supported.** | ✓ | |
| **IF, ELSE, END IF conditionals must be supported.** | ✓ | |
| **Not equals to <> command must be supported.** | ✓ | |
| **FOR NEXT, REPEAT UNTIL loops must be supported.** | ✓ | |
| **WHILE and END WHILE loops should be supported.** | ✓ | |
| **Conversion to INT must be supported.** | |
| I unfortunately did not have time to complete this task.  Since in pseudo code, types are not specified like in C# or in Swift, my app had limited functionality in mixing types of variable. Trying to add a number and a String would fail but it would not know why. Therefore, this task required a bit more thinking and work to support an INT conversion. | X | |
| **Arithmetic operators must be supported, including the MOD command.** | |
| I did include the MOD command into my program. In usual pseudo code, the operator is defined as MOD whereas in my program I had to use %.  This was because of the way that the Shunting Yard algorithm worked because each digit/operator had to be 1 character long for it to work. I did add capability for numbers longer than 1 digit but did not have time to support the longer MOD operator. Therefore, I decided to use the % operator which is used in most modern programming languages instead.  The adjustment to the code to support the MOD command would not be too complex, but I felt my time was better spent elsewhere. | ✓ | |

## The program should be simple, clear and easy to use

|  |  |
| --- | --- |
| **This app is aimed at GCSE students, aged 14 to 16, it should be designed in a friendly way.** | |
| The app was designed with big, colourful buttons. It is interactive, easy to use and input code and simple to understand the output and the functionality of the app as a whole. | ✓ |
| **The app should be clearly laid out, with very obvious instructions and prompts of how to use.** | |
| The app had an incredibly simple layout, with the main panels, clear pop ups with instructions and obvious buttons.  However, there were no clear instructions about how to use the app. For someone who has absolutely no previous coding/computing experience, they may have trouble understanding how to use it. The app may have benefitted with an introductory tutorial page and better laid out documentation for the instructions of the pseudo code syntax. | ✓ |
| **The app should follow iPad and iOS conventions.** | |
| The app follows iPad and iOS conventions well and accords to Apple’s User Interface Guidelines completely. It responds well to touch and is simple, obvious and easy to use. Any user familiar with iOS and iPad apps should have no problem using this application. | ✓ |

# Evaluation from the Client

The final step in my evaluation was to show my client the application and see his reaction/feedback.

I initially went to my client with an almost finished version of the application. It still had a few bugs and was missing a few features which made the app difficult to use. Some of these features included the Clear Code button, the Code Library, the additional keyboard with pseudo code keywords and no error messages apart from “Syntax error”.

This was a letter he wrote to me after our first meeting:

|  |
| --- |
| *Dear Jacob,*    *Thank you very much for showing me your PseudoCode interpreter this morning and walking me through its abilities.  Overall I have to say I’m very impressed, this is something that I’ve wanted as a teaching tool for a very long time.  It does do exactly what I wanted, enforcing the use of ‘exam specification’ pseudocode, with up-to-date rules about the Keywords and syntax required.  It gives good, if limited feedback on mistakes and affords the satisfaction of seeing a well-composed algorithm run and produce output.*    *You have already identified some improvements that would make the program easier to use, such as a ‘Clear’ button in the editor to reset the environment.   I would also suggest that being able to save and load text files of pseudocode would be a ‘nice to have’ although I appreciate this wasn’t in the original specification or your objectives.*    *The handling of errors however is part of the objectives and here I think we agree it would be nice to see more detail as to what kind of error it is and, ideally, some suggestions as to how to fix it.   I suggested that a simple way of enhancing what is currently provided might be to include a browser that could be directed with an error code to a specific webpage.  Then the teacher – or even students, could write the help instructions, which is something we would enjoy doing, leaving you simply to provide as detailed error codes as possible.*    *There are still one or two keywords, for example WHILE END WHILE that have not been implemented due to pressure of time.  The app is perfectly usable and very useful without these, but it would obviously be good to implement these at a later stage.   Personally I also find the iPad keyboard hard to use, and would therefore appreciate the inclusion of shortcuts to common Keywords, but this is possibly a mixed blessing – the advantage of having no prompts or predictive text is that users have to think carefully about what they are doing.*    *Thinking carefully is the essence of pseudocode and you have created a tool which encourages that, while delivering a real reward for a successful algorithm.  That’s a real step forward and I thank you for the work that you have done.*    *Regards*  *Graham* |

After this feedback, I went back and made all the improvements that he had mentioned here and that we had discussed. After our final meeting with a completely finished product, this was a second letter that he sent to me:

|  |
| --- |
| *Dear Jacob,*    *Thanks for meeting with me again and walking me through the updates to your PseudoCode interpreter.  Once again I’m impressed, not just with the final product, it’s level of polish and functionality, but also with the way that you’ve taken my earlier feedback on board and made the changes that we discussed.   The ‘Clear’ button is in place as requested and you have obviously spent a good amount of time implementing the ability to save and load code examples.  These two features really add to the applications utility.*    *You’ve also added a browser and the ability to handle errors in a more ‘pupil friendly’ way, which is much appreciated.*    *As far as I can see you have achieved all of the objectives, including adding support for the keywords WHILE END WHILE.  The only thing that seems to be missing is support for the INT function and, as I explained, this is a very minor issue for me, since it’s only appeared in examined pseudocode once and there are ways of generating integer values without it.  You might like to consider, for your evaluation, what would be needed in order to finish this objective but the application is perfectly usable without it.*    *Once again, thank you for this application.  You’ve achieved something really substantial, written an interpreter and provided something that will actually get used and give benefit to your pupils.  Well Done.*    *Graham Macleod*  *Head of Academic Computing.* |

# Current Errors in Program

There are three major errors that I could not fix due to lack of time within my program currently.

1. If an INPUT statement exists within a loop, the ProgramInterpreter class will analyse the code and interpret this as one INPUT statement within the whole program, even though there will me many more. Therefore, the app will only ask for one input. As I mentioned earlier, if an input is requested past the provided inputs, the program will simply return negative 1. Therefore, if an input is in a for loop, it will provide the actual input in the first iteration and then -1 for all other iterations.   
     
   This highlights a major flaw in the way I designed the inputs to work. In most programs, the input is asked for while the code is running, whereas I ask for all inputs beforehand. This made it easier to interpret the code and to validate it but meant that flaws like this were left in the app.   
     
   This would cause a lot of work to fix since it would mean I would have to redesign the whole way that inputs work, the whole way that the ProgramInterpreter works and many other parts of the program.
2. The second error was not part of my original objectives but still could cause a lot of confusion to some students.  
     
   When stepping line by line through a program, as shown in the final section of my video, there is an error when stepping through loops. Since a loop instruction is considered as one instruction in my code, stepping over a loop will perform the whole block of code at once but the highlighted line will go to the first line in the loop code. The highlighted line will then not return to the start of the loop all iterations of the loop were already completed when run() was performed on that Instruction object.  
     
   This would not require lots of work to fix, as it just involves some more specific changing of the index inside the Program object of the currently highlighted line. Since this was no part of my original objectives, I did not invest any time into fixing it, but it will be very important to fix this issue in the future.
3. A third major error is the not a number error message. To save time, I return 0 when a number is attempted to divide by 0. This is incorrect and may be very confusing whilst teaching young computer science students.  
     
   Ideally, the program should have not been able to run/stopped running and an error message saying you cannot divide by 0 and an explanation for why this is the case should be shown. However, this involves lots of additional work in implementing as the outcomes of the program need to be decided ahead of time.  
     
   I felt this was not as important as other features of the program so I implemented this temporary fix.

# Improvements for the Future

There are a number of improvements that could be made to my program, but during my discussion with my client and from my own evaluation, these were the most important:

1. There is currently no support for arrays. This would be fairly easy to implement as an Array could be of the type [Any] which would suit me perfectly. This is very important in my opinion as many of the past paper questions that I looked at included arrays and meant that I could not include them during the testing stage of my product.
2. My application requires more error messages. This is something that is very important since one of the biggest reasons for the application was to improve the understanding of students. If the interpreter will say “Syntax error” for every single error, the student will get frustrated as they will not know what their error is.  
     
   I tried to make it as easy as possible to include more specific error messages into my program however as I mentioned, this is very time intensive and I felt only a few error messages were necessary to demonstrate the functionality that the program had to offer.
3. One feature which I find would represent real-world coding a lot better would be to change the entire program to ask for inputs as the code goes along. This is how the Console in C# works and would remove some of the major flaws that I mentioned in the previous section.   
     
   This however, would require a lot more work and was not possible in my time frame.
4. One nice feature about my program currently is the step-by-step running of the program. However, a key feature to understanding how programming works better would be to see what is inside the variables currently after each line in the program.   
     
   The example past paper question that I used in the testing section of this document highlights the need for a feature like this and during my evaluation, my client commented that it would be a very nice addition. This is particularly relevant in exam questions including trace tables.  
     
   This would not be hard to do as the memory of the program is stored in the Program object, however it would require a lot of work to the user interface which is quite time consuming.
5. Another nice feature would be flowcharts. As discussed in my analysis, flowcharts are key to developing an understanding an introduction to programming and algorithms. Codef2flow provided a very nice interface to generate a graphic representation of the inputted algorithm as the user typed and this could be very useful in some cases, but not in the way that I set my application out to be.  
     
   If my application could view their pseudo code as a flowchart, or even somehow input a flow chart and see the code generated from that, it would be very beneficial to students I think.  
     
   This would be a very difficult feature to implement as it requires a great deal of work to generate the custom graphics automatically and to develop a system to input graphics to generate code would be even more difficult.
6. A final feature that was mentioned as a passing idea in my interview with my client but never set as an objective would be translation. I was taught C# when learning GCSE and I am familiar with Swift and a number of other languages.  
     
   I think a really useful tool inside the application would be the ability to export the pseudo code that a student has written into another language that they are learning so that they compare and better understand the code.  
     
   This would not be difficult to do. Each object that represents a command in code contains all the necessary information to quite easily translate it to another language. However, it would be time consuming as I would have to manually specify the language syntax for each supported language and configure this for exporting the code.

Links

**RegEx**

<https://en.wikipedia.org/wiki/Regular_expression>

<https://regexr.com>

**Shunting Yard**

<https://en.wikipedia.org/wiki/Shunting-yard_algorithm>

<http://csis.pace.edu/~wolf/CS122/infix-postfix.htm>

<https://github.com/brettshollenberger/shunting-yard-algorithm>