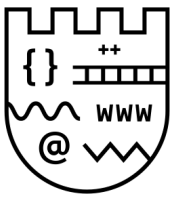

Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Τμήμα Πληροφορικής

Τεχνική αναφορά για NCO-04-01

Μεταγλωττιστής της γλώσσας SimplePascal

Αλέξανδρος Κόρκος
alexkork@csd.auth.gr
3870

Θεσσαλονίκη, 28 Δεκεμβρίου 2022

Εισαγωγή

Η εργασία αυτή αναπτύχθηκε με το εργαλείο ANTLr v4 σε συνδυασμό με την γλώσσα Java.

Η δομή του Project έχει ως εξής:

```
SimplePascal
├── ...
├── src
│   ├── ...
│   ├── SimplePascal.g4
│   └── Visitor.java
├── SimplePascal.jar
├── Counter.p
├── test.p
├── ...
├── SimplePascalTest1.p
└── SimplePascalTest2.p
```

Το αρχείο `SimplePascal.g4` περιέχει την λεξική και την συντακτική ανάλυση, με βάση την εκφώνηση της εργασίας. Στο αρχείο `Visitor.java` γίνεται η σημασιολογική ανάλυση και συνεπώς, η παράγωγη του πίνακα συμβολών και του συντακτικού δένδρου. Τα υπόλοιπα αρχεία του φακέλου `src`, αποτελούν βοηθητικές κλάσεις για την ολοκλήρωση της εργασίας.

Τα αρχεία με κατάληξη `.p`, αποτελούνε προγράμματα σε γλώσσα SimplePascal που μπορούν να εκτελεσθούν.

Τέλος, το αρχείο `SimplePascal.jar` είναι εκείνο που θα εκτελεσθεί για να γίνει η παραγωγή των στόχων τις εργασίας. Για αυτό τον λόγο, στο φάκελο `SimplePascal` θα πρέπει να ανοιχθεί ένα παράθυρο της γραμμής εντολών και πληκτρολογηθεί η παρακάτω πρόταση [1]:

```
java -jar SimplePascal.jar <filename>.p
```

όπου `<filename>.p` το όνομα του αρχείου (π.χ. SimplePascalTest1.p).

Είναι σημαντικό, το αρχείο `.jar` και το πρόγραμμα (αρχείο `.p`) που θα εκτελέσετε να βρίσκονται στον **ίδιο φάκελο**.

[1] Είναι αναγκαίο να υπάρχει εγκαταστημένο κάποιο Java JDK, Download Java JDK 18

Κεφάλαιο 1

Λεξική ανάλυση

Εδώ θα παρουσιαστούν εν συντομία, κάποιοι από τους σημαντικότερους λεξικούς κανόνες.

```
RCONST: [0-9]*DOT[0-9]+(E DASH?[0-9])?
    | ([1-9][0-9]*)E DASH?[0-9]
    | (ZERO H)((ZERO|([1-9][0-9]|[A-Fa-f])+)DOT(ZERO|([1-9][0-9]|[A-Fa-f])+))
    | (ZERO B)([0-1]*DOT[0-1]*[1][0-1]*);

ICONST: ZERO
    | [1-9]+[0-9]*
    | (ZERO H)([1-9] | [a-fA-F])+([0-9] | [a-fA-F])*
    | (ZERO B)[1]+[0-1]* ;
```

Κώδικας 1.1: Υλοποίηση RCONST και ICONST

Όπως φαίνεται στο παραπάνω απόσπασμα 1.1, πραγματοποιείται υλοποίηση των σταθερών για ακέραιους και πραγματικούς αριθμούς όχι Μονό για την δεκαδική αναπαράσταση των αριθμών, άλλα και για την δεκαεξαδική και δυαδική τους αναπαράσταση.

```
CCONST : EARS(LETTER | NPC)EARS;
```

Κώδικας 1.2: Υλοποίηση CCONST

Από το κώδικά 1.2 φαίνεται η υλοποίηση των σταθερών χαρακτήρων. Περικλείονται από μονά αυτακια (EARS) και το περιεχόμενο μπορεί να είναι είτε ένα non-printing character (NPC) είτε ένας και μόνο χαρακτήρας στο διάστημα ASCII που δίνει η εκφώνηση της εργασίας.

```
ID : UNDERSCORE*([A-Za-z])+(([A-Za-z] | [0-9])*|(UNDERSCORE*([A-Za-z] | [0-9]
↪  )+)*);
```

Κώδικας 1.3: Υλοποίηση ID

Εδώ γίνεται η αναγνώρισή των αναγνωριστικών, σύμφωνα πάντα με την εκφώνησή της εργασίας. Μια σημείωση, η δήλωση για την αναγνώριση των συμβολοσειρων είναι πιο πάνω από αυτή για τα ID, καθώς στο εργαλείο ANTLr η δήλωση που βρίσκεται χαμηλότερα έχει μεγαλύτερη προτεραιότητα (εάν ήτανε ανάστροφη η σειρά εμφάνισης, τα ID θα αναγνωριζόταν σαν String).

Κεφάλαιο 2

Συντακτική ανάλυση

Το εργαλείο ANTLr, είναι γνωστό πως μπορεί να λειτουργήσει ακόμα και όταν υπάρχει αριστερή ανάδρομη. Για αυτό τον λόγο, δεν αντιμετωπίστηκαν αυτά τα φαινόμενα μόνο σε ορισμένες περιπτώσεις που ήτανε απαραίτητα για την σημασιολογική ανάλυση. Επιπλέον, έχουνε οριστεί κάποιες συναρτήσεις για την μετατροπή μεταξύ των συστημάτων αναπαράστασης των αριθμών (`hexadecimalToDecimal`, `hexadecimalToReal`, κτλ).

```
expression : NOTOP expression
        | op = (ADDOP | SUBOP) expression
        | expression op = (MULOP | DIVOP | DIV | MOD | ANDOP) expression
        | expression op = (ADDOP | SUBOP) expression
        | expression op = (INOP | LTEQ | GTEQ | LT | GT | EQU | NEQ) expression
        | expression OROP expression
        | variable
        | ID LPAREN expressions RPAREN
        | constant
        | LPAREN expression RPAREN
        | setexpression
        ;
```

Κώδικας 2.1: Υλοποίηση του κανόνα expression

Όπως φαίνεται στο παραπάνω τμήμα 2.1, έχουνε γίνει αρκετές αλλαγές με σκοπό να επιλυθεί οποιοδήποτε ζήτημα προτεραιότητας τελεστών. Στο εργαλείο ANTLr, ο κανόνας που βρίσκεται πιο πάνω έχει και την μεγαλύτερη προτεραιότητα.

```
variable_defs : identifiers? COLON typename (SEMI identifiers? COLON typename)*
↪  ;
type_defs     : ID EQU type_def (SEMI ID EQU type_def)* ;
constant_defs : ID EQU expression (SEMI ID EQU expression)* ;
```

Κώδικας 2.2: Υλοποίηση κανόνων για τις μεταβλητές, τους τύπους και τις σταθερές

Οι κανόνες του αποσπάσματος 2.2 που δόθηκαν στην εκφώνηση της εργασίας, υλοποιούνταν με αριστερή αναδρομή. Το εργαλείο ANTLr μπορεί να χειριστεί της αριστερές αναδρομές αυτόματα και χωρίς πρόβλημα ωστόσο, για τον ευκολότερο χειρισμό για την ευκολότερη σημασιολογική ανάλυση οι εκφράσεις τροποποιήθηκαν όπως φαίνεται και στο κώδικα 2.2.

Κεφάλαιο 3

Σημασιολογική ανάλυση

Για να επιτευχθεί η σημασιολογική ανάλυση, είναι αναγκαίο να υπάρχει ένα συντακτικό δένδρο (Abstract Syntax Tree – AST). Για κάθε κόμβο του δένδρου, το εργαλείο παρέχει μια βασική μέθοδο η οποία είναι κενή, η οποία καλείται όταν ο parser αντιστοιχήσει την συμβολοσειρα της εισόδου με τον κανόνα που αναπαριστά ο κόμβος του δένδρου. Για να γίνει λοιπόν η συντακτική ανάλυση, πολλές από αυτές τις μεθόδους υλοποιήθηκαν από τον συντακτική αυτής της αναφοράς. Οι μέθοδοι που εν τέλει υπερφορτωθηκανε φαίνονται στο αρχείο `Visitor.java`.

3.1 Πίνακας συμβόλων

Ο υλοποιημένος πίνακας συμβολών, κρατάει τις μεταβλητές, σταθερές και τύπους και όλες τις σχετικές πληροφορίες, όπως εμβέλεια (τοπική ή καθολική), τύπο δεδομένων, το αναγνωριστικό τους όνομα και την τιμή.

```
+-------------------------------------------------------------------------------------------+
|                                  Symbol table:                                            |
+-------------------------------------------------------------------------------------------+
+-------------------------------------------------------------------------------------------+
|                         Variables, types and constants:                                   |
+-------------------------------------------------------------------------------------------+
ID: ar1 | scope: Global | dataType: Real | type: Array
ID: y | scope: Global | dataType: Integer | type: Array
ID: j | scope: Global | value is not set yet | dataType: Integer
ID: b | scope: Global | dataType: dataType is defined by the user (i.e. Type)| type: Array
ID: w | scope: Global | value is not set yet | dataType: Real
ID: monday | scope: Global | value: 0 | dataType: Enum
ID: k | scope: Local | value is not set yet | dataType: Integer
ID: friday | scope: Global | value: 2 | dataType: Enum
ID: omega | scope: Local | value: W | dataType: Char
ID: i | scope: Global | value is not set yet | dataType: Integer
ID: oioio | scope: Global | value: 27.0 | dataType: Integer
ID: r | scope: Global | type: Record
ID: int | scope: Local | value: 5 | dataType: Integer
ID: re | scope: Global | type: Record
ID: pppp | scope: Global | value: true | dataType: Boolean
ID: c | scope: Global | dataType: Char | type: Array
ID: day | scope: Global | type: EnumStruct
ID: day_array | scope: Global | dataType: Char | type: Array
ID: xx | scope: Global | dataType: Real | type: Array
ID: z | scope: Global | value is not set yet | dataType: Real
ID: ar | scope: Global | dataType: Integer | type: Array
ID: g | scope: Global | value: 11.0 | dataType: Integer
ID: re1 | scope: Global | type: Record
ID: c | scope: Local | value is not set yet | dataType: Integer
ID: gar | scope: Global | type: SubArea
ID: tuesday | scope: Global | value: 1 | dataType: Enum
ID: arre1 | scope: Global | dataType: dataType is defined by the user (i.e. Type)| type: Array
ID: x | scope: Global | dataType: Integer | type: Array
ID: k | scope: Global | value is not set yet | dataType: Integer
+-------------------------------------------------------------------------------------------+
|                              Functions and procedures:                                    |
+-------------------------------------------------------------------------------------------+
ID: _all | type: Procedure | return: Void | parameters: [ID: n | scope: Local, ID: x | scope: Local]
ID: try_me | type: Function | return: Real | parameters: [ID: i | scope: Local, ID: j | scope: Local]
ID: try_me | type: Function | return: Boolean | parameters: [ID: i | scope: Local, ID: j | scope: Local]
ID: kkk | type: Procedure | return: Void | parameters: []
ID: nested | type: Procedure | return: Void | parameters: [ID: i | scope: Local]
ID: try_me | type: Function | return: Void | parameters: []
```

Σχήμα 3.1: Πίνακας συμβολών για το αρχείο SimplePascalTest2.p

Επιπλέον ο όπως φαίνεται από το σχήμα 3.1, κρατά πληροφορία για τα υποπρογράμματα, όπως το αναγνωριστικό όνομα, το είδος (συνάρτηση ή διαδικασία), των τύπο δεδομένων της επιστρεφόμενης τιμής και τέλος τις παραμέτρους (εάν υπάρχουν) του υποπρογράμματος.

3.2 Συντακτικό δένδρο

Το συντακτικό δένδρο που δημιουργείτε κάθε φορά, παρέχεται από το εργαλείο ANTLr. Στην συνέχεια το δένδρο διασχίζεται με preorder τρόπο (διότι το εργαλείο ANTLr χρησιμοποιεί LL parser) ξεκινώντας από την ρίζα του δένδρου, που είναι ο κανόνας programm. Τέλος, έχει υλοποιηθεί και μια κλάσης `TreeUtils`, για την εκτύπωση του δένδρου σε περιβάλλον εντολών όπως φαίνεται και παρακάτω.

```
+-------------------------------------------------------------------------------+
|                          Abstract syntax tree:                                |
+-------------------------------------------------------------------------------+
program
  header programtry_it;
  declarations
    constdefs const
      constant_defs g=
        expression
          constant 0B1011
      ;
    typedefs type
      type_defs ar=
        type_def array[
          dims
            limits
              limit 1
              ..
              limit g
          ]of
          typename
            standard_type integer
        ;ar1=
        type_def array[
          dims
            limits
              limit 1
              ..
              limit g
          ]of
          typename
            standard_type real
        ;re=
        type_def record
          fields
            field
              identifiers x1,x2
              :
              typename
                standard_type integer
          end
        ;re1=
        type_def record
          fields
            field
              identifiers b
              :
              typename
                standard_type boolean
          end
        ;gar=
        type_def
          limit -g
          ..
          limit g
```

Σχήμα 3.2: Απόσπασμα του συντακτικού δένδρο για το αρχείο SimplePascalTest2.p

Βιβλιογραφία

[1] Bart Kiers, *Tiny Language for ANTLR 4*, 2020.

[2] Bart Kiers, *How to exclude " and in ANTLR 4 string matching?*, 2013.

[3] Gabriele Tomassetti, *The ANTLR Mega Tutorial*, 2016.

[4] Gerald Rosenberg , *SnippetsTest for Antlr4*, 2021.

[5] Ruslan Spivak, *Let's Build A Simple Interpreter. Part 13: Semantic Analysis.*, 2020.