****

**Faculty of Science and Technology**

**2019/2020**

**Level 4**

**Introduction To Programming**

**Assignment 2**

**Hangman Program**

**Analysis, Design, and Implementation**

**Report**

1. **Self-Assessment of Performance: 5%**

**Tutor : Andrew Watson**

|  |  |
| --- | --- |
| **Student ID** | **5220251** |

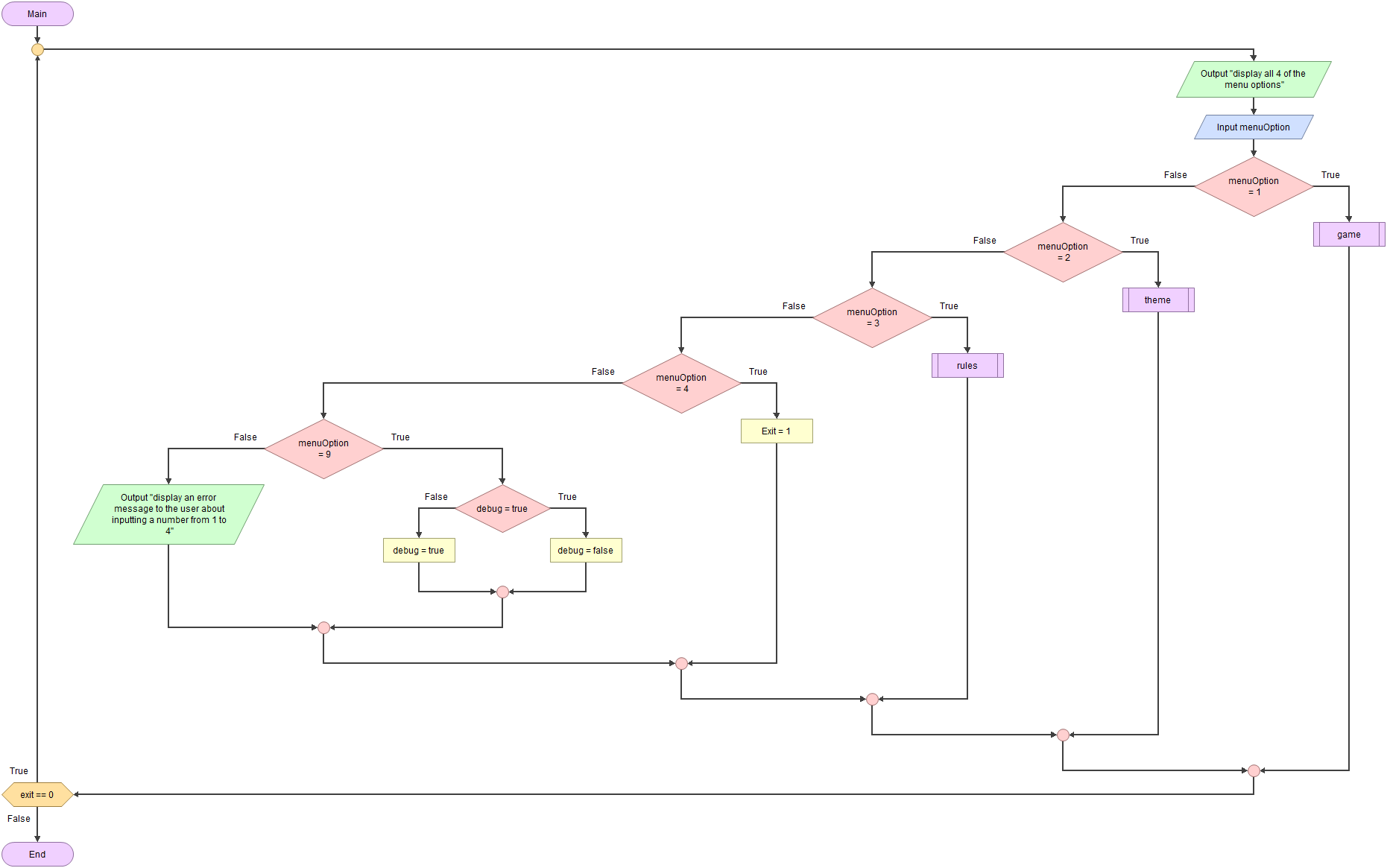
**Circle the appropriate response (has been highlighted red):**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Did I submit the assignment on time?** | **Yes** | **No** |  |  |
| **Did I complete the assignment?** | **Yes** | **No** |  |  |
| **If not, approx. how much did I complete?** | **%** |  |  |  |
| **How happy am I with what I submitted?** | **Very happy** | **Satisfied** | **Disappointed** | **Ashamed** |
| **What mark do I expect?** | **80%** |  |  |  |
| **Did I spend enough time on the assignment?** | **Yes** | **No** |  |  |
| **Did I get it proof-read by someone else?** | **Yes** | **No** |  |  |
| **Have I properly ‘referenced’ it?** | **Yes** | **No** |  |  |
| **Could I improve the presentation?** | **Yes** | **No** |  |  |

**Answer the following questions:**

|  |  |
| --- | --- |
| **The best part of my performance was:** | The use of error checking in my file especially in the guessing function, by using a do while loop and switch statement I was able to give specific error messages |
| **The worst part of my performance was:** | My efficiency, in some parts of my code things could have take up less lines, such as putting every available word in 1 array instead of 10, I should have starts earlier to give myself more time to fix this |
| **One way in which I could improve the content of my assignment is:** | Add more words to each category and perhaps also add a hard mode to the cryptic clues theme |
| **One way in which I could improve the presentation of my assignment is:** | Have a title on my main menu, although it is functional an gives the user the needed information it looks quite bland |
| **One thing I will do to improve my performance in my next assignment is:** | Start sooner and comment my code and splitting my code into different files from the beginning to help better manage my time near the end of the project |
| **Another thing I will do to improve my performance in my next assignment is:** | Start planning sooner, when I started coding, I didn’t have a proper flow diagram drawn out which caused me to make a lot of errors early on as I was doing everything on the fly instead of thinking things through |

**2. Analysis: 15%**



Shown in the diagram above is the most basic layout for my program and most of the modules that are assessable from the main menu module, I decided to start designing with the main menu in mind first as it is the first thing the user sees and every other module will be accessed through this menu, or through another module that is accessed via this menu. The four main options presented to the user upon starting the menu module will be to access the game module, change the theme/difficulty though the theme module, view the rules or quit the game. A fifth option is available to enter the debug mode, but this is not presented to the user immediately as it is only to be used for game testing and not general play.

Menu module:

Looking at the menu itself in the diagram the first thing that is noticeable is that almost everything is wrapped in a do while loop, every module needs to be accessed from the menu so it is very likely that this module will be run over and over. It needs to be run at least once in order to gather at least one user input however it is possible that the user will only want to run the program once as they might immediately choose to quit. Knowing all this I felt a do while loop was best. After outputting all the menu options, it would check for a user input. Depending on this input it will access and run certain modules and after each of these modules the menu module will resume, after every menu option (apart from 4) the menu loops as the exit condition has not been met allowing the user to easily choose between all the modules as soon as they have completed the last one. By having the validation check be at the end of the loop rather than at the beginning it means if the user does press 4 and decides to quit the game, the appropriate variable can be changed just before the loop ends so it can quit instantly instead of having to repeat the loop which is what would happen if the flagging happened at the beginning or somewhere in the middle.

This module only takes one input and the only major outputs are the choices it presents to the user upon starting the loop and an error message that is displayed if the user’s input is invalid. All other inputs and outputs are handled by other modules and the only processes are checking the users input against the various menu options. I will cover every module included in the menu section apart from the main game module as it contains many other modules and has its own diagram which I will get to later.

Theme module:

The theme module will allow the user to select from various themes and difficulties, to keep things simple both for the user and from a design perspective there will be a number of themes each with a hard version, all themes and their more difficult alternative will be listed one after another one a single screen and by having all themes and difficulties on a single screen rather than a separate screen or module for each it makes the choice easier to make as they can see everything and once and also makes it less complicated as it means they only have to navigate though one screen. One the surface it is a simple module as it simply outputs a list of all the available options and takes the input of the user’s choice to be used to select a them when the game module runs but it is slightly more complex than that. Similar to the main menu module the entire module will loop but instead of being based on an exit condition being flagged it will only exit the loop if the users input is valid, if it is invalid, the input will be flagged and the loop will repeat instead of exiting. By having the module loop over and over it means I can display error messages to the user if the module has looped before so they know what they are doing wrong but also by keeping the users input inside the module until it is confirmed to be valid it prevents invalid responses going into separate modules and possibly causing a crash in the software. Additionally, this module essentially is only gathering a single input so when this input is gathered and is determined to be correct the module has fulfilled its purpose and can be ended. The only complex part about this module is that the input will have to be accessed outside of the theme module, after the theme module ends the user will be returned to the menu module where it will immediately end and loop, if the users input is accessible in the menu module and the user decides to start the game, the input can be used in the game module.

Rules module:

This is by far the simplest module; it clears the screen so each section of the rules can be displayed, and it requires no input apart from a single key press to end the module and return to the menu. In the output I will cover four sections:

Basic rules- For users who have never played hangman before it will be important to cover how to play, as I designer I should assume the user knows nothing and I need to give all information necessary in order to allow the user to operation my program.

Score- I will cover the scoring system in more detail in my analysis but it is important to inform the user how it works so they know how they should be trying to play the game in order to score a good amount of points.

Hints- In the game module I will have a hint be automatically displayed to the user when they have 2 lives remaining in order to help them if they are struggling, this needs to be conveyed to the user as well as the exception case that is the cryptic clues category which is presented more like a riddle where the hint is displayed from the beginning, as long is this is presented clearly it should prevent user confusion about when and why hints or clues are appearing.

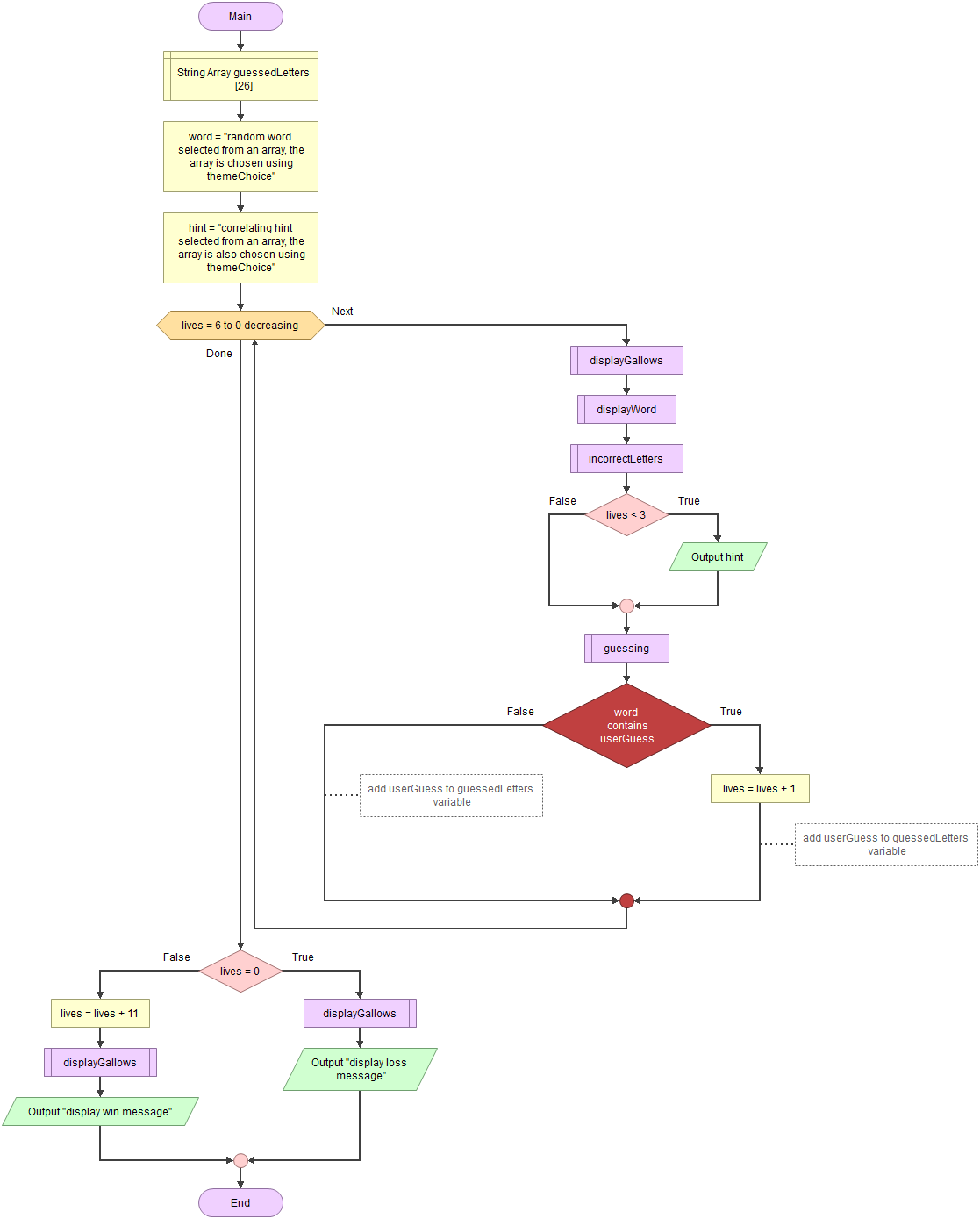
Debug mode- Although this is not intended to be used by most general users, for bug testing purposes it is important that the method of accessing debug mode and what it does is displayed to the user somewhere. I decided this should be included in the rules section as firstly, it is not to be used nearly as commonly as every other option that is mentioned on the main menu and additionally, unlike the other menu options like ‘play’ or ‘rules’, ‘debug mode’ is not nearly as obvious as to what it does and is best only turned on after they user fully understands what it does to prevent confusion and potentially ruining the users experience.

Exit:

Although this could be its own module, I decided to keep in within the main menu module, seeing as the main menu module will be the first module to run, and when it quits the entire program will close I decided to keep the design compact, by keeping it inside the menu I could make it a single line of code flipping a 0 to a 1 and instantly ending the program. If I had designed the program differently or I wanted to add a confirmation to quitting it might warrant its own module but in its current state when using the loop displayed above in my main module it is simply more efficient to not separate it.

Debug:

This is the hidden menu option that can only be accessed after reading the rules or by having prior knowledge. Similarly, to ‘exit’ I could have made this a separate module, but I decided not to. Although it is around 10 lines and could be put into its own module, I decided to keep it in the menu as the debug bool that is changed when taking the menu input needs to be accessed by the game module. If it was in another module it would have to be accessed through the menu so it could be input into the game module so I decided as it was only 10 lines it would be easier to have it already in the menu so it can much more easily be input into the game module.



Shown in the diagram above is the basic outline for the game module that is accessed when pressing one on the main menu. Although less modules can be accessed by the user, the module itself has access to more modules than the main menu as this is where the bulk of the logic occurs but most of it is out of the user’s hands. Despite this I would say this is arguably more important than the main menu as most if that is merely set up for what is the main game which is entirely contained within this module. The four modules that are accessed from within this module are; ‘Gallows’ which displays the graphic for said hangman, the score and the hint if the user has lost 4 or more lives, ‘Displaying word’ which displays the word in its current state with correctly guessed letters and blanks in their appropriate positions, ‘Incorrect letters’ which displays all letters that have already been guessed that were not in the word and ‘Guessing’ which allows the user to enter a character and checks if it is actually in the word.

Game module:

Apart from assigning the correct word and hint to the appropriate variables and defining an array in order to keep a record of users, the middle section of the module where most of the game is played is contained within a for loop. The user must have 6 lives, and these are the standard rules of hangman so by using a loop that will automatically end after 6 tries makes sense. When the loop runs, the gallows are displayed, the word is displayed, the incorrect letters are displayed and finally after all the information is presented to the user in order the user can make a guess. Although it is included in the guessing module so I may cover it later it is worth mentioning that if the user guesses a letter that is in the word their life count is increased by one, the two down sides of using a for loop in this way is that a life is always taken off so when we don’t want the user to lose a life we have to retroactively fix it and exiting the loop can be a little messy. If the user loses the game, then the user life counter will drop down from 1 to 0 and the loop will break but if the user wins, they will have more than 0 lives and the loop will not want to break. The solution to this is to reduce the amount of lives the user has by 10 if the user wins. This way the loop breaks at which point the user has 0 or less lives, if the user has 0 lives we know they have lost as their count will have dropped by a count of 1 from 1 to 0, but if they have any other number it means they have that has been dropped by more than 1 which can only happen if that user wins. Therefore if lives equals zero we can display an updated version of the gallows and present a failure message and if not, we add the lives we took away previously (plus an extra as 1 more life is taken off by the for loop) and display an accurate version of the gallows before presenting a victory message. After either of these scenarios the program waits for any key press before ending the module and returning to the menu. In actuality this module is surprisingly simple as it requires no input and only outputs a victory or failure message, but it houses many other modules that require many inputs and display many outputs.

Gallows module:

The gallows module takes no input and is almost entirely used to display graphics and information to the user, the first thing it must display is the graphic of the gallows and how much of the man to display depending on how many lives the user has lost. First it can take how many lives the user has lost and using that to print one line of the gallows with the man for every life lost, for every life the user hasn’t yet lost a line of empty gallows is printed. Six lines are printed every time to ensure that the gallows stay the same shape the amount of lives the user has left merely determines how much of the gallows is empty and how much is filled with the titular hanged man. The next thing the module does is calculate and print the score, print the amount of lives left and the hint (if the user has 2 or less lives).

Displaying word module:

Again, this module takes no input, but this is because it is actually called within the gallows, but this was difficult to show on the flowchart program I was using so I just displayed it after for ease of understanding. Every letter that has been guessed correctly is compared to every letter that is in the actual word and every letter that is in both is printed, if there are any letters that are in the word that have not been correctly guessed a blank is printed. Although the 10 lives are taken away when the user wins in the main game module the game finds out if the user has one in this module. At the start of the module a Boolean which tracks if the user has won is set to true assuming the user has won by default but if a blank is every printed the program the knows that the word has not been fully guessed and knows the user has not yet won and the Boolean is set to false, if every letter has been guessed and there are no blanks that means the user has won and the Boolean is not changed. The module ends, it returns to the game module which then takes away 10 lives which leads to the victory condition of the game as discussed above.

Incorrect letters module:

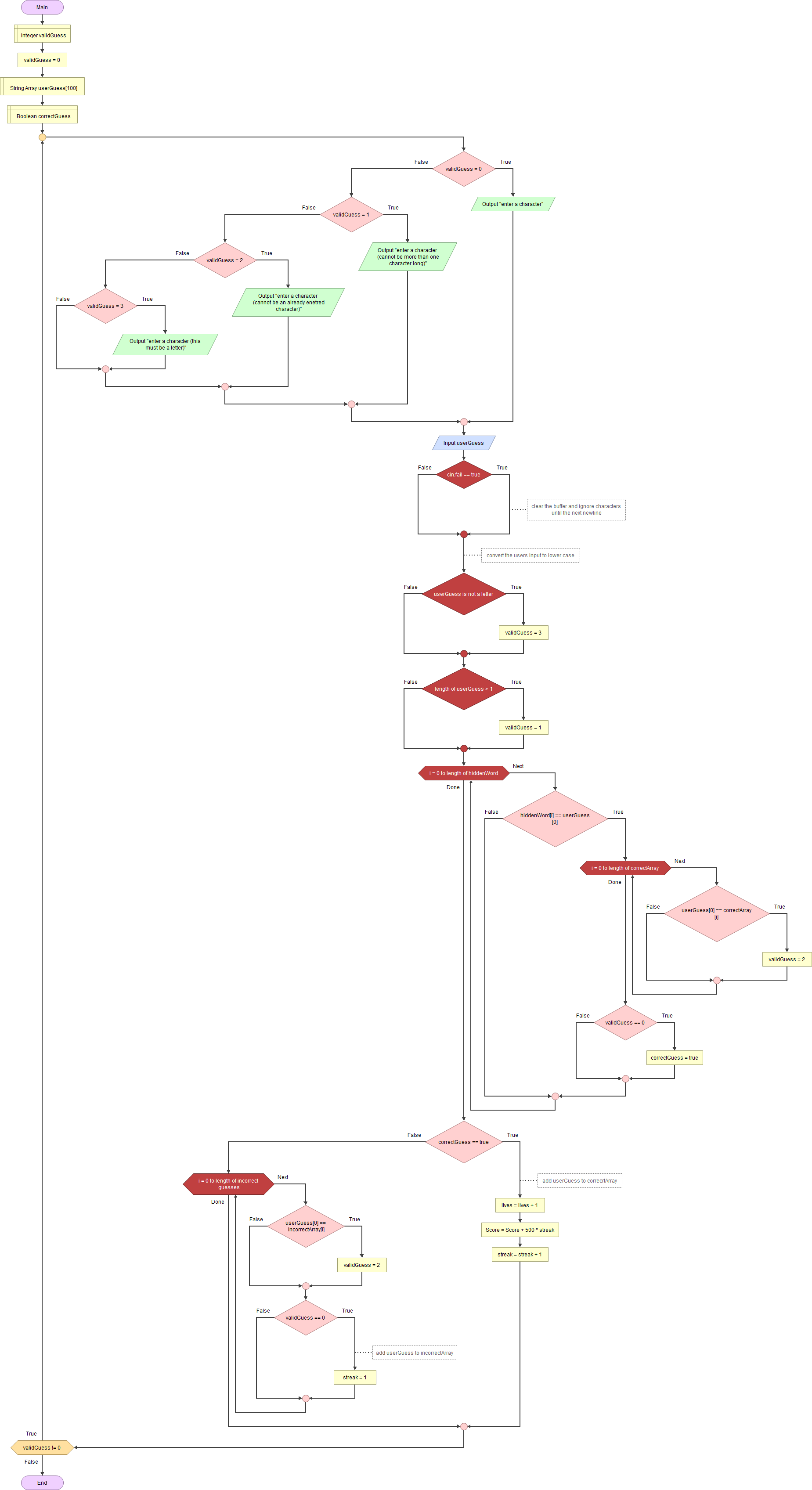
Functionally this module is very simple, it takes no input and runs a for loop that prints out every letter that has been guessed incorrectly and prints a space between them. These guesses are kept track of in the Guessing module which is accessed through the game module. It requires no input and outputs an array that has been calculated in a separate module but it is vital to display this information to this user so they don’t have to keep track of what they have already input and stopping the user from putting the same guess multiple times stopping them from playing the game properly.

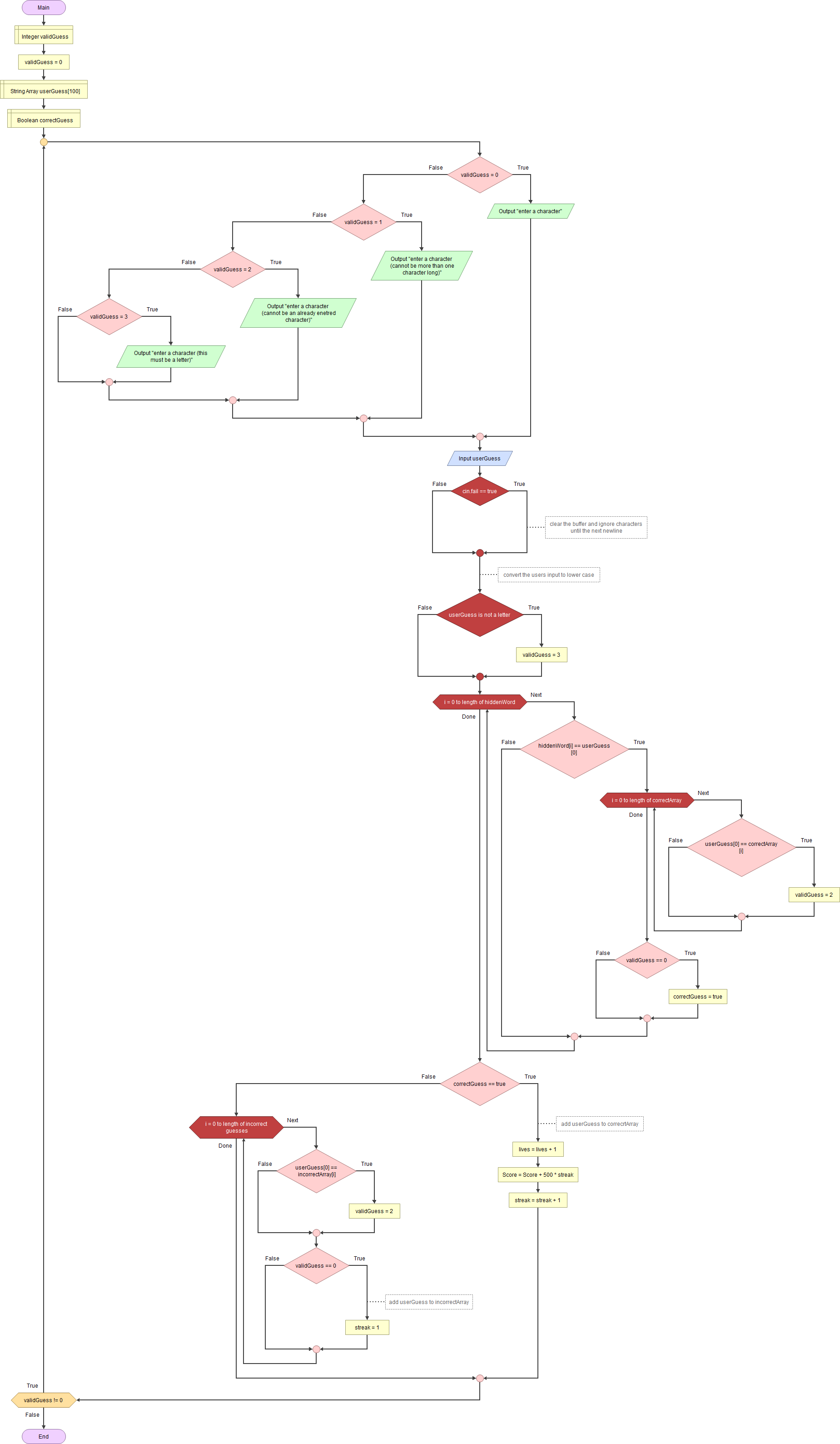
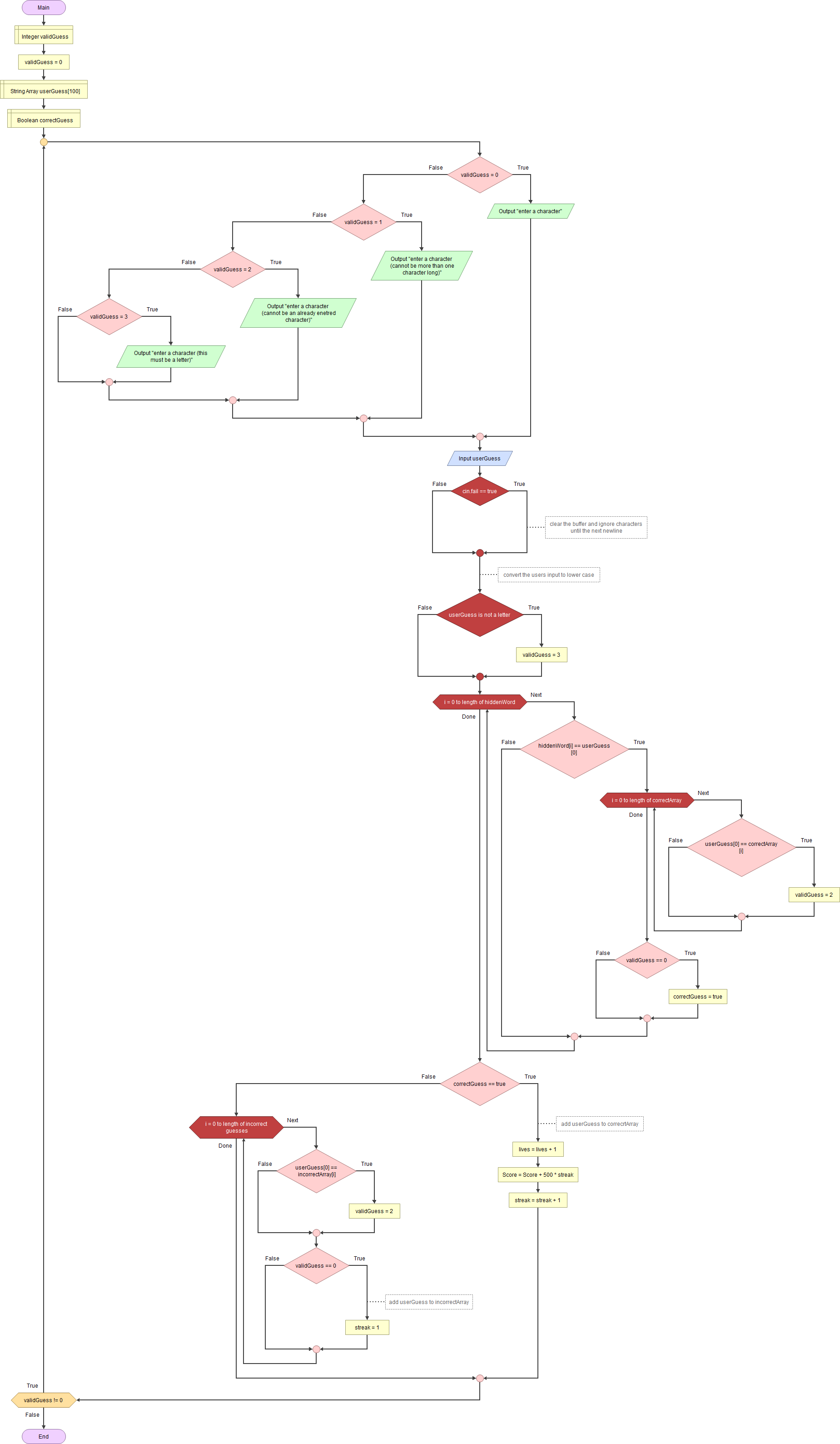
Guessing module:

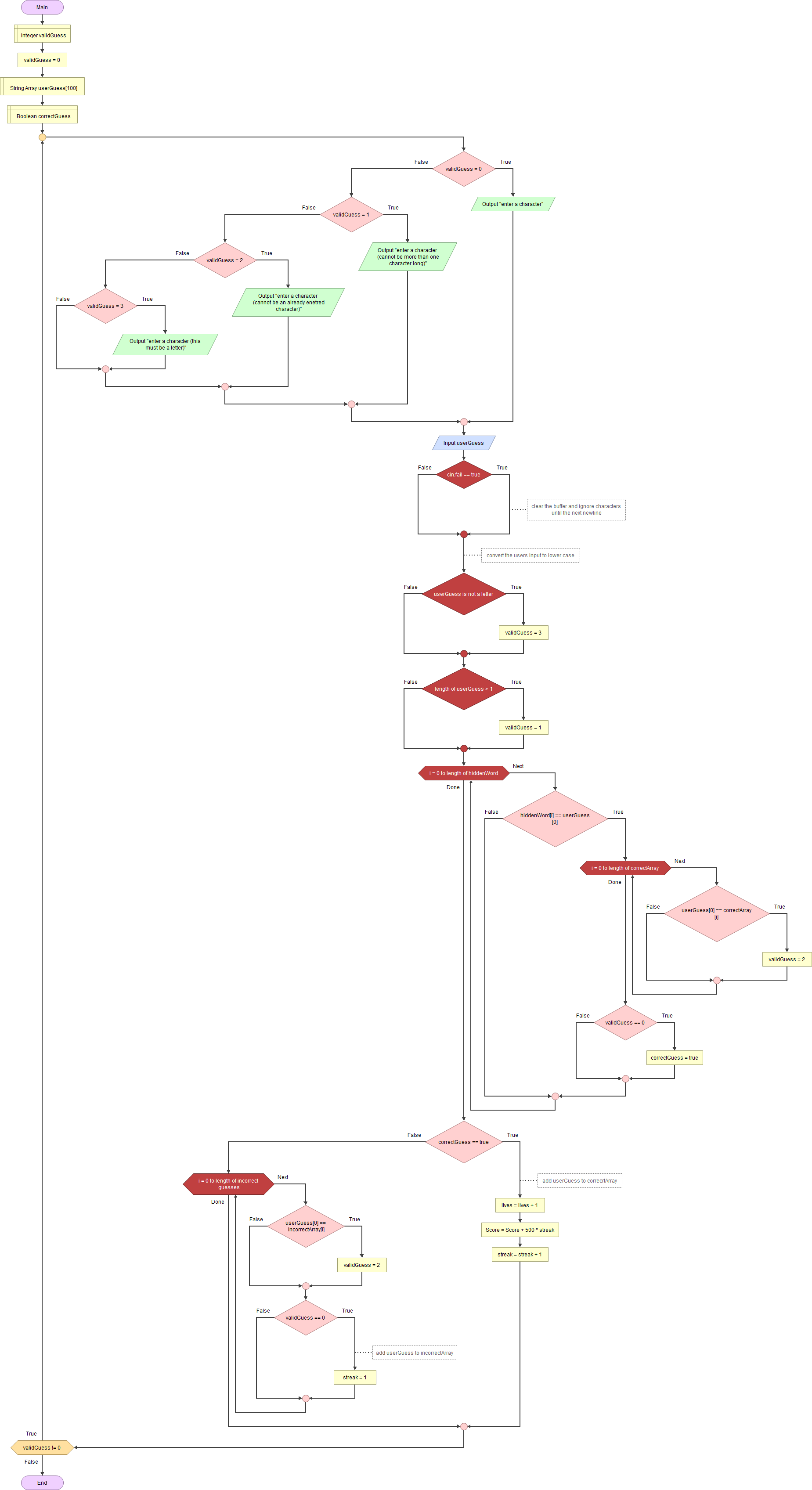
The only part of the game module that actually takes the users input and calculates it, in a similar way to the main menu and the theme module most of the module is wrapped in a for loop that only breaks if the users input is not flagged as invalid. What is different about this loop is that is has multiple distinct error codes. When choosing a single digit from 1 to 10 for a theme there is only one error code needed to tell the user to input an appropriate number. When guessing for hangman the user could enter something that isn’t a letter, something that is more than one character long, or something that has already been guessed. All these scenarios have their own unique error message to help the user be informed about what they have done and how to enter a proper input next time. The input is flagged in a different way for each thing that it might have done wrong and this enables the user to be informed about how to properly about how to play and use the program which is always useful. Apart from this the module runs all the validation checks on the users input as well as checking if it is a correct guess by checking against every letter in the word. If it is a correct guess it is added to the correct guesses array to that it will be displayed and can no longer be guessed, as well as this a life is restored and the appropriate amount of score is given to, they user. When the loop finally ends it returns to the game module so if the game has not yet ended, the entire process can start again from gallows to guessing until the game is done.

**3. Design: 15%**

Where the previous section talked about what the program was going to do in rather broad strokes this section will walk thought the character testing module (which in my case is the guessing module) and talk about how I made it work using my code, to avoid repeating myself I may not cover every detail of every single module but this will focused much more on why it works rather than how it works and every detail about that. (Also due to the flow diagrams size I have tried to crop it in a way in which it is readable and have left a full-size uncropped image at the bottom).







Looking at the flow diagram can be quite daunting, but I’ll talk it though step by step explaining what everything does and all that code that correlates to each step.

First, the ‘guessing’ function is called in the ‘game’ function, the gallows, score, current state of the word and possibly hint have been displayed to the player and they are ready to guess. The function starts and some variables have to be defined, an integer called validGuess which will determine what error message to display as well as when the function can end, a char array called userGuess which holds the users input and a Boolean called corretGuess that is flagged if the users guess is correct and letters need to be printed. All other variables in this function are referenced or passed from other parts of the code and will be explained one they are reached. The first thing the program does is enter a do-while, as previously mentioned I have been using these for error checking purposes and if the validGuess variable doesn’t change only then is the function allowed to end. At the top of the do-while loop is a switch statement which tests the validGuess variable and prints out an appropriate message. One the first loop it will always print out the default message, as the user has not entered anything yet there is no error message needed and the default value of validGuess is 0, the function assumes the user’s input is correct until proven otherwise. After the correct message has been displayed input is gathered from the user and assigned to the userGuess variable. Immediately after the user has input their choice the program detects for any failures regarding ‘cin’ and if any are found, clear() and ignore() are used to repair the stream before any problems arise. Although this is a form of error checking even if this occurs vaildGuess is not changed, this is because it is redundant and the only error if would be flagging (i.e it being more than one character long) will be flagged later any way if that is the case. The user input is converted to lower case before ‘isalpha()’ is used to check if the input is a letter, if it is not a letter it is flagged as error 3 and validGuess is changed. The input is then checked for length and if it is more than one character long it is flagged as error 1 and validGuess is changed appropriately.

After this it runs a for-loop for every letter of hiddenWord (this is an array containing every letter in the word the user is trying to guess and is pulled from a different part of the program) tests every item in its array to see if any of them match the users guess. If the users guess does match up it is first checked against every letter in correctArray (this is an array containing all letters in the word that have already been guessed correctly) and if it has already been guessed it is flagged as error 2 and validGuess is changed appropriately. After this for loop is complete it checks to see if validGuess is still zero, if it is still zero and has reached this point it means that the user has entered a single character that it is the word that has not already been guessed and correctGuess is changed to true to signify this.

After it has finally exited the first for loop it checks if correctGuess was true and if it was, it increases lives (a variable to determine how many guesses the user gets) by one, increases score (a variable to keep track of how well the user is doing) by 500\*streak and increases streak (a variable to keep track of how many correct guesses the user has gotten in a row) by one.

If it exits the first for loop and the guess was not correct it runs a for loop and checks it against every letter in incorrectArray (an array that stores all the guesses the user has taken that are not in the word), if the input is already in incorrect it is flagged as error 2 and validGuess is changed appropriately.

After this it does one final check to see if it has reached this point without any errors, by this point any valid and correct guesses will have skipped this section so anything that makes it to this point and is still flagged as 0 (A.K.A valid) must be valid but be incorrect. In this case the letter is added to incorrectArray and the streak is reset to 1.

At the end of the do while loop if the user’s input was valid and validGuess is still 0 the function ends but if it is anything but zero it repeats and uses validGuess’s value to display an appropriate error message when the loop restarts.

**4. Testing and Conclusions: 5%**

I used a fairly similar method of testing for errors in very part of my code that required a user input so my first test we to get a set of characters that were all wrong in different ways and test them multiple times at every opportunity. I needed to make sure each input would not intake any value that was to long or any value that was not of the appropriate type, I also needed to check that inputs of a ridiculous length did not break the program. Originally I simply used an array with a length of 1000 instead of 100 to hold user inputs but this was inefficient and simply didn’t work, after figuring out a way to ignore characters after a certain point and a way to clear the buffer I was able to fix this problem. The next test was to give it to users and after giving it to multiple friends to test there were no errors with the code but on a single hint I had used a type of punctuation that was not part of regular ASCII set but I had never seen it because there are a lot of hints and I could test all of them. After this finally playtest I was confident that the program worked.

Throughout the development process there were many small changed and things that I tweaked over time but by far the thing that changed the most was my validation checks, originally there were no validation checks as my planning in the early stages was honestly not very good. After this I simply added basic checks to test if an integer was to high or low or if a character was a letter and not a punctuation mark, but this was also poor as the entire program would break if the wrong type was assigned to the wrong variable. Eventually I had every user input being entered into a 100 character array which allowed me to test for type and length and finally I started implementing do while loops and switch statements to allow for specific error statements as well as stopping invalid inputs getting to different parts of the program. It started as nothing but quickly became a central part of my program.

As mentioned, multiple times before my planning was not great early on and I didn’t have a full flow diagram completed till about halfway through the program but after that was created everything fell into place a lot easier. I had a lot of individual pieces that I was working on but when I could see how to fit them together in gave me a new perspective and how they fit together and how this worked to the programs benefit, like in the case of the debug mode being changeable from the main menu but effecting many different modules, and how it could be detrimental when I would get buffer overloads from other modules. After all is said and done though I am very happy with how everything fits together and am proud of the work I have done.

**Appendices .References**

Brightspace, 2020. Week 13 Part B ITP Menu and Input Validation Schemes V1\_1 AY2019 2020[online]. <https://brightspace.bournemouth.ac.uk/d2l/le/content/71176/viewContent/499001/View>

[Accessed 29th December 2019]

Code snippet in slide 21 modified and used in program

Riddles.com, 2020. Classic riddles[online]

<https://www.riddles.com/classic-riddles>

[Accessed 21st January 2020]

Riddles were used as inspiration for the cryptic clues categories