evALUATION

# success criteria

Overall, the test cases for all the ERL modules, 1 to 11, were complertely successful, which I proved when doing all the test cases in the testing section. Because my success criteria are just descriptions of how the ERL should be executed, these therefore all met the criteria, so the success criteria for the ERL have been completely met, with all my test cases working as expected.

Now, I am going to analyse the module 12 success criteria for the interface. These do not have subjective tests, like the previous, so I am going to try and prove that the criteria has been met, using the knowledge from my stakeholder’s testing to help with this.

|  |  |  |
| --- | --- | --- |
| No. | Criteria | Proof and Success? |
| 12.1 | Have an input and an output window, for which ERL can be entered, and then the result is displayed in the output window whenever a run button is pressed. | Two, clear distinct windows, with an obvious run button in the corner, making it easy and obvious to use the core features of the program, so therefore I feel that this segment is successful. |
| 12.2 | Allow for user inputs to be taken directly from the output window, like a console interface, and allowing the user to properly interact with the program | Allows text to be written into the text field but does not allow the cursor to move outside of the input area in the window, and none of the outputted text can be deleted, only the entered text.  This simulates how IDEs take inputs, so therefore this is successful. |
| 12.3 | Allowing for ERL files to be uploaded from the local computer, and put into the input window, and also downloaded, where the current contents of the input window will be saved into a file | Has the two buttons in the top right of the editor window, which will correctly function, with the folder opening the local file system, where a text or erlcode file can be opened, and the save button downloading an erlcode file with the input’s current contents.  This was shown in the video demo and works correctly so I count it as a success. There may be some issues with the icon, as it confused Tanish, my stakeholder, however when he learnt the interface it was fine so I think this criteria was successful. |
| 12.4 | Allow files to be uploaded from the local computer, as well as modified in the interface, so that the user can use the file handling in the ERL and see the results. | The interface for the files is very clear, with labelled buttons, with the only slight issue being the main icon.  This makes it very easy for the user to navigate between files, as well as to create and upload their own files, and write directly into them.  Additionally, the 4 tools that appear on the right-hand side add additional functionality to the original criteria, and renaming, downloading, resetting, and deleting files directly in the user interface makes it very easy to use files with the project.  Overall, I think that this criteria is a success. |
| 12.5 | Allow for basic highlighting of different keywords in the input window, as well as different colours for strings, numbers, Booleans, methods and each type of token that the language contains. | Above are some examples of this, and it makes it a lot clearer to distinguish the different keywords. Additionally, I also implemented auto-indentation based off the different keywords, which makes the code written a lot more readable.  Therefore, I feel that this section is definitely a success. |
| 12.6 | A resource containing a clear description of the ERL is available, with details on the syntax to be used by students, as a more friendly alternative to the badly organised and confusing specification. | I have a 216-line guide to the ERL, available from a link on the main page of the interpreter. This goes through every bit of syntax, explaining how to use it and is a much more logical order. The document is also editable, making it easier for students to modify and practice code in it.  This is also a success, and a small extract is shown above. |
| 12.7 | A resource containing exam questions and solutions is available, so that a student can use it to help practice questions for an exam if they are using ERL in a revision setting rather than inside of lessons | I did not manage to implement this criteria, and that was partially due to time constraints. I could have easily implemented a very basic version, just a separate webpage with a bunch of questions and answers attached, however I feel like a more advanced version could be added.  Therefore, I am going to leave this feature for a potential future update of the interpreter and have excluded it from this phase of development. This means that I have failed this criteria, but it was a less important one so is not a major issue, and I feel like I can make a much better product out of it, but I will discuss this |

### my conclusions

In total, out of the 80 criteria, only one of them was a failure, which I am very happy as. Seeing as I was originally tentative that I might not get all of the ERL features implemented, I think this is a great result, and therefore the whole project should be a success.

Likewise, all of the feedback from my stakeholders was very successful, and the session with the year 10 class was also very positive, so therefore I am happy to t

# code maintainability

Overall, I am very happy with how maintainable the code is. In a lot of my past projects, my code quality has been very poor, but I make a deliberate effort to improve it in this project, which made the process a lot easier, especially when returning to old code later in the process.

## A screenshot of a computer Description automatically generatedsmall scale maintenance

A screenshot of a black screen

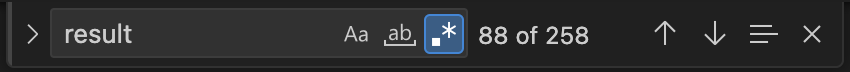
Description automatically generatedThe first main example of how the program has been maintained is with the structure and naming of the different classes. This consistent naming allowed me to easily identify and navigate the different classes in the program, as well as the inheritance orders being very logical.

A screen shot of a computer program

Description automatically generated I also made heavy usage of commenting and good variable naming, especially for algorithms that were a lot less self-explanatory is nature, such as the method for parsing if and switch statements in the parser.

Because these methods look very complex, the comments easily allowed me to understand how they worked when re-reading the algorithm after a month.

The names of the properties and methods of classes were also made as generalised as possible, as it allowed for everything to be more easily identified.

If I had a main issue with my naming, it would be the use of the keyword “result” which I used 258 times throughout the entire program.

Even though it is a temporary keyword, used to store intermediate results and values, I still like I could sometimes specify slightly more the exact purpose of the variable. This mainly occurred at the start of the project, and later I named variables more descriptively as I often got confused in development as to the purpose of a “result” variable.

The use of result is that it can be used multiple times with different purposes, which I did use at many points to save memory, as a generic variable to hold different bits of data throughout a function, which meant less unique variables had to be created so the performance would increase.

One example of the reusability is in build\_for\_loop(), where I used result to store the assignment, final value, step value as well as each line of contents in the loop, to check for errors and so on, saving 3 different memory locations needing to be made.

However, in lots of cases, such as for arithmetic and logical parsing, using result is very redundant. Here, result stores the comparison operator that is in the middle of two different expressions, which is very poor naming because result is not used in any other context, and it provides absolutely no insight into what the variable does without having to look at comments, so I feel naming could have been improved.

## larger scale maintenance

Even though I am very happy with the smaller scale maintenance, it is how the program is structured on a larger scale that has made the process very easy to work on and will make it easy to make new additions in the future due to how flexible everything is.

This essentially means that, for any new feature that needs to be added for the syntax of the language, the corresponding code in the Lexer, Parser and the Token’s evaluation code needs to be added, and then it should seamlessly work with the rest of the pre-existing code. This is because the parser works in a top-down way, so as long as the new token or structure is added to the parser in its original place, it will be built into the ast correctly, where evaluate() will be called on it.

The best example of this is before the half-way evaluation, when I needed to add NOT because I didn’t do it in the logic section. However, when adding it, all I had to do was the following:

🡺 Add the corresponding class, extending Token with a single child  
🡺 Add a “NOT” case in make\_identifier() to create one of these tokens  
🡺 Add a method in the parser to check for it, and set its child to the comparison parsing method’s result  
🡺 Change the AND and OR parsing methods to call this method instead of the comparison method  
🡺 Set the Not classes evaluate() method to return the opposite boolean of its child’s evaluate() value

Apart from these features, none of the original code had to be changed, as every module is very independent. This was a deliberate choice to avoid dependencies between the different modules, and as long as everything is made in the lexer, weaved correctly into the parser to produce the correct abstract syntax tree, and given a correct evaluate() method, the system should correctly function with the feature.

## negatives

A screen shot of a computer program

Description automatically generatedAt the end of module 11, I was very happy with how maintainable the project was, because everything was very centralised, such as with having a single place to output all errors and printing.

However, when implementing the user interface, I feel the code became slightly messier, with lots of the different interface code scattered across multiple different locations, without many central methods to control outputs, inputs and waiting.

For example, I do not find the code on the right very readable, and despite all the commenting describing the steps that it does, the approach to taking inputs is not a very logical process, and I find the approach to be slightly haphanded.

This is true with some other interface aspects, such as needing to wait for 4ms after each time something is printed in order to stop loops crashing the program, a

# future of the project

For potential future development and expansion of the project, there are two different types of additions I can make: the first being changes to the actual language, but secondly being additional interface and quality of life features which would make the experience better for a user.

## language changes

A white background with black text

Description automatically generatedThe entirety of the GCSE exam reference language has been implemented, so there is no expansion I can make for GCSE students. However, the language used by OCR for the Computer Science A level is a superset of the GCSE Language, so follows the same base syntax but had some additional features which could be implemented.

This includes some small changes, such as allowing arguments to be passed to functions by reference or by value using the colon syntax as shown on the right-hand side.

Additional changes to file handling are also made, with unique functions for opening a file depending on if it will be read from or written to.

Both of these changes are relatively minor, so should be small changes to implement and would work well with the existing system.

### object oriented programming

A screenshot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generatedThe main addition the A-level Exam Reference Language makes is the addition of classes, which then includes public and private methods and attributes, constructor methods and inheritance.

The full list of additions can be seen on the right, as specified in the A-level specification, and it include a lot of features, which would be a massive upgrade to the system.

This large addition would probably more complex than any other module that I have currently incorporated into the language, as previously it was adding functions and this will be significantly more complicated to add.

However, adding these features should be very useful, as the majority of the work will have already been done for the bulk of the language, and it will vastly increase the number of students who can use and benefit from the language.

Additionally, because the complexity of the algorithms presented for a level students will be significantly more than the GCSE counter parts, so the interpreter should be a lot more useful because it can run these much more advanced algorithms which would take even longer to try and run by other means, such as by using a trace table.

More generally, this would also engage a-level students with the basic features as well, such as the formatting of loops and if statements, also making them more familiar with the basics which will still be useful to them when in examinations.

Also at this point, it would be technically possible to code the whole project inside of the language, because adding classes adds the layer of complexity needed to create the interpreter.

Overall, these additions would be very useful, so I could see myself doing this in the future, after my A level is complete.

### error handling

The error handling of the project can always be upgraded, by adding more descriptive messages to make it a lot easier for the user to fix. Currently, I am sure that there are some cases which still break the program when trying to run, because I cannot account for every single possible combination, but hopefully bug reports from users over time should help to eliminate these when they are discovered.

For the addition of new errors, I feel like existing error messages could easily be improved by adding additional cases as to what cause the exact error, as it could more usefully give feedback as to the issue, rather than a more generic, default message. This is more specifically focusing on the syntax of different structures, such as the declarative lines for if statements an while loops.

A black background with white text

Description automatically generatedFor example, the generic error message for the incorrect token being present after an if condition could be expanded to undergo additional checks, such as if the invalid token was a colon, like it is in python, then the error message could explain that ERL uses “then” instead of colons, to target any python users who have decided to use ERL and are unfamiliar.

This can be expanded for other common misconceptions, mainly from syntaxes from other languages, to make the feedback more useful for any new students.

## usability features

There are also additional features which could be implemented that would not affect the core purpose of the program but would make it more convenient or easier for students to use.

Primarily, these ideas are the discarded ideas from the Analysis stage, and although they would have been beneficial, they were beyond the scope for this phase of development. However, I could easily see them being added in future updates, as some of them would be very useful.

### trace tables

My favourite potential feature from the analysis stage is adding trace tables to track how the variables’ values change when the program would be run.

Not only would this help with debugging any logical errors in their code, it would also tie in perfectly with the specification. This is because trace tables often come up in questions, usually tied with ERL code with questions asking the student to trace out the values of the variables throughout. This would allow the user to easily check if the trace table they made was correct and would extend the usability of the interpreter.

Therefore, this two-in-one feature would work perfectly, and is a lot more of a student-friendly approach to debugging programs, compared to more complex versions used in actual IDE’s, so I feel like it is a perfect fit for the program and the target audience.

# next time?