The data structures to be used in the parser:

Arrays to store the inputs into then checking it to determine what everything needs to be outputted also used as a Queue to hold the input from the scanner to later be able to determine the parse tree and then output it from the scanner.

Important Ideas:

At the beginning of each function for a non-terminal symbol, print to a string buffer a new line containing [indent]. Before ending the function, print to the buffer a new line of [indent]. The indent is the spaces one should keep before the Non-Terminal symbol. When a token is recognized you will print to the buffer [indent][Token]

Pseudo-code:

Idea:

Int counter = 0

String Parser(input)

Println (“<program>”)

Call StatementList(input)

Print (“</Program>”)

Return $$

StatementList(input)

If input\_size == counter,

If the end of file then return and exit

Return

Statement(input)

StatementList(input)

Println (“</ stmt list>”)

Return

Statement(input)

Try

Convert the input, input.get(counter)

Change to string, string. valueOf(input.get(counter))

Check if the string is an int, integer. parseInt (string. valueOf(input.get(counter)))

Catch(numberformatexception)

If here then it is not a number so call it read and return

Println (“ Read” +input.get(counter))

Increment counter, counter ++

Println (“</stmt>”)

If here then it is a number, so don’t go to next slot since its done later

Expr(input)

Expr(list input)

Println (“<expr>”)

Call term and term tail

Term(input)

Term\_tail(input)

Println (“</expr>”)

Term\_tail(input)

If (input size < counter)

Try

Convert the input, input.get(counter)

Change to string, string. valueOf(input.get(counter))

Check if the string is an int, integer. parseInt (string. valueOf(input.get(counter)))

Catch

If its here then not a number.

Print that the term tail is null

If it is equal to either + or – operations

Println (“<term tail>”)

Add\_op(input)

Term(input)

Term\_tail(input)

Println (“</term tail>”)

Term(input)

Println (“<term>”)

Call factor and factor tail to check

Println (“</term>”)

Fact\_tail(input)

If (input size < counter)

Try

Convert the input, input.get(counter)

Change to string, string. valueOf(input.get(counter))

Check if the string is an int, integer. parseInt (string. valueOf(input.get(counter)))

Catch

If its here then not a number.

Return and end branch

If it is equal to either \* or /

Println (“<fact tail>”)

Mult\_op(input)

Factor(input)

Fact\_tail(input )

Println(“</fact tail>”)

If not print null inbetween fact tail

Factor(input)

Try

Convert the input, input.get(counter)

Change to string, string. valueOf(input.get(counter))

Check if the string is an int, integer. parseInt (string. valueOf(input.get(counter)))

Catch

If it’s here then not a number.

Call it read and return.

If (parseint < counter)

If it is either operation +, -, \*, /

Println (“<factor>”)

Println (“ “ +input.get(counter))

Inc counter, Counter++

Expr(input)

Prntln(input.get(counter))

Prntln(“</factor>”)

return

Else

Println (“<factor>”)

Println (“ “ +input.get(counter))

Prntln(“</factor>”)

Inc counter, Counter++

Return

Println (“<factor>”)

Println (“ “ +input.get(counter))

Prntln(“</factor>”)

Inc counter, Counter++

Return

Add\_op(input)

Prntln(“<add\_op>”)

Check if plus or minus

If (string == +)

Println(“+”)

Else if (string == -)

Println(“-“)

Else

Println(error mult\_op)

Exit

Counter++

Println(“</add\_op>”)

Mult\_op(input)

Println(“<mult op>”)

Check if multiplication or division

If (string == \*)

Println (“\*”)

Else if (string == /)

Println (“/”)

Else

Println(“error mult\_op”)

Exit

Counter ++

Println(“ </mult op>”)

Return

Test cases :

Read A

Check 8 \* rand

Read B

Empty test case

Followed by another to see if there is issue

Next

Contributors:

Jonathan Turner: Pseudo Code outline, data structure explanation, parser class coding

Jose Trevino: Pseudo code, test case selection and document outline