Homework 1 - LL(1) Calculator Parser - Translator to Java

Part 1

1.

Η LL(1) γραμματική που δημιούργησα είναι:

syn → exp

exp → term exp2

exp2 → + term exp2
| - term exp2
| ε

term → factor term2

term2 → * factor term2
| / factor term2
| ε

factor → num
| (exp)

num → digit multiple

multiple → num
| ε

digit → 0
| 1
| 2
| 3
| 4
| 5
| 6
| 7
| 8
| 9

FIRST sets:

FIRST(syn) = FIRST(exp) = FIRST(term) = FIRST(factor) = { (} U FIRST(num) = { (} U FIRST(digit) = { (, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

FIRST(exp2) = { +, - , ε }

FIRST(term2) = { *, /, ε }

FIRST(multiple) = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ε }

FOLLOW sets:

FOLLOW(syn) = { EOF,) }

FOLLOW(exp) = FOLLOW(syn) = { EOF,) }

FOLLOW(exp2) = FOLLOW(exp) = { EOF,) }

FOLLOW(term) += FIRST(exp2) += { +, - , ε } += { +, - , FOLLOW(exp) } += { +, -, EOF,) }

FOLLOW(term2) = FOLLOW(term) = { +, -, EOF,) }

FOLLOW(factor) += FIRST(term2) += { *, /, ε } += { *, /, FOLLOW(term) } += { *, /, +, -, EOF,) }

FOLLOW(num) = FOLLOW(factor) = { *, /, +, -, EOF,) }

FOLLOW(multiple) = FOLLOW(num) = { *, /, +, -, EOF,) }

FOLLOW(digit) += FIRST(multiple) += { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ε } += { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, FOLLOW(multiple) } += { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *, /, +, -, EOF,) }

FIRST+ sets:

FIRST+(syn) = FIRST+(exp) = FIRST+(term) = FIRST+(factor) = { (, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

FIRST+(exp2) = { +, - , EOF,), ε }

FIRST+(term2) = { *, /, +, -, EOF,), ε }

FIRST+(multiple) = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *, /, +, -, EOF,), ε }

LOOKAHEAD TABLE:

	+, -	*, /	(	digit
exp2	term exp2	error	error	error
term2	ε	factor term2	error	error
factor	error	error	(do nothing)	(do nothing)
num	error	error	error	multiple

2. Το compile γίνεται με: make compile

Η εκτέλεση με: make execute και εισαγωγή στο terminal

Γενικά κράτησα την δομή των παραδειγμάτων, οπότε υπάρχουν τα αρχεία ParseError.java, Main.java και το Calculator.java μέσα στο οποίο γίνεται όλη η δουλειά.

Οι συναρτήσεις που υλοποιούν το calculator ακολουθούν την λογική της LL(1) γραμματικής που βρίσκεται από πάνω.

Part 2

Ο scanner εκτός από string literals, αναγνωρίζει και επιστρέφει και identifiers. Στην περίπτωση των string literals έχω αλλάξει λίγο τον κώδικα έτσι ώστε να επιστρέφει και τα εισαγωγικά (“) στην αρχή και στο τέλος του string, για να κάνω λίγο πιο εύκολη την παραγωγή του κώδικα στο επόμενο βήμα.

Στον parser τα non terminals def_args και rest_args είναι αυτά που αναγνωρίζουν τα ορίσματα που βρίσκονται στον ορισμό μίας συνάρτησης και δέχονται μόνο identifier. Αντίστοιχα στα ορίσματα στην κλήση μιας συνάρτησης μέσα στην main τα call_args και rest_call δέχονται μόνο call_stm που είναι τα πάντα εκτός από identifier, ενώ μέσα στο σώμα στον ορισμό μίας συνάρτησης δέχεται και identifier.

Στο Makefile σύμφωνα με μία ερώτηση στο Piazza για διευκόλυνση έβαλα τα path των jar να είναι ένα επίπεδο έξω από τον φάκελο του παραδοτέου μου.