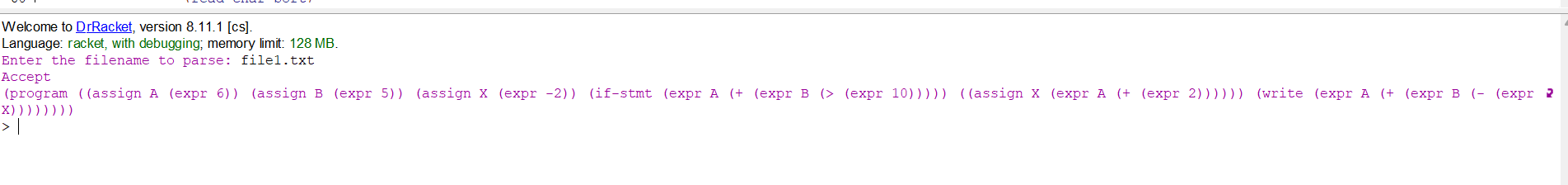
Raine Morrigan

CS 441

3/28/25

Morrigan Parser Summary

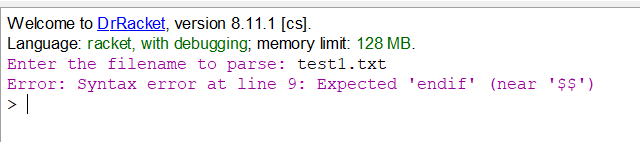
Seeing the grammar, I was at a loss for where to begin. I ultimately started the development process researching BRAG (a Better Racket AST Generator) to see if it would be a good tool to use as a launching point. Even after spending a few attempts at reading the documentation, it seemed still beyond my abilities, so I changed tactics. Having good experiences with Claude in the first assignment, I decided to seek help there again. The code in the first incarnation was riddled with syntax errors and missing information. I used the same prompt on ChatGPT to compare and had much the same story. After spending several hours debugging and trying to understand what was going on, I decided to try paying for Claude. 

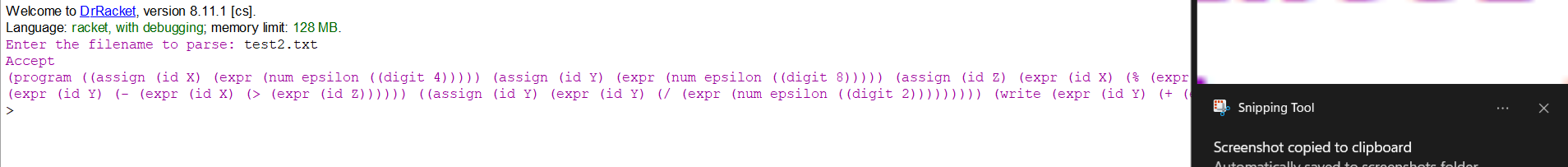
file1.txt result

After I paid for the month, Claude was able to take a simple prompt to fix the syntax errors that remained and gave me the first compliable version. I did not know it was working as I had a small confusion for a while before I was able to fix the missing semicolon in file1.txt. I tested each of the given samples and received the appropriate warnings from the README. With each sample input giving expected errors or successes, I wanted to work on testing and expanding the code.

I used ChatGPT to create more text files to use as samples as well as asked Claude for RackUnit test cases. The files provided by ChatGPT included two symbols that weren’t covered by sample texts, “/” and “%”. We were told that were weren’t limited to only the samples, so I tried to expand out my parser as much as I could since I was able to use AI to clean up the code so quickly. The test cases provided have not yet been included as I decided to make my own. Ultimately, Claude didn’t give extensive enough test cases to be worth running, so I leaned more into the test file method.

With the new files made, I tested them and ran into those foreseen issues of not included symbols. I debated if they were meant to be included, but as mentioned before, we were not limited to these samples and modulo and division are on calculators. So, I decided to add them manually. Additionally, I changed the testing files so that there were more errors than just missing semicolons. Below are my generated tests results (though they are the latest version now, as I later need more changes):

test1.txt | missing endif line 9 

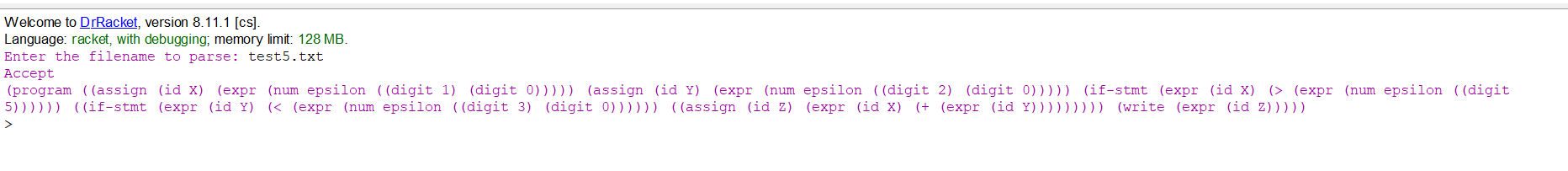
test2.txt | Accept 

test3.txt | missing endif line 6 A screenshot of a computer

AI-generated content may be incorrect.

test4.txt | Accept A close-up of a white background

AI-generated content may be incorrect.

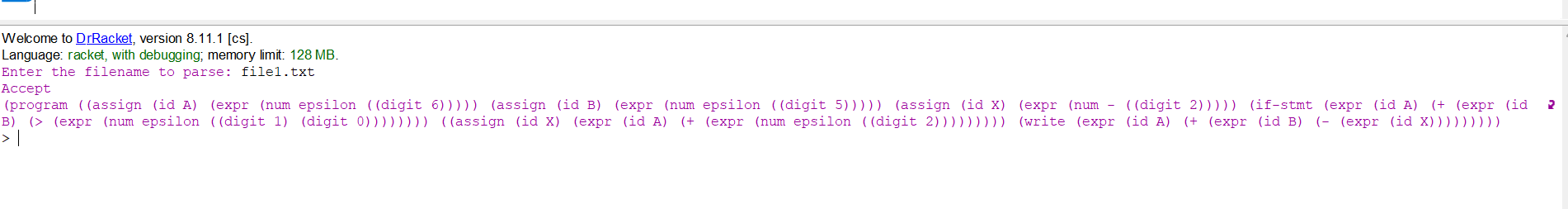
test5.txt | Accept 

It was at this point that I realized that my program wasn’t fully parsing expressions and I went to work on solving that issue. I quickly realized that the parse-expr wasn’t calling the helper functions made for parsing numbers. In addition, I added some bounds checking to make sure there is a digit after a numsign but that should be hopefully necessary as the Parser should default it back to a operator previously before arriving at this point in the code.

A screenshot of a computer program

AI-generated content may be incorrect.

With these changes made, the resulting tree breaks down expr into num, numsign, and digits. Unfortunately, the identifiers did not follow the proper Grammar and I had to follow a similar process as before and made a parse-id function to break down identifiers. This was straightforward though and I was easily able to get it to recognize ids. I wasn’t sure if I wanted to change “assign” to “stmt” as that was what was written in the grammar, but it helps me so I’ve decided to leave it.

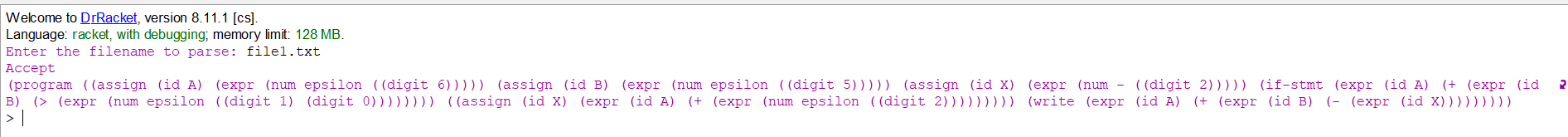


Now that everything is being properly identified in the parse tree, I wanted to clean up my code and make sure that all the helper functions are nested for better information hiding as noted in the assignment. I pasted my code into Claude and asked it to restructure the code for better security. It nested all the parsing subfunction into the parsing function. I

Overall, Claude was invaluable to me in this project. I was very overwhelmed by the project when it was just getting started as the sheer scope of the assignment felt so large when I would research the generators. Trying to find examples of what writing a parser would even look like was difficult beyond those in the text. Claude gave me a starting place of semi-workable code that I was able to modify and debug into something useable. I did find it poignant that the amount of time I needed to spend debugging tiny mistakes was able to be removed with money. There was still a fair amount of work that I had to manually do with what Claude provided, but I imagine if I was still using the free account, I would not have been as successful as I was. With tools like these at hand, I feel less intimidated by facing unknown tasks.

Below are my final sample file results:

File1.txt:



File2.txt:

A screenshot of a computer

AI-generated content may be incorrect.

File3.txt:

A screenshot of a computer

AI-generated content may be incorrect.

File4.txt:

A screenshot of a computer

AI-generated content may be incorrect.