**Clayton Heathscott**

**Dr. Vamsi Paruchuri**

**CSCI 3370 PRINCIPLES OF PROGRAMMING LANGUAGES**

**19 February 2020**

**Lab 2: Tokenizer**

**Question: Submit: Your code and a Word/PDF Report including screenshots of output, short description of your approach, challenges you faced and key take-aways**

When I began Lab 2, I made a list of tasks that I needed to complete in a specific order so I could maximize me productivity and learning. My first step was to learn how to read a file, while I was confused as to how I was supposed to go about this at first, it did not take long for me to have a program that split any file into a list of lines. My second step from there was to find a way to erase any comments found in the two jack files we were given. I made this my second step because I believed that this would take me the longest to complete, but later in this paper it will be made clear that I was wrong on that assumption.

I was able to erase all the comments and quickly moved to breaking up each line into 3 different subsets: numbers, strings, and symbols. From there I implemented if then statements that checked each one of these cases and I was able to correctly categorize everything in “main.jack” from there on while “squaregame.jack” gave me countless issues. One of the most prominent issues I had with this code was the issue of classifying string literals, it took me over 3 days to figure out how to differentiate between what was inside a quote and what was not, and also how to erase those words from my main printing list so they would not come back later printing as identifiers. Once I was able to do that, there was still one final issue. On “squaregame.jack,” there were two comments hidden and tagged onto the ends of two separate lines of code. I was unable to print the tokens for these lines until I went and sifted through the debugger hundreds of times, watching were each individual character was moving and how. Once I got that working, I had finally finished coding the program, and the output was just as I had wanted it to be. From this point I took each print and wrote it to a new file, ultimately finishing the project

My key take-aways from this project were as follows: labs such as these provide a perfect gateway into learning a new language, forcing myself to write code admittedly longer than it probably should have been made me even more comfortable with this language than c++. My next take-away was that the debugger is a friend, not a foe. When I was completely lost on this project, sitting down and sifting through line by line helped me understand what I was writing, and sped up the process greatly. My last takeaway was that I should not be so hard headed when I write code, sometimes it is necessary to scrap a function or two to make the code more efficient, but more often than not I would keep the larger block of code, and add more code on top of it to make it work the way I wanted it to. I enjoyed this lab a lot and I am looking forward to what other projects we have this semester.

**Main.jack output**

<Tokens>

<Keyword> class </Keyword>

<identifier> Main </identifier>

<Symbol> { </Symbol>

<Keyword> function </Keyword>

<Keyword> void </Keyword>

<identifier> main </identifier>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> { </Symbol>

<Keyword> var </Keyword>

<identifier> Array </identifier>

<identifier> a </identifier>

<Symbol> ; </Symbol>

<Keyword> var </Keyword>

<Keyword> int </Keyword>

<identifier> length </identifier>

<Symbol> ; </Symbol>

<Keyword> var </Keyword>

<Keyword> int </Keyword>

<identifier> i </identifier>

<identifier> sum </identifier>

<Symbol> , </Symbol>

<Symbol> ; </Symbol>

<StringConstant> HOW MANY NUMBERS? </StringConstant>

<Keyword> let </Keyword>

<identifier> length </identifier>

<identifier> Keyboard </identifier>

<identifier> readInt </identifier>

<Symbol> = </Symbol>

<Symbol> . </Symbol>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> ; </Symbol>

<Keyword> let </Keyword>

<identifier> a </identifier>

<identifier> Array </identifier>

<identifier> new </identifier>

<identifier> length </identifier>

<Symbol> = </Symbol>

<Symbol> . </Symbol>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> ; </Symbol>

<Keyword> let </Keyword>

<identifier> i </identifier>

<identifier> 0 </identifier>

<Symbol> = </Symbol>

<Symbol> ; </Symbol>

<integerConstant> 0 </integerConstant>

<Keyword> while </Keyword>

<identifier> i </identifier>

<identifier> length </identifier>

<Symbol> ( </Symbol>

<Symbol> < </Symbol>

<Symbol> ) </Symbol>

<Symbol> { </Symbol>

<StringConstant> ENTER THE NEXT NUMBER: </StringConstant>

<Keyword> let </Keyword>

<identifier> a </identifier>

<identifier> i </identifier>

<identifier> Keyboard </identifier>

<identifier> readInt </identifier>

<Symbol> [ </Symbol>

<Symbol> ] </Symbol>

<Symbol> = </Symbol>

<Symbol> . </Symbol>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> ; </Symbol>

<Keyword> let </Keyword>

<identifier> i </identifier>

<identifier> i </identifier>

<identifier> 1 </identifier>

<Symbol> = </Symbol>

<Symbol> + </Symbol>

<Symbol> ; </Symbol>

<integerConstant> 1 </integerConstant>

<Symbol> } </Symbol>

<Keyword> let </Keyword>

<identifier> i </identifier>

<identifier> 0 </identifier>

<Symbol> = </Symbol>

<Symbol> ; </Symbol>

<integerConstant> 0 </integerConstant>

<Keyword> let </Keyword>

<identifier> sum </identifier>

<identifier> 0 </identifier>

<Symbol> = </Symbol>

<Symbol> ; </Symbol>

<integerConstant> 0 </integerConstant>

<Keyword> while </Keyword>

<identifier> i </identifier>

<identifier> length </identifier>

<Symbol> ( </Symbol>

<Symbol> < </Symbol>

<Symbol> ) </Symbol>

<Symbol> { </Symbol>

<Keyword> let </Keyword>

<identifier> sum </identifier>

<identifier> sum </identifier>

<identifier> a </identifier>

<identifier> i </identifier>

<Symbol> = </Symbol>

<Symbol> + </Symbol>

<Symbol> [ </Symbol>

<Symbol> ] </Symbol>

<Symbol> ; </Symbol>

<Keyword> let </Keyword>

<identifier> i </identifier>

<identifier> i </identifier>

<identifier> 1 </identifier>

<Symbol> = </Symbol>

<Symbol> + </Symbol>

<Symbol> ; </Symbol>

<integerConstant> 1 </integerConstant>

<Symbol> } </Symbol>

<StringConstant> THE AVERAGE IS: </StringConstant>

<Keyword> do </Keyword>

<identifier> Output </identifier>

<identifier> printString </identifier>

<Symbol> . </Symbol>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> ; </Symbol>

<Keyword> do </Keyword>

<identifier> Output </identifier>

<identifier> printInt </identifier>

<identifier> sum </identifier>

<identifier> length </identifier>

<Symbol> . </Symbol>

<Symbol> ( </Symbol>

<Symbol> / </Symbol>

<Symbol> ) </Symbol>

<Symbol> ; </Symbol>

<Keyword> do </Keyword>

<identifier> Output </identifier>

<identifier> println </identifier>

<Symbol> . </Symbol>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> ; </Symbol>

<Keyword> return </Keyword>

<Symbol> ; </Symbol>

<Symbol> } </Symbol>

<Symbol> } </Symbol>

**SquareGame.jack**

<Tokens>

<Keyword> class </Keyword>

<identifier> SquareGame </identifier>

<Symbol> { </Symbol>

<identifier> field </identifier>

<identifier> Square </identifier>

<identifier> square </identifier>

<Symbol> ; </Symbol>

<identifier> field </identifier>

<Keyword> int </Keyword>

<identifier> direction </identifier>

<Symbol> ; </Symbol>

<identifier> constructor </identifier>

<identifier> SquareGame </identifier>

<identifier> new </identifier>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> { </Symbol>

<Keyword> let </Keyword>

<identifier> square </identifier>

<identifier> square </identifier>

<Symbol> = </Symbol>

<Symbol> ; </Symbol>

<Keyword> let </Keyword>

<identifier> direction </identifier>

<identifier> direction </identifier>

<Symbol> = </Symbol>

<Symbol> ; </Symbol>

<Keyword> return </Keyword>

<identifier> square </identifier>

<Symbol> ; </Symbol>

<Symbol> } </Symbol>

<identifier> method </identifier>

<Keyword> void </Keyword>

<identifier> dispose </identifier>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> { </Symbol>

<Keyword> do </Keyword>

<identifier> square </identifier>

<identifier> dispose </identifier>

<Symbol> . </Symbol>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> ; </Symbol>

<Keyword> do </Keyword>

<identifier> Memory </identifier>

<identifier> deAlloc </identifier>

<identifier> square </identifier>

<Symbol> . </Symbol>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> ; </Symbol>

<Keyword> return </Keyword>

<Symbol> ; </Symbol>

<Symbol> } </Symbol>

<identifier> method </identifier>

<Keyword> void </Keyword>

<identifier> run </identifier>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> { </Symbol>

<Keyword> var </Keyword>

<identifier> char </identifier>

<identifier> key </identifier>

<Symbol> ; </Symbol>

<Keyword> var </Keyword>

<identifier> boolean </identifier>

<identifier> exit </identifier>

<Symbol> ; </Symbol>

<Keyword> let </Keyword>

<identifier> exit </identifier>

<identifier> key </identifier>

<Symbol> = </Symbol>

<Symbol> ; </Symbol>

<Keyword> while </Keyword>

<identifier> exit </identifier>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> { </Symbol>

<Keyword> while </Keyword>

<identifier> key </identifier>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> { </Symbol>

<Keyword> let </Keyword>

<identifier> key </identifier>

<identifier> key </identifier>

<Symbol> = </Symbol>

<Symbol> ; </Symbol>

<Keyword> do </Keyword>

<identifier> moveSquare </identifier>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> ; </Symbol>

<Symbol> } </Symbol>

<identifier> if </identifier>

<identifier> key </identifier>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> { </Symbol>

<Keyword> let </Keyword>

<identifier> exit </identifier>

<identifier> exit </identifier>

<Symbol> = </Symbol>

<Symbol> ; </Symbol>

<Symbol> } </Symbol>

<identifier> if </identifier>

<identifier> key </identifier>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> { </Symbol>

<Keyword> do </Keyword>

<identifier> square </identifier>

<identifier> decSize </identifier>

<Symbol> . </Symbol>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> ; </Symbol>

<Symbol> } </Symbol>

<identifier> if </identifier>

<identifier> key </identifier>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> { </Symbol>

<Keyword> do </Keyword>

<identifier> square </identifier>

<identifier> incSize </identifier>

<Symbol> . </Symbol>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> ; </Symbol>

<Symbol> } </Symbol>

<identifier> if </identifier>

<identifier> key </identifier>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> { </Symbol>

<Keyword> let </Keyword>

<identifier> direction </identifier>

<identifier> exit </identifier>

<Symbol> = </Symbol>

<Symbol> ; </Symbol>

<Symbol> } </Symbol>

<identifier> if </identifier>

<identifier> key </identifier>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> { </Symbol>

<Keyword> let </Keyword>

<identifier> direction </identifier>

<identifier> key </identifier>

<Symbol> = </Symbol>

<Symbol> ; </Symbol>

<Symbol> } </Symbol>

<identifier> if </identifier>

<identifier> key </identifier>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> { </Symbol>

<Keyword> let </Keyword>

<identifier> direction </identifier>

<identifier> square </identifier>

<Symbol> = </Symbol>

<Symbol> ; </Symbol>

<Symbol> } </Symbol>

<identifier> if </identifier>

<identifier> key </identifier>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> { </Symbol>

<Keyword> let </Keyword>

<identifier> direction </identifier>

<identifier> direction </identifier>

<Symbol> = </Symbol>

<Symbol> ; </Symbol>

<Symbol> } </Symbol>

<Keyword> while </Keyword>

<identifier> key </identifier>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> { </Symbol>

<Keyword> let </Keyword>

<identifier> key </identifier>

<identifier> key </identifier>

<Symbol> = </Symbol>

<Symbol> ; </Symbol>

<Keyword> do </Keyword>

<identifier> moveSquare </identifier>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> ; </Symbol>

<Symbol> } </Symbol>

<Symbol> } </Symbol>

<Keyword> return </Keyword>

<Symbol> ; </Symbol>

<Symbol> } </Symbol>

<identifier> method </identifier>

<Keyword> void </Keyword>

<identifier> moveSquare </identifier>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> { </Symbol>

<identifier> if </identifier>

<identifier> direction </identifier>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> { </Symbol>

<Keyword> do </Keyword>

<identifier> square </identifier>

<identifier> moveUp </identifier>

<Symbol> . </Symbol>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> ; </Symbol>

<Symbol> } </Symbol>

<identifier> if </identifier>

<identifier> direction </identifier>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> { </Symbol>

<Keyword> do </Keyword>

<identifier> square </identifier>

<identifier> moveDown </identifier>

<Symbol> . </Symbol>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> ; </Symbol>

<Symbol> } </Symbol>

<identifier> if </identifier>

<identifier> direction </identifier>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> { </Symbol>

<Keyword> do </Keyword>

<identifier> square </identifier>

<identifier> moveLeft </identifier>

<Symbol> . </Symbol>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> ; </Symbol>

<Symbol> } </Symbol>

<identifier> if </identifier>

<identifier> direction </identifier>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> { </Symbol>

<Keyword> do </Keyword>

<identifier> square </identifier>

<identifier> moveRight </identifier>

<Symbol> . </Symbol>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> ; </Symbol>

<Symbol> } </Symbol>

<Keyword> do </Keyword>

<identifier> Sys </identifier>

<identifier> wait </identifier>

<identifier> direction </identifier>

<Symbol> . </Symbol>

<Symbol> ( </Symbol>

<Symbol> ) </Symbol>

<Symbol> ; </Symbol>

<Keyword> return </Keyword>

<Symbol> ; </Symbol>

<Symbol> } </Symbol>

<Symbol> } </Symbol>