Project 1 Writeup: JST Language

Abhi Agarwal

Excuse me for the syntax - I wasn't fully able to incorporate unicode into Latex, and so I will use < >, and [ ] to represent some of the Unicode characters in Hacs. My approach has been to put the longest statement that's valid first and then go down.

1 How to test

There are certain tests that don't work with my given version. It's very strange, but I made some decisions that didn't allow for certain types of syntax, and I haven't been able to fix them in my program. For example:

Function calls () have to be ( ), with a space in between. I tried to fix this issue, but I still haven't been able to understand why this occurs within my given syntax. I have attached files that will run - my apologies for this. The arguments are fine though, we don't need to declares spaces when you're defining a function or a method in a class.

The test files: Hello.jst, String.jst work very well, and execute. The only change is to put spaces between the function calls, and arguments.

I still have some errors I haven't been able to fix within my code. I wasn't able to get my Expression/Operations section working ideally. I had a bug where it wouldn't expect my return function because I wasn't able to understand how to declare the n¡=0 part of Fib.jst. This part was quite tricky for me as I didn't really understand the use of the @ symbol very well, and therefore wasn't able to utilize it to its fullest extent.

2 Design decisions

2.1 Comment regular expression

There are a lot of ways to go about this. I thought it would be great for me to design something that even tackled something like:

```
var m /* hello this is a comment */ = g
```

There were different ways for me to parse this, but I presented the simplest way I could find. I couldn't get around the problem of:

```
var m = ``/* hello this is a comment */"
```

Because I didn't really understand how to check if the condition of it being inside " was true. I tried a bunch of different ways, but then I left it. Design wise my syntax wouldn't support that, and I made that decision when I didn't include it. It limits the scope of my grammar because it removes support for Comment.jst.

I thought the piece of code below was important to note. This meant that I was able to assign an Expression to a Object Literal. This wasn't included I believe, but it was a design decision I made because I thought it was important for people to create objects. I'm not sure how this would have worked if I hadn't referenced it specifically inside the Expressions.

[<E@2> = <LiteralTwo>]@1

I also had to adjust ArgumentSignature a little because it wasn't working when I was using pure regular expressions. I made it a little nested to explain what I was doing very clearly. The comma shows that there can be multiple arguments, but if it isn't then it would also return.