The intent of this language is to include all strings that accurately describe basic arithmetic expressions. For the structure of this language, we utilized a Pushdown Automata to be able to accept arithmetic equations that fit into the grammar.

Our PDA starts with q0, where it looks to see if the expression begins with a left parenthesis, a negative sign, or neither of those. If it starts with a left parenthesis then it will add an X to the stack, which is used to keep track. Then it will transition to q1 where then it will either check for more left parenthesis (again adding an X to the stack for every one it finds), check for more digits. Once the digit ends and it starts reading an operator, it branches off into two paths, one which is for addition (+), subtraction (-), and multiplication (x). The other path is explicitly only division, since division has more qualifiers to be valid. If it finds a plus, minus, or multiplication symbol, it will transition to q3.

Q3 then has the ability to once again check for more left parenthesis for further grouping support. Q3 then will transition to q4 ensuring that at least one number occurs after the operator (which also can be negative). If however back at q1 a division operator had been found instead, q1 would have transitioned to q2. Q2's purpose is basically to ensure that you cannot divide by 0, but otherwise works the same as the other path.

Once at q4, You can once again loop as many times as need be when reading more numbers or parenthesis. However there are new variables here which ensure two things. B ensures that in the scenario the user uses two minus signs back to back, that the 2nd one is then interpreted like a negative number, and thus needing at least one digit to proceed. C's purpose is to catch if we are dividing by 0 if we are using a negative number (the C in this scenario would have been pushed to the stack from the q2 to q4 transition). Thus if C is on the stack then only digits 1 through 9 can be read.

Q4 then will transition to Q5 to pop the A from the stack. It will then check for a right parenthesis, and if it's found there has to be an X on the top of the stack to ensure parenthesis balance. From Q5 there are 3 paths. Two of the paths are if there is another operator present , and if there is, it will transition to the necessary node to then go through the loop again (if a division sign is found it will transition to q2, and if the other operators are found it will transition to q3). However if we reach the end of the equation, then the program will transition to q6 which acts as our accepting state ending the program.

The purpose of this program would be for something like a calculator which needs to be able to determine what user inputs are allowed, since equations have to follow a specific syntax to be calculable. For that purpose this also could be heavily expanded as higher level math gets further incorporated needing more specific rules and flow starts.