ΡΑΦΑΗΛ-ΧΡΗΣΤΟΣ ΤΑΣΙΟΥΛΑΣ

115201800191

Μεταγλωττιστές

Εργασία 1

Στον φάκελο της εργασίας υπάρχουν 2 directories, ένα για κάθε μέρος της εργασίας όπως και τα jar αρχεία για να τρέξουν τα προγράμματα στο 2^{ο} μέρος.

Μέρος 1^{ο}

Στο directory αυτό υπάρχουν 4 αρχεία, το test-input, ένα README και δύο Java αρχεία συγκεκριμένα τα Calculator.Java και ParseError.Java.

Το test-input έχει ενδεικτικό input για το Calculator της πρώτης εργασίας.

Το README έχει εντολές για μεταγλώττιση των java αρχείων αλλά και για την εκτέλεσή τους. Επίσης υπάρχει εντολή για αφαίρεση των εκτελέσιμων. Πέρα από εντολές περιλαμβάνει συνοπτική περιγραφή της γραμματικής που χρησιμοποιήθηκε για το κομπιουτεράκι της εργασίας μαζί με ένα lookahead table. Η γραμματική που χρησιμοποιήθηκε είναι η εξής:

Goal => Expression

Expression => Xor_Term Expression2

Expression` => ^ Xor_Term Expression`

Expression` => ε

Xor_Term => And_Term Expression``

Expression`` => & And_Term Expression``

Expression`` => ε

And_Term => (Expression)

And_Term => Number

Όπου τα υπογραμμισμένα είναι τερματικά σύμβολα. Στη γραμματική αυτή δηλώνεται η προτεραιότητα των πράξεων ως εξής:

Προτεραιότητα(^)<Προτεραιότητα(&)<Προτεραιότητα(())

Για την κατανόηση της γραμματικής υπάρχει και το lookahead table το οποίο παρουσιάζεται ως εξής:

	$	^	&	(	)	Number
Goal(G)				E $		E $
Expression(E)				X E`		X E
Expression`(E`)	ε	^ X E`			ε	
Xor_Term(X)				A E``		A E
Expression``(E``)	ε	ε	& A E``		ε	
And_Term(A)				(E)		Number

Όπου τα υπογραμμισμένα είναι τερματικά σύμβολα.

Τα δύο Java αρχεία στο directory είναι τα Calculator.java και ParseError.java.

Το τελευταίο, είναι το αρχείο που χρησιμοποιείται στον κώδικα του φροντιστηρίου.

Το Calculator.java είναι το αρχείο στο οποίο τρέχει ο κώδικας. Αποτελείται από δύο αντικείμενα, το Calculator και το Expression.

Το Calculator, στην ουσία, έχει την main η οποία διαβάζει το δοσμένο από την εντολή εκτέλεσης input file και μετατρέπει κάθε γραμμή του σε Expression. Αφού ελέγξει κάθε Expression για parse error είτε πετάει exception καλώντας το άλλο java αρχείο σε περίπτωση parse error είτε εμφανίζει το εκτιμώμενο αποτέλεσμα το οποίο υπολογίζεται από μεθόδους που θα αναλυθούν παρακάτω.

Το Expression είναι ένα αντικείμενο το οποίο περιγράφει αριθμητικές εκφράσεις. Περιλαμβάνει την ίδια την έκφραση σε μορφή συμβολοσειράς, έναν αριθμό να κρατάει τη θέση του τελευταίου χαρακτήρα που εξετάζεται (current_index), τον ίδιο τον τελευταίο χαρακτήρα που εξετάζεται σε μορφή συμβολοσειράς (current_character), μία στοίβα που περιλαμβάνει όλους τους χαρακτήρες των οποίων θα εκτιμηθεί το αποτέλεσμα αυτή τη στιγμή και το αποτέλεσμα που έχει υπολογιστεί από τις πράξεις που έγιναν στην έκφραση (current_result).

Στον constructor της κλάσης αποθηκεύεται η δοσμένη έκφραση, δημιουργείται η στοίβα, αρχικοποιείται το current_index με 0 και διαβάζεται ο πρώτος χαρακτήρας της έκφρασης.

Η get_evaluated_result είναι μη στατική μέθοδος η οποία επιστρέφει το αποτέλεσμα που έχει αποθηκευτεί για την έκφραση.

Η μέθοδος advance αυξάνει το current_index κατά ένα και διαβάζει τον επόμενο χαρακτήρα.

Η μέθοδος read_character διαβάζει τον χαρακτήρα που βρίσκεται στη θέση current_index της έκφρασης αποθηκεύεται ως current_character και εκτυπώνεται αν το

current_index είναι μικρότερο από το μήκος της έκφρασης. Διαφορετικά αποθηκεύεται ο χαρακτήρας $ για τη δήλωση του EOF χωρίς να εκτυπώνεται.

Η μέθοδος evaluate βγάζει από την στοίβα τους 3 πρώτους της χαρακτήρες . Ο πρώτος και ο τρίτος αντιμετωπίζονται ως ψηφία. Ο δεύτερος ως operator. Πρώτα μετατρέπει σε ακέραιους τους 2 χαρακτήρες και ελέγχει αν ο operator ισούται με & η ^ και κάνει την αντίστοιχη πράξη. Διαφορετικά πετάει parse error. Το αποτέλεσμα της πράξης μπαίνει στην στοίβα και αποθηκεύεται στο current_result. Στην περίπτωση όπου υπάρχει μόνο ένας αριθμός αντιμετωπίζεται αυτός ως το αποτέλεσμα της συνάρτησης.

Η μέθοδος parse καλείται από το calculator για να ελεγχθεί και να εκτιμηθεί το αποτέλεσμα της έκφρασης από την οποία κλήθηκε. Καλεί την μέθοδο expr.

Η μέθοδος expr, στην ουσία, επεξεργάζεται τον current_character ως Expression(E), σύμφωνα με την γραμματική πιο πάνω. Υπολογίζει το σύνολο FIRST+(E) (= FIRST+(X) = {(, [0-9]}). Εάν ο current_character δεν ανήκει σε αυτό το σύνολο πετάει parse error, διαφορετικά καλεί τις επόμενες μεθόδους στη γραμματική term και expr0 με αυτή τη σειρά.

Η μέθοδος term, στην ουσία, επεξεργάζεται τον current_character ως Xor_Term(X)), σύμφωνα με την γραμματική πιο πάνω. Υπολογίζει το σύνολο FIRST+(X) (= FIRST+(A) = {(, [0-9]}). Εάν ο current_character δεν ανήκει σε αυτό το σύνολο πετάει parse error, διαφορετικά καλεί τις επόμενες μεθόδους στη γραμματική term0 και expr1 με αυτή τη σειρά.

Η μέθοδος expr0, στην ουσία, επεξεργάζεται τον current_character ως Expression`(E`), σύμφωνα με την γραμματική πιο πάνω. Υπολογίζει το σύνολο FIRST+(E`) = {^,), ε} . Εάν ο current_character δεν ανήκει σε αυτό το σύνολο πετάει parse error, διαφορετικά εφόσον ο current_character είναι ο ^, βάζει τον operator ^ στη στοίβα, καλεί το διάβασμα στον επόμενο χαρακτήρα, καλεί τις επόμενες μεθόδους στη γραμματική term και expr0 με αυτή τη σειρά και μετά καλεί την evaluate ώστε να υπολογίσει το αποτέλεσμα της XOR που προκύπτει από τις επόμενες αναδρομές.

Η μέθοδος expr1, στην ουσία, επεξεργάζεται τον current_character ως Expression``(E``), σύμφωνα με την γραμματική πιο πάνω. Υπολογίζει το σύνολο FIRST+(E``) = {&, ^,), ε}. Εάν ο current_character δεν ανήκει σε αυτό το σύνολο πετάει parse error, διαφορετικά εφόσον ο current_character είναι ο &, βάζει τον operator & στη στοίβα, καλεί το διάβασμα στον επόμενο χαρακτήρα, καλεί τις επόμενες μεθόδους στη γραμματική term0 και expr1 με αυτή τη σειρά και μετά καλεί την evaluate ώστε να υπολογίσει το αποτέλεσμα της AND που προκύπτει από τις επόμενες αναδρομές.

Η μέθοδος term, στην ουσία, επεξεργάζεται τον current_character ως And_Term(A), σύμφωνα με την γραμματική πιο πάνω. Υπολογίζει το σύνολο FIRST+(A) = {(, [0-9]}. Εάν ο current_character δεν ανήκει σε αυτό το σύνολο πετάει parse error, διαφορετικά εάν είναι η (, τότε καλεί το διάβασμα στον επόμενο χαρακτήρα και μετά την expr(). Μετά το τέλος της expr() αν ο current_character δεν ισούται με), θα ρίχνει parse error, αλλιώς θα εκτιμά το αποτέλεσμα της έκφρασης μέσα στην παρένθεση και θα διαβάζει τον επόμενο χαρακτήρα. Τέλος, εάν ο current character είναι ίσος με μονοψήφιο αριθμό, ο αριθμός θα μπαίνει στην στοίβα ώστε να υπολογιστεί το αποτέλεσμά του από τις πράξεις της έκφρασης και μετά καλείται το διάβασμα στον επόμενο χαρακτήρα.

Η μέθοδος First_plus επιστρέφει το σύνολο συμβολοσειρών (Set<String>) FIRST+ ενός δοσμένου non terminal σε μορφή string. Πρώτα αποθηκεύεται το σύνολο FIRST του δοσμένου non terminal. Εάν στο FIRST εμπεριέχεται το κενό, η μέθοδος επιστρέφει την ένωση των συνόλων FIRST και FOLLOW του non terminal διαφορετικά επιστρέφεται μόνο το FIRST.

Η μέθοδος Follow επιστρέφει το σύνολο συμβολοσειρών (Set<String>) FOLLOW ενός δοσμένου non terminal σε μορφή string. Προσθέτει πάντα το $ ως EOF και μετά παίρνει περιπτώσεις ανάλογα το non terminal που θα δοθεί με βάση την προαναφερθείσα γραμματική.

Στην περίπτωση που έχουμε το expr(:E), προσθέτει την) [επειδή A => (E)] στο σύνολο κι έτσι επιστρέφει FOLLOW(E) = {), $}.

Στην περίπτωση που έχουμε το term(:X), υπάρχουν δύο τύποι προτάσεων [E => X E`, E` => ^ X E`]. Από αυτές τις δύο προτάσεις συμπεραίνουμε ότι το FOLLOW(X) = (FIRST(E`) U FOLLOW(E`)) – {ε}, το ε ανήκει στο FIRST(E`).

Στην περίπτωση που έχουμε το expr0(:E`), επειδή E => X E`, ισχύει FOLLOW(E`) = FOLLOW(E).

Στην περίπτωση που έχουμε το term0(:A), υπάρχουν δύο τύποι προτάσεων [E` => A E``, E` => & A E``]. Από αυτές τις δύο προτάσεις συμπεραίνουμε ότι το FOLLOW(A) = (FIRST(E``) U FOLLOW(E``)) – {ε}, το ε ανήκει στο FIRST(E``).

Στην περίπτωση που έχουμε το expr1(:E``), επειδή X => A E``, ισχύει FOLLOW(E``) = FOLLOW(X).

Τέλος, η μέθοδος First επιστρέφει το σύνολο συμβολοσειρών (Set<String>) FIRST ενός δοσμένου non terminal σε μορφή string. Παίρνει περιπτώσεις ανάλογα το non terminal που θα δοθεί με βάση την προαναφερθείσα γραμματική.

Στην περίπτωση που έχουμε το expr(:E), επειδή E => X E`, επιστρέφει FIRST(E) = FIRST(X).

Στην περίπτωση που έχουμε το term(:X), επειδή X => A E``, επιστρέφει FIRST(X) = FIRST(A).

Στην περίπτωση που έχουμε το expr0(:E`), επειδή E` => ^ X E` | ε, επιστρέφει FIRST(E`) = {^, ε}.

Στην περίπτωση που έχουμε το term0(:A), επειδή A => (E) | Number , επιστρέφει FIRST(A) = {(,[0-9]}.

Στην περίπτωση που έχουμε το expr1(:E`), επειδή E`` => ^ A E`` | ε, επιστρέφει FIRST(E``) = {&, ε}.

Τα γράμματα A, E, E`, E``, X αντιστοιχούν σε κάθε non terminal όπως ορίστηκαν στο lookahead table πιο πάνω.

Μέρος 2ο

Στο directory αυτό υπάρχουν 7 αρχεία, τα input_example1-3, ένα Makefile ένα Java αρχείο, το scanner.flex και το parser.cup.

Τα 3 input_example αρχεία είναι αρχεία με τα κείμενα που δόθηκαν στην εκφώνηση της εργασίας.

Το Makefile περιλαμβάνει εντολές για την μεταγλώττιση των αρχείων και για την εκτέλεσή τους με κάθε input file ξεχωριστά. Τα αρχεία μεταγλωττίζονται με εντολές make all και make compile, εκτελούνται με τις make execute1/2/3 ανάλογα ποιο input θέλουμε και διαγράφει τα αρχεία που παράχθηκαν από τη μεταγλώττιση με την make clean.

Το Java αρχείο είναι το Parsexe.java. Με την main δημιουργεί inputstream από το αρχείο που δόθηκε και καλεί την parse, μέθοδο που ξεκινά το parsing του αρχείου που δόθηκε.

Το scanner.flex περιλαμβάνει ορολογίες για όλα τα tokens που χρησιμοποιούνται στο parser. Τα γράμματα ορίζονται ως κεφαλαία, μικρά και η κάτω παύλα. Ένα identifier είναι το kleene star των γραμμάτων, δηλαδή όλοι οι πιθανοί συνδυασμοί μεταξύ των γραμμάτων. Ορίζεται ενιαίο token) {, δηλαδή η δεξιά παρένθεση μαζί με κενό (WhiteSpace) και αριστερό άγκιστρο ως RightLeft(RL). Οι λέξεις if, else, reverse και prefix ορίζονται ως δεσμευμένες και μάλιστα οι 3 τελευταίες ακολουθούνται από WhiteSpace. Τα brackets, οι παρενθέσεις, ξεχωριστά μεταξύ τους και τα δύο, το κόμμα και το + ορίζονται ως tokens όπως και οι σταθερές συμβολοσειρές.

Το parser.cup περιλαμβάνει ορολογίες για terminals, non terminals και τη γραμματική μεταξύ τους. Ορίζονται επίσης συναρτήσεις όπως η print που καλεί την System.out.println για συμβολοσιερές, η is_blank_parameter που εξετάζει αν το parameter που δόθηκε σε μορφή συμβολοσειράς είναι ίσο με το κενό ή όχι. Αν δεν είναι κενό, επιστρέφει την συμβολοσειρά με τον τύπο String ως prefix διαφορετικά δεν αλλάζει τίποτα. Τέλος υπάρχει η make_condition, που με ορίσματα τα non terminals στο condition επιστρέφει ένα if else statement σε μορφή συμβολοσειράς.

Τα terminals που ορίζονται είναι τα ίδια που συζητήθηκαν παραπάνω. Ως Strings ορίστηκαν τα IDENTIFIER και STRING_LITERAL. Τα non terminals ορίστηκαν όλα ως String εκτός του program(PR) που ξεκινάει τη γραμματική και εκτυπώνει τα απαραίτητα κομμάτια του output. Τα string non terminals που ορίστηκαν είναι τα εξής:

expr_list(EL), functions(F), function_dec(FD), codeblock(CB), condition(C), function_call(FC), expr(E), arguments(A), parameters(P), might_blank_expression(MBE), str(S).

Το token PLUS ορίζεται με αριστερή προτεραιότητα και η γραμματική ξεκινάει ως εξής:

P => F EL

Όπου καλείται η print για κάθε γραμμή που ορίζει την κλάση main, τη συνάρτηση main, τα άγκιστρα που κλείνουν κλάση και συνάρτηση main όπως και για τα περιεχόμενα της

main τα οποία αντιπροσωπεύονται από το EL και τα περιεχόμενα των άλλων συναρτήσεων τα οποία αντιπροσωπεύονται από το F. Κάθε γραμμή κώδικα μέσα σε συνάρτηση ξεκινάει από δύο tabs. Τα non terminals ορίζονται ως εξής.

EL => EL E | E

F => F FD | ε

Ορίζονται αναδρομικά. Κάθε μη κενή E ορίζεται ανάμεσα στις παρενθέσεις τις συμβολοσειράς «System.out.println();».

Τα FD μπορεί να μην υπάρχουν καν στο πρόγραμμα και πάντα είναι μια γραμμή κάτω από τον προηγούμενο κώδικα. Τα FC, FD ορίζονται:

FC => IDENTIFIER (A)

FD => IDENTIFIER (P) { CB }

Το όνομα της συνάρτησης είναι identifier και ανάλογα αν είναι call η declaration ανάμεσα στις παρενθέσεις δέχεται A η P. Στο declaration, για να αποφευχθούν τα conflicts τα RPAREN και LBRACKET είναι το ενιαίο token RL. Μετά το LBRACKET υπάρχει το μπλοκ κώδικα της συνάρτησης και κλείνει με RBRACKET.

Ορίζονται επίσης τα:

A => MBE | MBE , A

P => MBE | MBE , P

Ορίζονται αναδρομικά, κάθε ένα χωρίζεται με κόμμα από το επόμενο. Μπορεί να είναι κενά ή expressions. Μάλιστα κάθε parameter ελέγχεται αν είναι κενό ή όχι με την κλήση της προαναφερθείσας is_blank_parameter για την εκτύπωσή του.

Ορίζεται επίσης:

CB => C | E

Το οποίο μπορεί να είναι ή υπόθεση ή μη κενή έκφραση. Κάθε μη κενή έκφραση συνοδεύεται από τη συμβολοσειρά “return “ και τερματίζει με ;.

Η υπόθεση ορίζεται ως:

C => IF (E PREFIX E) CB ELSE CB

Καλύπτεται αναδρομικά με το CB, και τονίζει την ύπαρξη else με κάθε if. Καλείται η make_condition που αναφέρθηκε παραπάνω για την επιστροφή της συμβολοσειράς υπόθεσης με ορίσματα (E, E, CB, CB). Μάλιστα τονίζει ότι η πράξη prefix μη κενών εκφράσεων υπάρχει μόνο σε if καθώς είναι η μόνη Boolean πράξη ανάμεσα σε 2 String πράξεις. Όπως θα εξηγήσουμε παρακάτω τονίζεται η μικρότερη προτεραιότητα της if από τις άλλες πράξεις.

Ορίζονται:

MBE => E | ε

E => E + E | S

Το MBE υπάρχει για την αποφυγή conflicts μεταξύ προτάσεων που χρειάζονται μη κενές εκφράσεις και αυτών που χρειάζονται και κενές εκφράσεις. Έτσι διευκολύνεται η δημιουργία καταστάσεων που δεν προκύπτουν κενές εκφράσεις σε σημεία που δεν πρέπει να είναι.

Το E είτε είναι string σταθερό ή μεταβλητό, είτε συνένωση (concat) δύο μη κενών εκφράσεων. Τονίζεται έτσι και η προτεραιότητα της πράξης από την if και όπως θα δούμε παρακάτω και η προτεραιότητα της reverse από την concat.

Τέλος, το S ορίζεται ως:

S => REVERSE S | FC | STRING_LITERAL | IDENTIFIER

Το S είναι είτε function call που είδαμε παραπάνω είτε το reverse ενός άλλου S είτε σταθερά ή μεταβλητή τύπου String. Τονίζεται έτσι ότι το reverse έχει τη δυνατότερη προτεραιότητα από όλες τις πράξεις στη γραμματική διότι βρίσκεται στη βάση της, το S. Η προτεραιότητα πράξεων που ακολουθήθηκε είναι η εξής:

IF PREFIX < + < REVERSE

Όλα τα υπογραμμισμένα είναι τερματικά tokens.