Variables

# design

## rough Description

This is an initial abstracted version of the plan as there are several ways I could approach this.

Firstly, the new Identifier Class will be created, and will be produced in the Lexing stage, containing a name method which is the string of characters it was created with.

However, as we may have multiple unique Instances with the same name, all instances with the same name must access their value from the same location.

This is where the Data Type class is used, with each instance having a name and value property, with the value being able to be accessed and changed.

The Data Types will be stored and created by a class called Symbol Table, which will include an array of all its children Data Types. For now, there will just be a single, global Symbol Table instance, and all new variables will be added to it, however in the future creating multiple tables allows scope for functions and procedures to have their own local variables.

Each Symbol Table will have a method which will take in a string, being the name of the identifier, and will search through their array and return the Data Type with the corresponding name. so that the value can be accessed, otherwise this method will create a new Data Type with the corresponding name that was entered into the Symbol Table method, with null as the starting value.

A new Identifier Error will be added if a variable with value null is trying to be accessed, as it means that it has not been declared in an assignment, and this Class will extend Error.

### assignment

Before the parsing can be added, a new Equals binary operator needs to be added. This will have the expected position property.

There will be a new method in the parser which will be called in parse() if the second token in the list of tokens is an Equals. The parser will therefore need a reset() method, as we will have to revert the continue() if the second token is not an Equals. From there this assignment() method will be called.

This will then construct the abstract syntax tree with the following BNF:

<assignment> ::= <Identifier><Equals><expression>

This is relatively straightforward but will need to check for every other possible error in how it is typed, with lots of checks for potential Syntax Errors.

At the end, this will return an Equals token, with the Identifier as its left child and expression as its right. From here, when executed the Equal’s evaluate() method will be called.

The equals evaluate() method will call upon the Identifier’s evaluate(), which will call the global Symbol Table’s finding method, and will return the corresponding Data Type, which the Identifier will return. The Equals will then set the value property of this child to the value from its right child’s evaluate() method, which would be a whole expression as the value.

### constants

A new class will be created called Template, which won’t be used by a single keyword, but for any keywords which do not contain any code, and will never be evaluated, but are placeholders.

These will have a tag property which will represent their type, for example when the Lexer detects a string of characters “const”, it will add a Template token with the tag “const”.

This will then be recognised by the Parser, and if it is present, then the assignment() method will give the Identifier a property that constant is true, which will then in evaluation be passed to the Symbol Table, which will then be passed to the Data Type.

If a Data Type ever receives constant being true, it will then record this and no longer allow its value to be changed, returning an Identifier Error if there is an attempt to do so.

## basic implementation

### lexer changes

Firstly, a new global constant called LETTERS will be needed, like our DIGITS example for the Lexer to use.

🡺 Identifier Class

We will need to create a new class, extending Token with a name property. This will have to be proper integrated into the existing code, such as being a possible factor in the bodmas expressions.

🡺 make\_identifier()

This will be a method in the Lexer class, to be called in make\_tokens() when the current character is in LETTERS. It will then continue cycling through the characters until a character than is not in DIGITS or CHARACTERS is encountered. It will then return a new identifier with all the characters that it had cycled through passed as a string.

This will be done using an array called name where the letters will be pushed to before finally being converted to a string. Also, the original position needs to be passed, so will have to be recorded.

### data type

This will be a generic class for all basic Data Types. It will have a property for the name as a string and value property. They will also have a constant Boolean property, and where this is true, they will have a declared property so that constants can keep track of whether they have been defined.

🡺 value getter

I am going to use getter and setter methods for this class, as they are something I have not experimented with before. The getter will check if value’s current value is null, which would represent an undefined variable. If it is null, it will return an Identifier Error, otherwise it will return the value.

🡺 value settervinar

If the constant property is false, then it will assign the new value. Otherwise, it will first check if it has been declared. If it has, it will return an Identifier Error stating that it has already been assigned. If not, it then sets declared to true and then assigns the new value.

Note that the starting values of name and constant will have to be passed to the Data Type in its constructor method.

### symbol table

Only property will be an array called table to store all the Data Types.

🡺 find()

Find will take in the details of the Identifier and will search through the array of Data Types to find one with a corresponding name and will return it. If one does not exist, it will create a new one, adding it to the array, and creating it with the same name and details about whether it is a constant which are passed into find() initially. This new Data Type will be returned from the method.

The Identifier’s evaluate() method will call upon find(), passing it details, and then Interpreter will return the corresponding Data Type with its details.

### parser upgrades

🡺 Equals

New class for the ‘=’ operator, extending Binary Operator. Has a position property and two children. The left child will be an Identifier, and the right child will be the root node of an expression, so when evaluate() is called, it sets the left child’s value to the evaluate() of the right child.

🡺 Template Keyword

A generic class for keywords which are only used in the parser, and do not have any evaluate() functionality. Has a string tag property to represent their type.

🡺 parse()

Will now call upon assignment() instead of evaluate()

🡺 assignment()

Will have a series of checks to build the corresponding BNF definition.

<assignment> ::= [const]<Identifier><Equals><expression>

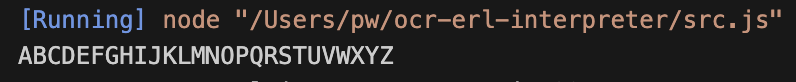
If the stream of tokens does not start with an Identifier, const, or have Equals in the second position then it will have to call evaluate() and return it. When checking for Equals, if it is not present then it will have to use a new reset() method to set the position to -1 and then call continue() again.

Lots of error checking still required.

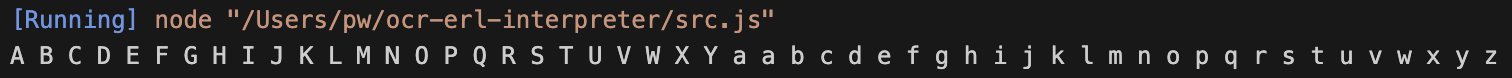
# development

## initial additions

### characters

I initially tried to generate the list of LETTERS from a line I found on the URL:

<https://codegolf.stackexchange.com/questions/71613/generating-the-alphabet-in-javascript>

This worked for just uppercase, but when trying to expand it to include lower I could not alter it to work.

Therefore, I went for the more basic option of unpacking a string of every character into the list.

### A screen shot of a computer Description automatically generatedA computer screen with text on it Description automatically generatedA computer screen with text Description automatically generatedlexer changes

This initial introduction followed the design stage exactly, ensuring to add the additional check to make\_tokens() to then call the make\_identifier() method.

As seen below, the two test cases functioned.

A screen shot of a computer code

Description automatically generatedA screen shot of a computer code

Description automatically generated



### A screen shot of a computer program Description automatically generatedsymbol table and data type

A black screen with text and numbers

Description automatically generatedA black background with white text

Description automatically generatedThis was my initial experimentation with using getters and setters. I realised that value had to be renamed to \_value, as otherwise it clashed with the property names. The “Error” output is also temporary as Identifier Errors have not yet been implemented.

A black screen with text and numbers

Description automatically generatedA black background with white and blue text

Description automatically generated

The two test cases then correctly function.

A screen shot of a computer code

Description automatically generatedI also had to add a name property, which I do not have a screenshot of, but this is declared in the constructor() method and takes its value from the parameter name, so is defined upon creation of a Data Type instance.

A black background with colorful text

Description automatically generatedNow implementing the Symbol table.

The initial test did not work, but the issue was that the find call returns the Data Type, but .value still then needs to be called in order to change the value, so the correct version is:

test.find(“hello”).value = 7

A computer screen with colorful text

Description automatically generatedThe next issue was that the newly created Data Types were not added to the table list, which was a quick fix, after realising that using [-1] was a python feature so had to be replaced.

### integration with existing code

A screen shot of a computer code

Description automatically generatedFirstly, global is declared as a variable for the whole program to use – this is temporary until functions are implemented. Identifier is also given the previously used find call as a value property.

A black background with blue and green text

Description automatically generatedA black background with colorful text

Description automatically generatedA screen shot of a computer code

Description automatically generatedThen, the basic Equals instance is created, with its corresponding character check in the Lexer’s make\_tokens()

Finally, the factor() method must be changed to also include Identifiers alongside Integers and floats, so that they can be parsed and included into expressions.

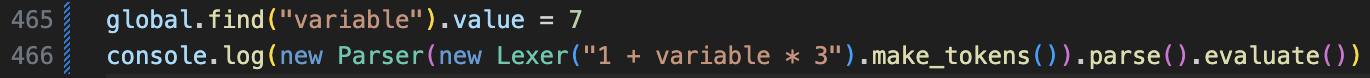
This is the test expression to be parsed, and initial problem occurred with Symbol Table being accessed before initialisation, so the Data Type and Symbol Table classes were moved to the top of the document in order to fix this.

A computer screen shot of a code

Description automatically generatedAfter this, the correct abstract syntax tree was produced, which only outputs Error due to variable not being defined yet, so it works as expected.

A computer screen with text on it

Description automatically generatedIn order to let it evaluate, the value of the Identifier must be returned.



A black screen with text and numbers

Description automatically generatedA black background with white text

Description automatically generatedHowever, this did not work, with the test case, as the old definition of value in Identifier was static and never changed, so a getter must be used so it calls find() on each access.

Changing this produced the expected result of 22.

## assignment

A black background with blue text

Description automatically generatedA computer screen with text

Description automatically generatedA screen shot of a computer code

Description automatically generatedThis was my initial attempt of implementing assignment.

A screen shot of a computer code

Description automatically generatedA screen shot of a computer

Description automatically generatedFor now, this excludes error handling until they are added, but it is clear to see the expected for how an assignment is laid out. The checks for Identifier and Equals ensure that a normal expression can still be parsed, even if it starts with an Identifier, resetting if need be.

The two test cases confirm that assigment works, as well as the new reset() method correctly functioning, and still allowing expressions to parse with Identifiers at the start.

### evaluation

A black screen with text and numbers

Description automatically generatedA screen shot of a computer code

Description automatically generatedThe initial attempt at creating an evaluate() method for Equals is shown on the left, but upon trying to run variable = 7 it resulted in the outputted “Error” being printed from the Data Type get method.

A screen shot of a computer code

Description automatically generatedA screen shot of a computer code

Description automatically generatedThis therefore means that the identifier’s evaluate() call is returning the value of the Data Type, rather than the Data Type instance itself. However, we cannot change evaluate() to return just the Data Type, as then Identifiers would not properly evaluate in arithmetic, so therefore a new evaluate() type method will have to be added to Identifier to allow both of these cases.

A screen shot of a computer code

Description automatically generatedassign() will be the new evaluate() equivalent to be used for Identifiers, so therefore as shown above, the Equals evaluate() method now calls assign() on the Identifier in order to be returned with the instance.

A computer screen with text and symbols

Description automatically generatedWhen running a test case of this however, it still did not work correctly. In order to check if the issue was with assignment or evaluation, the test case on the left is created, and will output the instance of the Data Type after it is assigned.

A black background with white text

Description automatically generatedA computer code with text and symbols

Description automatically generated with medium confidenceImmediately this showed that the final line in Equals’s evaluate() method is incorrect, as it should set the value of left to right, rather than left to right, so has been updated.

Now assignment properly works, however when trying to evaluate() in the terminal, it still crashed to program, so further debugging was required.

### debugging

A screen shot of a computer program

Description automatically generatedThis debugging process took a while. My technique was not the cleanest, as it involved adding lots of console.log() calls throughout the code to check the values of variables when trying to run the program, however it eventually led to the issue with the solution.

The problem was with the assignment() method. More specifically, the checks to see whether the stream of tokens was an assignment or an evaluation.

My testing made me realise that I did not understand the order of operations in JavaScript – ironic when building an interpreter. When checking if the current token was an instance of a Class, I had initially but the NOT operator (!) immediately next to this.token, however because the Precedence of ! is higher than that of instanceof, and as I was checking if the token was not an instance of the class this meant that it was not creating the intended result.

The solution is moving the ! outside of brackets which contain the instanceof expression. This therefore performs the check first, and then returns the opposite, as before it was comparing the NOT of the current token, which is a completely different check.

A black background with text

Description automatically generatedA screen shot of a computer code

Description automatically generatedThis new implementation stopped a JavaScript error occurring, but now an unexpected custom error message occurred instead.

After changing this if statement and realising that this was incorrectly used in a lot of places instead of self, it worked as expected.

### small changes

A screenshot of a cell phone

Description automatically generatedA black background with blue text

Description automatically generatedThere are 2 slight changes made.

A black background with text and numbers

Description automatically generatedA black background with white text

Description automatically generatedFirstly changing Equal’s evaluate() method to return null, so that undefined would not be outputted after an assignment.

A screen shot of a computer code

Description automatically generatedAdditimonally, adding extra lines to the parser to return null if an empty stream of tokens are inputted, as previously an empty instruction would cause the program to crash. This also has to be accounted for in the temporary runner.

Overall, it is now clear that assignment works as intended, and variables can be both defined and referenced to work as intended.

## error handling

A computer screen with colorful text

Description automatically generatedIdentifier Error is essentially a copy paste of the other lasses with some details changed. Inefficient as lots of reused codes with the errors so could be redesigned later to only use a single class for all errors.

A computer screen with text and numbers

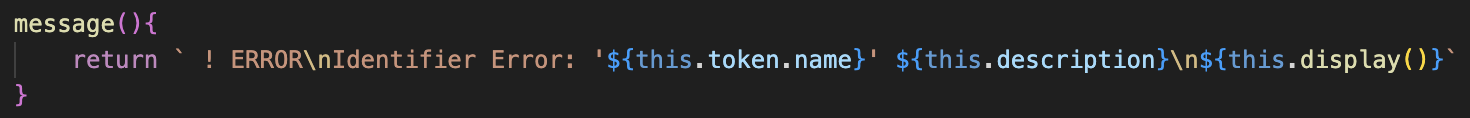
Description automatically generatedData Types will need access to the last token where they were accessed for the token in Identifier Error, so must redesign aspects of the program to allow the Identifier token to be passed through evaluate(), through find() in Symbol Table to be saved into the correct Data Type’s lastReferenced property.

A screen shot of a computer code

Description automatically generatedA screen shot of a computer code

Description automatically generatedMessage: “Variable was not defined”

A black background with white text

Description automatically generatedThis created the Error message on the left, which was correct but not nicely formatted. I therefore changed it slightly to create a new error message.

A computer screen with white text

Description automatically generatedTo conclude the error handling, I tested all of the cases to ensure that it works as expected.

A black background with white text

Description automatically generatedA computer screen with white text

Description automatically generated

## constants

### A screen shot of a computer program Description automatically generatedA computer screen shot of text Description automatically generatedlexer implementation

Following the plan, adding the new class.

A screen shot of a computer

Description automatically generatedIt is created in the make\_identifier() method and will have a long switch statement to return all of the keywords, and the default case will be to return an Identifier.

After correcting the syntax error of not having new, the test case produced the expected stream of tokens.

### A screen shot of a computer code Description automatically generatedA black background with white text Description automatically generatedparsing

A screen shot of a computer code

Description automatically generated

A screen shot of a computer program

Description automatically generatedI have decided to move the checking for whether the assignment is a constant or not into the parse() method, as it provides more consistency. Therefore, every time assignment() is called it will be one, and not potentially an expression(). This new addition in parse() is now for the assignment starting with the constant token, which passes in the tag as an argument to assignment().

A screenshot of a black background with white text

Description automatically generatedA screenshot of a black background with white text

Description automatically generatedA screen shot of a computer program

Description automatically generatedThe updated assignment method() will replace the Identifier with a new one with the constant property set to true. With hindsight, this is inefficient as instead of creating a new instance, the property could have just been changed, but that can be modified later.

A screenshot of a computer program

Description automatically generatedAfter running a test, the redesigned parse() checks work, but now every single Identifier is treated as a constant with no error messages.

A computer screen with text

Description automatically generatedA computer screen with text

Description automatically generatedThis turned out to be due to a small syntax error, however constants did still not function as intended, which lead to a further debugging which eventually led to me outputting the values of constant and declared at the start and end of the value setter to see how they changed.

A computer code with text

Description automatically generated with medium confidenceSomehow a “was not declared” Identifier Error occurred, which was only present in the getter function, which confused me significantly. Additionally, the after output never occurred, which meant that somewhere in the setter method it was stopping prematurely.

### the fix

Eventually I concluded that it was due to the system of handling errors. In the program, whenever a method occurs an error, it returns an instance of the Error class instead of an actual result, therefore meaning that the code that calls on it must always check whether an Error had occurred first.

A computer code on a black background

Description automatically generatedA computer screen with text on it

Description automatically generatedHowever, because I was using a setter, an Error could never be returned and checked in the Equal’s evaluate() method. Therefore, I will replace the value setter with a method called set(), with very similarly. functionality, taking in the new value as a parameter but allowing for this proper error handling.

.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedThis change has allowed the error handling to correctly function as shown by the tests.

### cleaning up

A black screen with white text

Description automatically generatedA screen shot of a computer code

Description automatically generatedSome of the cases where an invalid input had been entered causes the program to crash, such as nothing following the equals sign, as well as no Identifier being present after the const keyword.

A computer screen with text

Description automatically generatedMy solution was to add additional checks to see if the current token is null throughout assignment() to avoid any of these cases.

A screen shot of a computer code

Description automatically generated**A screenshot of a cell phone

Description automatically generated**Now that the incomplete inputs are dealt with, the final decision is whether constants that have been defined should be allowed to have their value changed by another constant keyword.

A black background with blue and white text

Description automatically generatedA screenshot of a computer

Description automatically generatedWhen attempting to research this, it was difficult to find a standard, so I tested it with JavaScript and it allowed this to occur – so therefore I did the same in my code.

This is very bad programming practice anyway, but it is a slight change that I can decide upon whether I want to keep or remove later to see what is best, and this concludes the development for variables.

# evaluation

|  |  |  |
| --- | --- | --- |
| No. | Criteria | Implemented? |
| 2.1 | Allow for identifiers to be used to store values, defined with the ‘=’ syntax, which can later be read. | Badge Tick with solid fillCompleted |
| 2.2 | Ensure the new variable system integrates with the existing arithmetic, for both setting values and inside expressions. | Badge Tick with solid fillCompleted |
| 2.3 | Let constants be created with the “const” keyword, which are variables that will return errors if the user attempts to redefine them | Badge Tick with solid fillCompleted |
| 2.4 | Ensure full error handling, so that variables cannot be accessed before assignment, and that all invalid cases are accounted for. | Badge Tick with solid fillCompleted |

Overall, the success criteria for this module have been completely implemented, however I will not be running the test cases because they require more features to be implemented.

In terms of the code itself, I am not currently happy that the global symbol table is being stored in a variable, rather than as a property to a class as it does not feel very complete, however this can be changed later in the program, but likely when subroutines are added because the scope system will need to be changed. Apart from this, I feel like this implementation is very good as it allows it to easily be expanded later on in the project.