

Μεταγλωττιστές 2018

Προγραμματιστική Εργασία #2

Ονοματεπώνυμο: Αναστασία Τζώρα

ΑΜ: Π2015196

1. Κανόνες γραμματικής

<Program> -> Stmt_list #
Stmt_list -> Stmt Stmt_list | ε
Stmt -> V = Expr | print Expr
Expr -> Term Term_tail
Term_tail -> AndOrop Term Term_tail | Boolean | ε
Term -> Factor Factor_tail
Factor_tail -> Factor Factor_tail | ε
Factor -> (Expr) | V | Notop V
Notop -> not
AndOrop -> and | or
Boolean -> true|false|t|f|0|1

V : όνομα μεταβλητής αποτελούμενο αποκλειστικά από γράμματα

2. Έλεγχος συμβατότητας

Αρχικά πραγματοποιήσαμε έλεγχο συμβατότητας με το πρόγραμμα που μας δόθηκε στις οδηγίες του παραδοτέου. Οι κανόνες έχουν ως εξής:

Stmt_list -> Stmt Stmt_list

|.

Stmt -> V assign Expr

| print Expr.

Expr -> Term Term_tail.

Term_tail -> AndOrop Factor Factor_tail

| Boolean

|.

Factor_tail -> Factor Factor_tail.

Factor -> (Expr)

| V

| Notop V.

Boolean -> 0

| 1

| true

| false

| t

| f.

Notop -> not.

AndOrop -> or

|and.

Τα αποτελέσματα έχουν ως εξής:

- You have unrealizable nonterminals in your grammar. They are: Stmt Expr Term Factor_tail V
- The nullable nonterminals are: Stmt_list Term_tail.
- The endable nonterminals are: Boolean Factor_tail Term_tail Term Expr Stmt_list Stmt.
- No cycles.

nonterminal	first set	follow set	nullable	endable
Stmt_list	print	∅	yes	yes
Stmt	print	print	no	yes
Expr	∅	print	no	yes
Term	∅	or and 0 1 true false t f print	no	yes
Term_tail	or and 0 1 true false t f	print	yes	yes
Factor_tail	(Expr) not	print	no	yes
Factor	(Expr) not	(Expr) not	no	no
V	∅	(Expr) not assign	no	no
Boolean	0 1 true false t f	print	no	yes
Notop	not	∅	no	no
AndOrop	or and	(Expr) not	no	no

The grammar is LL(1).

Ως επιπλέον επιβεβαίωση, μετατρέψαμε τον ψευδοκώδικα σε κώδικα που αναγνωρίζει ο LL1 parser generator τον οποίο βρήκαμε στη σελίδα <https://planetcalc.com/5600/>

Προσπαθήσαμε να διατηρήσουμε τη μορφή όσο γίνεται πιο κοντά στον αρχικό ψευδοκώδικα:

```
stmt = expr,"=",boolean;
```

```
expr=term, expr1;
```

```
expr1=" and ",term,expr1|" or ",term,expr1|;
```

```
term=factor, term1;
```

```
term1=" and ", factor, term1 | " or ", factor, term1;
```

```
factor="(", expr , ")" | variable | " not ", variable;
```

```
variable=letter , {letter};
```

```
letter =
```

```
"a"|"b"|"c"|"d"|"e"|"f"|"g"|"h"|"i"|"j"|"k"|"l"|"m"|"n"|"o"|"p"|"q"|"r"|"s"|"t"|"u"|"v"|"w"|"x"|"y"|"z";
```

```
boolean = "true"|"false"|"t"|"f"|"0"|"1";
```

```
syntax=stmt;
```

Τα αποτελέσματα έχουν ως εξής:

Grammar analysis		
Result	Description	Addition
Success	Expression parsing	
Success	Every nonterminal has a rule in grammar.	
Success	There is no cyclic reference.	
Success	There is only one root rule.	syntax
Success	There is no left recursion.	
Success	LL(1) condition conformance.	
Success	Expression example has been parsed successfully.	

3. Πίνακες με FIRST και FOLLOW sets για όλα τα μη τερματικά σύμβολα

Nonterminal symbol	FIRST set	FOLLOW set
stmt \rightarrow expr, '=', Boolean	'(' [a-z] ' not '	\$
expr \rightarrow term, expr1	'(' [a-z] ' not ')' '='
expr1 \rightarrow (' and ', term, expr1) (' or ', term, expr1) ϵ	ϵ ' and ' ' or ')' '='
term \rightarrow factor, term1	'(' [a-z] ' not ')' ' and ' ' or ' '='
term1 \rightarrow (' and , factor, term1) (' or , factor, term1) ϵ	ϵ ' and ' ' or ')' ' and ' ' or ' '='
factor \rightarrow ('(', expr, ')') variable (' not ', variable)	'(' [a-z] ' not ')' ' and ' ' or ' '='
variable \rightarrow letter, letter x	[a-z])' ' and ' ' or ' '='
letter \rightarrow 'a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j' 'k' 'l' 'm' 'n' 'o' 'p' 'q' 'r' 's' 't' 'u' 'v' 'w' 'x' 'y' 'z'	[a-z])' [a-z] ' and ' ' or ' '='
boolean \rightarrow 'true' 'false' 't' 'f' '0' '1'	'f' 't' 'true' 'false' '0' '1'	\$
syntax \rightarrow stmt	'(' [a-z] ' not '	\$
letter x \rightarrow (letter, letter x) ϵ	ϵ [a-z])' ' and ' ' or ' '='

4. Συνοπτική περιγραφή των δύο αρχείων κώδικα

Parser.py

Βασιστήκαμε στο προτεινόμενο Recursive descent parser example (ανακτήθηκε από:
<https://gist.github.com/mixstef/946fce67f49f147991719bfa4d0101fa>)

Εισάγαμε τους κανόνες γραμματικής όπως δίνονται στον ψευδοκώδικα.

Μετατρέψαμε τα START sets ώστε να ταιριάζουν στους κανόνες της γραμματικής που δημιουργήσαμε.

Η εισαγωγή των κανονικών εκφράσεων έγινε με τη βιβλιοθήκη Plex σύμφωνα με τα παραδείγματα που δίνονται στο
<http://mixstef.github.io/courses/compilers/lecturedoc/unit4/module1.html>

Προσπαθήσαμε να μετατρέψουμε το κομμάτι του κώδικα που αφορά την εισαγωγή των δεδομένων από αρχείο ώστε η εισαγωγή να γίνεται από τη χρήση μέσω του ηλεκτρολογίου. Ωστόσο, τελικά επιλέξαμε τη λύση της εισαγωγής από αρχείο με την ονομασία input.txt

Για τα ονόματα των μεταβλητών επιλέξαμε να είναι ακολουθία μόνο από γράμματα.

Όσον αφορά τη συντακτική ανάλυση δε διαχωρίσαμε tokens για τα and και or και τα αντιμετωπίσαμε ενιαία γιατί στο κομμάτι του συντακτικού δεν διαφοροποιούνται. Το ίδιο επιλέξαμε για τις boolean εκφράσεις true και false (και τις παραλλαγές τους). Με τη χρήση της plex ενεργοποιήσαμε το case insensitivity για τις Boolean εκφράσεις.

Runner.py

Μετατρέψαμε το scanner.py ώστε να εκτελεί τις αναθέσεις και να εκτυπώνει τα αποτελέσματα.

5. Αποτελέσματα εξόδου

...

6. Πηγές

Στο θεωρητικό κομμάτι βασιστήκαμε στην ύλη του μαθήματος:

<http://mixstef.github.io/courses/compilers/>

Στη δημιουργία του parser:

<https://gist.github.com/mixstef/946fce67f49f147991719bfa4d0101fa>

Για τη βιβλιοθήκη plex:

<http://mixstef.github.io/courses/compilers/lecturedoc/unit4/module1.html>