ΜΕΤΑΦΡΑΣΤΕΣ

ΕΡΓΑΣΤΗΡΙΑΚΗ ΆΣΚΗΣΗ

ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2024 – 2025

ΠΑΓΩΝΗΣ ΔΗΜΗΤΡΙΟΣ Α.Μ 4985
ΤΖΑΛΟΚΩΣΤΑΣ ΔΗΜΗΤΡΙΟΣ Α.Μ 4994

ΜΕΡΟΣ 1^{Ο}

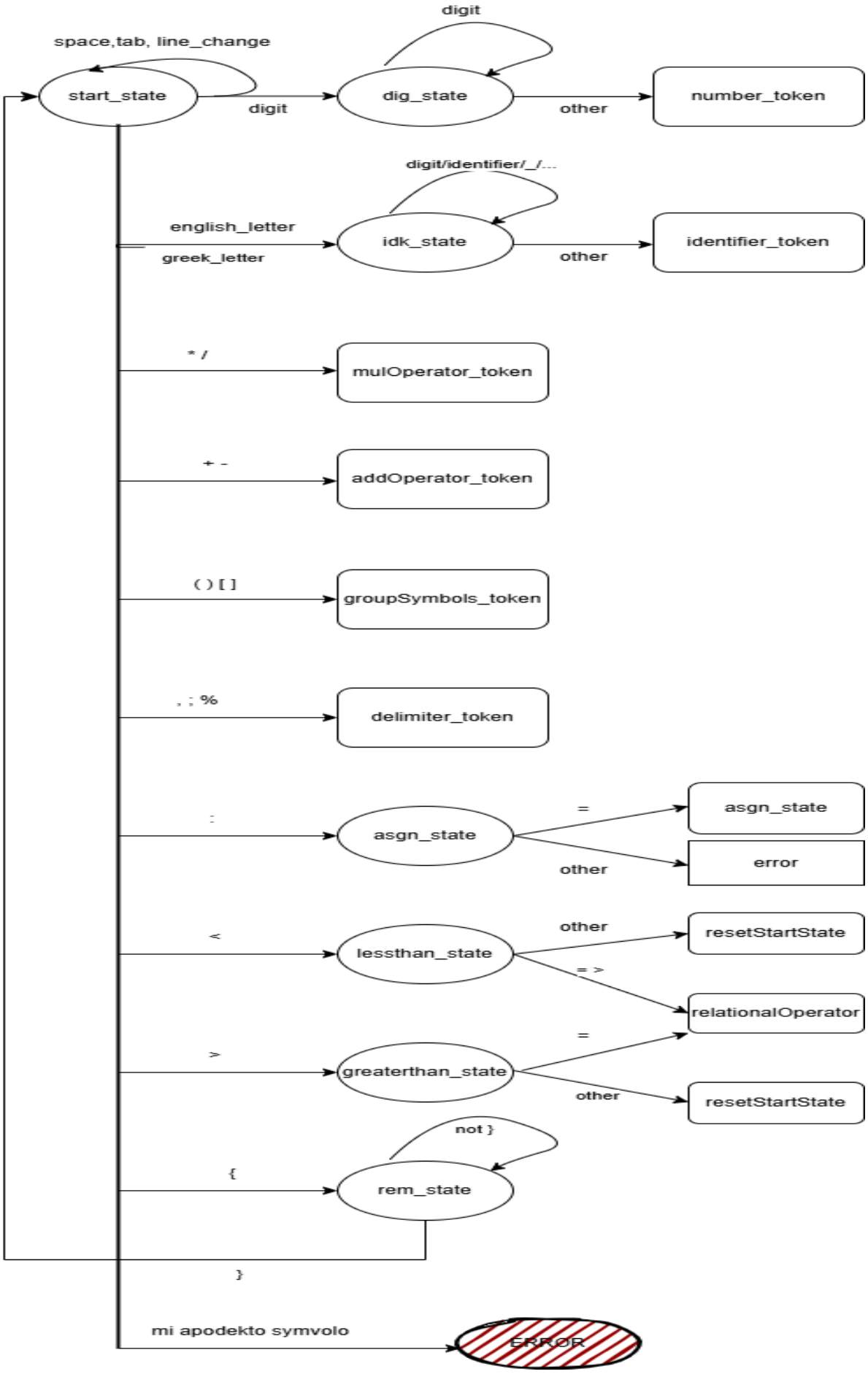
ΛΕΚΤΙΚΟΣ ΑΝΑΛΥΤΗΣ

Ο λεκτικός αναλυτής (lexer) είναι το πρώτο στάδιο της μεταγλώττισης και έχει ως στόχο να διαβάσει το αρχικό αρχείο πηγαίου κώδικα (.gre) και να το μετατρέψει σε λεκτικές μονάδες (tokens). Κάθε λεκτική μονάδα αποτελεί μια ακολουθία χαρακτήρων που έχει συγκεκριμένη σημασία για τη γλώσσα, όπως είναι οι λέξεις-κλειδιά, τα ονόματα μεταβλητών, οι αριθμοί, οι τελεστές και τα σύμβολα.

Η λειτουργία του λεκτικού αναλυτή βασίζεται σε ένα πεπερασμένο αυτόματο, το οποίο μεταβαίνει από κατάσταση σε κατάσταση ανάλογα με τον χαρακτήρα που διαβάζει κάθε φορά. Για κάθε ακολουθία χαρακτήρων που αναγνωρίζει, δημιουργεί το αντίστοιχο token και το προσθέτει στη λίστα των λεκτικών μονάδων που θα χρησιμοποιηθούν από τα επόμενα στάδια της μεταγλώττισης.

Με αυτόν τον τρόπο, ο λεκτικός αναλυτής μετατρέπει τον ακατέργαστο πηγαίο κώδικα σε μια ακολουθία από tokens, τα οποία αποτελούν τη βάση για τη συντακτική και σημασιολογική ανάλυση που ακολουθεί.

Στην παρακάτω διαφάνεια σας παραθέτουμε μια πρώτη προσέγγιση του αυτομάτου λειτουργίας του λεκτικού αναλυτή, το οποίο χρησιμοποιήσαμε σαν βάση για να ξεκινήσουμε.


digit
space,tab, line_change
start_state
digit
dig_state
other
number_token
digit/identifier/_/...
english_letter
greek_letter
idk_state
other
identifier_token
* /
mulOperator_token
+ -
addOperator_token
( ) [ ]
groupSymbols_token
, ; %
delimiter_token
:
asgn_state
=
other
asgn_state
error
<
lessthan_state
other
= >
resetStartState
relationalOperator
>
greaterthan_state
=
other
resetStartState
not }
{
rem_state
}
mi apodekto symvolo
ERROR

Αρχικά, χρησιμοποιήσαμε την κλάση Token όπως την παραθέτετε στον ηλεκτρονικό σας σύγγραμμα, προκειμένου να αποθηκευτεί η πληροφορία που μεταφέρει ο λεκτικός αναλυτής στα επόμενα στάδια του μεταγλωττιστή.

```
class Token:
    Tabnine | Edit | Test | Explain | Document
    def __init__(self, recognized_string, family, line_number):
        self.recognized_string = recognized_string
        self.family = family
        self.line_number = line_number

    Tabnine | Edit | Test | Explain | Document
    def __repr__(self):
        return f"[{self.recognized_string} {self.family} {self.line_number}]"
```

Έπειτα όπως θα δείτε και παρακάτω ορίσαμε μια κλάση Lex προκειμένου να κατηγοριοποιήσουμε καλύτερα τις φάσεις της μεταγλώττισης . Έτσι, ορίσαμε λίστες με όλους τους έγκυρους χαρακτήρες της γλώσσας, όπως ελληνικά και αγγλικά γράμματα, αριθμούς, τελεστές, σύμβολα και λέξεις-κλειδιά και με αυτόν τον τρόπο μπορούμε να αναγνωρίσουμε τι επιτρέπεται στη γλώσσα και να απορρίψουμε μη έγκυρους χαρακτήρες.

```
class Lex:
    # characters and allowed symbols start
    greek_letters = ['α', 'β', 'γ', 'δ', 'ε', 'ζ', 'η', 'θ', 'ι', 'κ', 'λ', 'μ',
            'ν', 'ξ', 'ο', 'π', 'ρ', 'σ', 'τ', 'υ', 'φ', 'χ', 'ψ', 'ω','ς','Α', 'Β', 'Γ', '
            'Ν', 'Ξ', 'Ο', 'Π', 'Ρ', 'Σ', 'Τ', 'Υ', 'Φ', 'Χ', 'Ψ', 'Ω',
            'Ά', 'Έ', 'Ό', 'Ί', 'Ύ', 'Ή', 'Ώ', 'ά', 'έ', 'ό', 'ί', 'ύ', 'ή', 'ώ']
    english_letters = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N'
            'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z','a', 'b', 'c', 'd', 'e', 'f',
            'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
    numbers = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
    addOperators = ['+', '-']
    mulOperators = ['*', '/']
    delimeters = [';', ',', '%']
    groupSymbols = ['(', ')', '[', ']']
    relationalOperators = ['<', '>', '=']
    assignents = [':', '=']
    multiTokens =[':=', '<=', '>=', '<>']
    logicalOperators = ['and', 'or', 'not']
    comments = ['{', '}'] # DEN EINAI TOKEN EINAI SXOLIO PREPEI NA AGNOHTHEI
```

Οι πιο βασικές μέθοδοι της Lex παραθέτονται παρακάτω :

- Η __init__ αρχικοποιεί τον λεκτικό αναλυτή, διαβάζει το αρχείο και μηδενίζει τους απαραίτητους δείκτες.
- Η read_file_content διαβάζει το περιεχόμενο του αρχείου πηγαίου κώδικα.
- Η lex_states είναι η βασική μέθοδος που διαβάζει χαρακτήρα-χαρακτήρα, αλλάζει κατάσταση ανάλογα με το τι διαβάζει και όταν ολοκληρωθεί ένα token το δημιουργεί και το αποθηκεύει.
- Η next_token δημιουργεί ένα αντικείμενο Token με τα στοιχεία του ήδη αναγνωρισμένου token.
- Η resetStartState επαναφέρει τον αναλυτή στην αρχική κατάσταση μετά από κάθε token και προσθέτει το παρόν token στη λίστα αποτελεσμάτων.

Γενικά ο τρόπος με τον οποίο λειτουργεί το αυτόματο είναι ο εξής:

Ξεκινάει από την αρχική κατάσταση και διαβάζει τον επόμενο χαρακτήρα. Αν είναι ψηφίο, πηγαίνει σε κατάσταση αριθμού και συνεχίζει να διαβάζει μέχρι να τελειώσει ο αριθμός. Αν είναι γράμμα, πηγαίνει σε κατάσταση ονόματος και συνεχίζει μέχρι να τελειώσει το όνομα. Αν είναι τελεστής ή σύμβολο, το αναγνωρίζει αμέσως ή ελέγχει αν είναι διπλός τελεστής (όπως :=, <=). Αν είναι σχόλιο ({ ... }), αγνοεί ό,τι υπάρχει μέχρι να βρει το κλείσιμο (}). Όταν ολοκληρωθεί ένα token, το προσθέτει στη λίστα αποτελεσμάτων και επιστρέφει στην αρχική κατάσταση για να συνεχίσει.

Για παράδειγμα, αν διαβάσει τη γραμμή: x := 5 + 3

Θα παράξει τα εξής tokens: [x identifier 1] [:= assignment 1] [5 number 1] [+ addOperator 1] [3 number 1]

ΣΥΝΤΑΚΤΙΚΟΣ ΑΝΑΛΥΤΗΣ

Μετά τη λεκτική ανάλυση, ακολουθεί το στάδιο της συντακτικής ανάλυσης, όπου εξετάζεται η δομή και η ορθότητα των προτάσεων του προγράμματος σύμφωνα με τους κανόνες της γραμματικής της γλώσσας. Ο συντακτικός αναλυτής λαμβάνει ως είσοδο τη σειρά των λεκτικών μονάδων (tokens) που παρήγαγε ο λεκτικός αναλυτής και ελέγχει αν αυτές σχηματίζουν σωστές συντακτικές δομές.

Στο στάδιο αυτό, γίνεται διάκριση ανάμεσα σε απλές και δομημένες εντολές. Οι απλές εντολές περιλαμβάνουν εντολές ανάθεσης τιμής σε μεταβλητές, είσοδο και έξοδο δεδομένων, καθώς και κλήσεις υποπρογραμμάτων. Οι δομημένες εντολές αφορούν πιο σύνθετες δομές, όπως εντολές επιλογής (π.χ. if-else) και επανάληψης (π.χ. while, for).

Ο συντακτικός αναλυτής που υλοποιείται βασίζεται σε μια ιεραρχική ανάλυση ,όπου κάθε συντακτικός κανόνας της γλώσσας αντιστοιχεί σε μια μέθοδο της κλάσης Syntaktikos. Μέσα από αυτές τις μεθόδους, ελέγχεται αν οι λεκτικές μονάδες ακολουθούν τη σωστή σειρά και δομή, ενώ σε περίπτωση που εντοπιστεί απόκλιση από τη γραμματική, παράγεται κατάλληλο μήνυμα συντακτικού λάθους.

Επιπλέον, κατά τη διάρκεια της συντακτικής ανάλυσης, γίνεται και η διαχείριση των παραμέτρων των υποπρογραμμάτων, τόσο με μεταφορά τιμής όσο και με αναφορά, ενώ υποστηρίζονται και τοπικές

και κύριες συναρτήσεις, με σαφή διαχωρισμό των ρόλων τους στο πρόγραμμα. Με αυτόν τον τρόπο, ο συντακτικός αναλυτής διασφαλίζει ότι το πρόγραμμα ακολουθεί τη σωστή δομή και είναι έτοιμο για τα επόμενα στάδια της μεταγλώττισης.

Όπως και στον λεκτικό, έτσι και στον συντακτικό δημιουργήσαμε μια κλάση Syntaktikos η οποία εκτελεί τις λειτουργίες που αναφέρθηκαν και παραπάνω.
Οι κύριες συναρτήσεις και η λειτουργία τους παραθέτονται παρακάτω:

- Η __init__ αρχικοποιεί τον parser, δημιουργεί το λεξικό αναλυτή και ξεκινάει την ανάλυση καλώντας τη program().
- Η program ελέγχει αν το πρόγραμμα ξεκινά σωστά με τη λέξη-κλειδί πρόγραμμα και ένα όνομα προγράμματος (identifier) και καλεί το programblock για να συνεχίσει την ανάλυση.
- Η programblock ελέγχει τις δηλώσεις μεταβλητών (declarations), τα υποπρογράμματα (subprograms) και το κύριο σώμα του προγράμματος, όπως επίσης ελέγχει αν υπάρχουν οι λέξεις-κλειδιά «αρχή_προγράμματος» και «τέλος_προγράμματος» .
- Οι declarations & varlist επεξεργάζονται τις δηλώσεις μεταβλητών και τη λίστα μεταβλητών.
- Οι subprograms , func και proc επεξεργάζονται τις δηλώσεις υποπρογραμμάτων (συναρτήσεις και διαδικασίες).
- Οι sequence & statement επεξεργάζονται ακολουθίες εντολών και μεμονωμένες εντολές (ανάθεση, if, while, for, διάβασε, γράψε, εκτέλεσε κ.λπ.).
- Οι expression,term και factor επεξεργάζονται αριθμητικές εκφράσεις, όρους και παράγοντες (αριθμοί, μεταβλητές, παρενθέσεις).

- Οι condition,boolterm και boolfactor επεξεργάζονται λογικές εκφράσεις (συνθήκες).
- Οι relational_oper,add_oper και mul_oper ελέγχουν αν τα επόμενα tokens είναι τα σωστά τελεστικά σύμβολα (σχέσης, πρόσθεσης, πολλαπλασιασμού).

ΤΕΣΤ ΠΡΏΤΟΥ ΜΕΡΟΥΣ

Προκειμένου να τεστάρουμε και να βεβαιωθούμε πως το πρώτο μέρος δουλεύει ορθά , προσθέσαμε στο τέλος του προγράμματος μια δοκιμαστική συνάρτηση η οποία θα εκτυπώνει όλα τα tokens και στο τέλος EOF εφόσον όλα κύλισαν ομαλά και δεν προέκυψε κάποιο σφάλμα.

Το αρχείο που χρησιμοποιήσαμε για έλεγχο είναι το test.gre :

test.gre

```
1   πρόγραμμα τεστ
2       δήλωση α,β
3       δήλωση γ
4
5       συνάρτηση αύξηση(α,β)
6           διαπροσωπεία
7           είσοδος α
8           έξοδος β
9           αρχή_συνάρτησης
10              β := α + 1 ;
11              αύξηση := α + 1 { δεν μπαίνει ερωτηματικό
12                                είναι τέλος block }
13          τέλος_συνάρτησης
14
15          διαδικασία τύπωσε_συν_1(χ)
16              διαπροσωπεία
17              είσοδος χ
18              αρχή_διαδικασίας
19                  γράψε χ+1
20          τέλος_διαδικασίας
21
22          αρχή_προγράμματος
23              α := 1 ;
24              β := 2 + α * α / (2 - α - (2*α));
25              γ := αύξηση(α,%β);
26              για α:=1 έως 8 με_βήμα 2 επανάλαβε
27                  εκτέλεσε τύπωσε_συν_1(α)
28              για_τέλος;
29              β := 1 ;
30              όσο β<10 επανάλαβε
31                  εάν β<>22 ή [β>=23 και β<=24] τότε
32                      β := β+1
33                  εάν_τέλος { όχι ερωτηματικό, είναι τέλος block }
34              όσο_τέλος; { θέλει ερωτηματικό
35                           χωρίζει εκτελέσιμες εντολές }
36              διάβασε β;
37              επανάλαβε
```

```
38              β := β + 1
39            μέχρι β<-100
40    τέλος_προγράμματος
41
```

Το output του συγκεκριμένου αρχείου ήταν αυτό:

```
[[πρόγραμμα keyword 1], [τεστ identifier 1], [δήλωση keyword 2], [α identifier 2], [, delimeter 2], [β identifier 2], [δή
λωση keyword 3], [γ identifier 3], [συνάρτηση keyword 5], [αύξηση identifier 5], [( groupSymbol 5], [α identifier 5], [,
delimeter 5], [β identifier 5], [) groupSymbol 5], [διαπροσωπεία keyword 6], [είσοδος keyword 7], [α identifier 7], [έξοδ
ος keyword 8], [β identifier 8], [αρχή_συνάρτησης keyword 9], [β identifier 10], [:= assignent 10], [α identifier 10], [+
 addOperator 10], [1 number 10], [; delimeter 10], [αύξηση identifier 11], [:= assignent 11], [α identifier 11], [+ addOp
erator 11], [1 number 11], [τέλος_συνάρτησης keyword 13], [διαδικασία keyword 15], [τύπωσε_συν_1 identifier 15], [( group
Symbol 15], [χ identifier 15], [) groupSymbol 15], [διαπροσωπεία keyword 16], [είσοδος keyword 17], [χ identifier 17], [α
ρχή_διαδικασίας keyword 18], [γράψε keyword 19], [χ identifier 19], [+ addOperator 19], [1 number 19], [τέλος_διαδικασίας
 keyword 20], [αρχή_προγράμματος keyword 22], [α identifier 23], [:= assignent 23], [1 number 23], [; delimeter 23], [β i
dentifier 24], [:= assignent 24], [2 number 24], [+ addOperator 24], [α identifier 24], [* mulOperator 24], [α identifier
 24], [/ mulOperator 24], [( groupSymbol 24], [2 number 24], [- addOperator 24], [α identifier 24], [- addOperator 24], [
( groupSymbol 24], [2 number 24], [* mulOperator 24], [α identifier 24], [) groupSymbol 24], [) groupSymbol 24], [; delim
eter 24], [γ identifier 25], [:= assignent 25], [αύξηση identifier 25], [( groupSymbol 25], [α identifier 25], [, delimet
er 25], [% delimeter 25], [β identifier 25], [) groupSymbol 25], [; delimeter 25], [για keyword 26], [α identifier 26], [
:= assignent 26], [1 number 26], [έως keyword 26], [8 number 26], [με_βήμα keyword 26], [2 number 26], [επανάλαβε keyword
 26], [εκτέλεσε keyword 27], [τύπωσε_συν_1 identifier 27], [( groupSymbol 27], [α identifier 27], [) groupSymbol 27], [γι
α_τέλος keyword 28], [; delimeter 28], [β identifier 29], [:= assignent 29], [1 number 29], [; delimeter 29], [όσο keywor
d 30], [β identifier 30], [< relationalOperator 30], [10 number 30], [επανάλαβε keyword 30], [εάν keyword 31], [β identif
ier 31], [<> relationalOperator 31], [22 number 31], [ή keyword 31], [[ groupSymbol 31], [β identifier 31], [>= relationa
lOperator 31], [23 number 31], [και keyword 31], [β identifier 31], [<= relationalOperator 31], [24 number 31], [] groupS
ymbol 31], [τότε keyword 31], [β identifier 32], [:= assignent 32], [β identifier 32], [+ addOperator 32], [1 number 32],
 [εάν_τέλος keyword 33], [όσο_τέλος keyword 34], [; delimeter 34], [διάβασε keyword 36], [β identifier 36], [; delimeter
36], [επανάλαβε keyword 37], [β identifier 38], [:= assignent 38], [β identifier 38], [+ addOperator 38], [1 number 38],
επανάλαβε keyword 37], [β identifier 38], [:= assignent 38], [β identifier 38], [+ addOperator 38], [1 number 38], [μέχρι
 keyword 39], [β identifier 39], [< relationalOperator 39], [- addOperator 39], [100 number 39], [τέλος_προγράμματος keyw
ord 40]]
EOF reached
```

Η εκτέλεση του πρώτου μέρους με το δοσμένο αρχείο εισόδου δείχνει ότι το πρόγραμμα λειτουργεί σωστά, καθώς αναγνωρίζει και τυπώνει όλα τα tokens που περιέχονται στο πρόγραμμα. Τα tokens που εμφανίζονται στην έξοδο αντιστοιχούν σε λέξεις-κλειδιά, identifiers, αριθμούς, τελεστές και διαχωριστικά, όπως ακριβώς ορίζονται στη γλώσσα. Επιπλέον, τα σχόλια αγνοούνται, ενώ τα λάθη στη σύνταξη ή στους χαρακτήρες εμφανίζονται με σαφή μηνύματα. Η ορθή εκτύπωση των tokens επιβεβαιώνει ότι το πρώτο στάδιο της ανάλυσης εκτελείται

επιτυχώς.

ΜΕΡΟΣ 2Ο

ΕΝΔΙΑΜΕΣΟΣ ΚΩΔΙΚΑΣ

Ο ενδιάμεσος κώδικας είναι μια αναπαράσταση του αρχικού προγράμματος σε μια πιο απλή, αλλά ακόμα υψηλού επιπέδου, μορφή. Δεν είναι τόσο κοντά στη γλώσσα μηχανής, αλλά ούτε τόσο σύνθετος όσο η αρχική γλώσσα. Η πιο συνηθισμένη μορφή ενδιάμεσου κώδικα είναι οι τετράδες (quads).

Η κλάση Endiamesos είναι υπεύθυνη για τη δημιουργία, διαχείριση και ενημέρωση των τετράδων του ενδιάμεσου κώδικα.

```
class EndiamesosKwdikas:
    def __init__(self):
        self.total_quads = []  # Λίστα με όλες τις τετράδες που έχουν παραχθεί
        self.count = 1         # Μετρητής για την αρίθμηση των τετράδων (ετικέτα)
        self.T_i = 1           # Μετρητής για τα προσωρινά ονόματα μεταβλητών (T_1, T_2, .
```

Πιο αναλυτικά, οι παρακάτω μέθοδοι αποτελούν τον βασικό μηχανισμό παραγωγής και διαχείρισης του ενδιάμεσου κώδικα:

- nextQuad: Επιστρέφει τον αριθμό της επόμενης τετράδας.
- genQuad* Δημιουργεί και προσθέτει μια νέα τετράδα στη λίστα.
- newTemp: Επιστρέφει νέο όνομα προσωρινής μεταβλητής.
- emptyList: Επιστρέφει κενή λίστα.
- makeList: Επιστρέφει λίστα με ένα στοιχείο.
- merge : Συγχωνεύει δύο λίστες.
- backPatch: Ενημερώνει τετράδες με τη σωστή ετικέτα άλματος.

Πέρα από τη δημιουργία της κλάσης EndiamesosKwdikas, αλλαγές έγιναν και στην κλάση Syntaktikos, η οποία πλέον σε κάθε σημείο που αναλύει μια συντακτική δομή, παράγει και τον αντίστοιχο ενδιάμεσο κώδικα (τετράδες), διαχειρίζεται τα άλματα και τις ετικέτες με backpatching, και στο τέλος εξάγει όλο τον ενδιάμεσο κώδικα σε αρχείο.

Οι βασικές αλλαγές που έγιναν στην κλάση Syntaktikos σχετικά με τον ενδιάμεσο κώδικα είναι οι εξής:

1. Εισαγωγή και χρήση της κλάσης EndiamesosKwdikas

- **Δημιουργία αντικειμένου**: Στον constructor (__init__), προστέθηκε η γραμμή:

```
self.endiamesos = EndiamesosKwdikas()
```

2. Παραγωγή τετράδων κατά την ανάλυση

- **Παραγωγή τετράδων**: Σε διάφορες συντακτικές δομές (π.χ. programblock, funcblock, procblock, assignment_stat, if_stat, while_stat, for_stat, print_stat, input_stat, call_stat, expression, term, factor, κ.ά.), καλείται η μέθοδος genQuad της κλάσης EndiamesosKwdikas για να δημιουργηθεί η αντίστοιχη τετράδα του

ενδιάμεσου κώδικα.

```
def programblock(self, name):
    self.declarations()
    self.subprograms()
    #self.lexer_results = self.lexer.lex_states()


    if self.lexer_results and self.lexer_results[-1].family == "keyword" and self.lexer_results[-1].recognized_string == "αρχή_προγράμματος":
        self.lexer_results = self.lexer.lex_states()

        self.endiamesos.genQuad('begin_block', name, '_', '_')
        self.sequence()
        self.endiamesos.genQuad('halt','_','_','_')
        self.endiamesos.genQuad('end_block',name,'_','_')

        if self.lexer_results[-1].family == "keyword" and self.lexer_results[-1].recognized_string == "τέλος_προγράμματος":
            self.lexer_results = self.lexer.lex_states()

        else:
            print(f"ERROR: Δεν υπάρχει leksi 'τέλος_προγράμματος', line {self.lexer_results[-1].line_number}")
            exit(-1)
    else:
        print(f"ERROR: Δεν υπάρχει leksi 'αρχή_προγράμματος', line {self.lexer_results[-1].line_number}")
        exit(-1)
```

3. Backpatching και διαχείριση λιστών ετικετών

- **Backpatching**: Σε δομές ελέγχου (if, while, for, do-while), χρησιμοποιούνται οι μέθοδοι makeList, merge, backPatch της EndiamesosKwdikas για να διαχειριστούν τα άλματα και τις ετικέτες που δεν είναι γνωστές εκ των προτέρων.

```
if self.lexer_results[-1].family == "keyword" and self.lexer_results[-1].recognized_string == "εάν":
    self.lexer_results = self.lexer.lex_states()

    condition = self.condition()
    self.endiamesos.backPatch(condition[0], self.endiamesos.nextQuad())
```

4. Εξαγωγή του ενδιάμεσου κώδικα σε αρχείο

- **Γράψιμο τετράδων σε αρχείο**: Στο τέλος της ανάλυσης, καλείται η μέθοδος intCode που γράφει όλες τις τετράδες του ενδιάμεσου

κώδικα σε αρχείο (intFile.int):

```
Tabnine | Edit | Test | Explain | Document
def intCode(self, x):
    lenQuads =  len(self.endiamesos.total_quads)
    for i in range(lenQuads):
        quad = self.endiamesos.total_quads[i]
        x.write(str(quad[0]))
        x.write(": ")
        x.write(str(quad[1]))
        x.write(", ")
        x.write(str(quad[2]))
        x.write(", ")
        x.write(str(quad[3]))
        x.write(", ")
        x.write(str(quad[4]))
        x.write("\n")
```

5. Δημιουργία προσωρινών μεταβλητών

- **newTemp**: Σε εκφράσεις και υπολογισμούς, χρησιμοποιείται η μέθοδος newTemp για δημιουργία προσωρινών μεταβλητών που αποθηκεύουν ενδιάμεσα αποτελέσματα.

ΠΙΝΑΚΑΣ ΣΥΜΒΟΛΩΝ

Στον μεταγλωττιστή που υλοποιήσαμε, ο πίνακας συμβόλων αποτελεί μια δυναμική δομή που παρακολουθεί και καταγράφει όλα τα συμβολικά ονόματα που εμφανίζονται στο πρόγραμμα κατά τη διάρκεια της

συντακτικής ανάλυσης και της παραγωγής ενδιάμεσου κώδικα. Για κάθε νέο scope (π.χ. πρόγραμμα, συνάρτηση, διαδικασία), δημιουργείται μια νέα εγγραφή στον πίνακα συμβόλων, η οποία περιλαμβάνει όλες τις οντότητες που δηλώνονται στο συγκεκριμένο πεδίο ορατότητας, όπως μεταβλητές, παράμετροι, προσωρινές μεταβλητές και υποπρογράμματα.

Καθώς ο συντακτικός αναλυτής διαβάζει το πρόγραμμα, προσθέτει δυναμικά νέες οντότητες στον πίνακα συμβόλων μέσω των αντίστοιχων μεθόδων, ενώ με το πέρας κάθε scope, οι σχετικές εγγραφές αφαιρούνται, διατηρώντας έτσι πάντα την πληροφορία που είναι προσβάσιμη σύμφωνα με τους κανόνες εμβέλειας της γλώσσας. Ο πίνακας συμβόλων χρησιμοποιείται τόσο για τη σημασιολογική ανάλυση, όπως ο έλεγχος ύπαρξης και τύπου μεταβλητών και παραμέτρων, όσο και για την παραγωγή ενδιάμεσου και τελικού κώδικα, παρέχοντας τα απαραίτητα offsets, nesting levels και λοιπές πληροφορίες για κάθε οντότητα.

Η δομή του πίνακα συμβόλων είναι ιεραρχική, ώστε να υποστηρίζει σωστά τα επίπεδα εμφωλευμένων scopes, και ενημερώνεται σε πραγματικό χρόνο κατά την ανάλυση του προγράμματος. Με αυτόν τον τρόπο, διασφαλίζεται η ορθότητα της διαχείρισης των ονομάτων και η σωστή παραγωγή του τελικού κώδικα, ενώ παράλληλα διευκολύνεται η υλοποίηση της σημασιολογικής ανάλυσης μέσα από τις λειτουργίες του πίνακα συμβόλων.

Ο πίνακας συμβόλων (self.pinakasSymvolwn) είναι βασικό μέλος της κλάσης Syntaktikos και χρησιμοποιείται σε όλη τη διάρκεια της συντακτικής ανάλυσης.

Σε κάθε νέο scope (π.χ. πρόγραμμα, συνάρτηση, διαδικασία) καλείται η μέθοδος **new_scope** για να δημιουργηθεί νέο επίπεδο στον πίνακα συμβόλων.

Κατά τη δήλωση μεταβλητών, παραμέτρων, προσωρινών μεταβλητών, συναρτήσεων και διαδικασιών, δημιουργούνται αντίστοιχες οντότητες (Entities) και προστίθενται στον πίνακα συμβόλων με τις μεθόδους **new_entity** και **new_argument**.

Ο πίνακας συμβόλων χρησιμοποιείται για να ελέγχεις αν μια μεταβλητή ή παράμετρος έχει ήδη δηλωθεί, να βρίσκεις offsets για μεταβλητές, να διαχειρίζεσαι nesting levels, και να παρέχεις πληροφορίες για την παραγωγή ενδιάμεσου και τελικού κώδικα.

Με το πέρας κάθε scope, καλείται η **delete_scope** για να αφαιρεθεί το αντίστοιχο επίπεδο από τον πίνακα συμβόλων, διατηρώντας έτσι μόνο τις οντότητες που είναι ορατές στο τρέχον σημείο του προγράμματος.

ΤΕΣΤ ΔΕΥΤΕΡΟΥ ΜΕΡΟΥΣ

Κατά τη διαδικασία εκτέλεσης του μεταφραστή μου, παράγεται αυτόματα το αρχείο ενδιάμεσου κώδικα με κατάληξη .int. Το αρχείο αυτό δημιουργείται από την κλάση Syntaktikos, η οποία κατά τη συντακτική ανάλυση του προγράμματος εισόδου, μετατρέπει κάθε συντακτική δομή (όπως αναθέσεις, εκφράσεις, εντολές ελέγχου ροής, κλήσεις

συναρτήσεων κ.λπ.) σε τετράδες (quads) μέσω της κλάσης EndiamesosKwdikas. Οι τετράδες αυτές αποθηκεύονται σε λίστα και, μόλις ολοκληρωθεί η ανάλυση, γράφονται σειριακά στο αρχείο intFile.int. Κάθε γραμμή του αρχείου περιέχει μία τετράδα στη μορφή: αριθμός: τελεστής, τελούμενο1, τελούμενο2, αποτέλεσμα. Με αυτόν τον τρόπο, το αρχείο .int αποτυπώνει με σαφήνεια τον ενδιάμεσο κώδικα που παράγεται από το αρχικό πρόγραμμα και αποτελεί σημαντικό εργαλείο για τον έλεγχο και την επαλήθευση της ορθής λειτουργίας του μεταφραστή μου.

```
1: begin_block, αύξηση, _, _
2: +, α, 1, T_1
3: :=, T_1, _, β
4: +, α, 1, T_2
5: :=, T_2, _, αύξηση
6: end_block, αύξηση, _, _
7: begin_block, τύπωσε_συν_1, _, _
8: +, χ, 1, T_3
9: out, T_3, _, _
10: end_block, τύπωσε_συν_1, _, _
11: begin_block, τεστ, _, _
12: :=, 1, _, α
13: *, α, α, T_4
14: -, 2, α, T_5
15: *, 2, α, T_6
16: -, T_5, T_6, T_7
17: /, T_4, T_7, T_8
18: +, 2, T_8, T_9
19: :=, T_9, _, β
20: par, α, CV, _
21: par, β, REF, _
22: par, T_10, RET, _
23: call, αύξηση, _, _
24: :=, T_10, _, γ
25: :=, 1, _, α
26: >, 2, 0, 29
27: <, 2, 0, 31
28: =, 2, 0, 33
29: >=, α, 8, 37
30: jump, _, _, 33
31: <=, α, 8, 37
32: jump, _, _, 33
```

```
38: <, β, 10, 40
39: jump, _, _, 50
40: <>, β, 22, 46
41: jump, _, _, 42
42: >=, β, 23, 44
43: jump, _, _, 49
44: <=, β, 24, 46
45: jump, _, _, 49
46: +, β, 1, T_11
47: :=, T_11, _, β
48: jump, _, _, 49
49: jump, _, _, 38
50: input, ;, _, _
51: +, β, 1, T_12
52: :=, T_12, _, β
53: -, 0, 100, T_13
54: <, β, T_13, 56
55: jump, _, _, 51
56: halt, _, _, _
57: end_block, τεστ, _, _
```

ΜΕΡΟΣ 3^{Ο}

ΤΕΛΙΚΟΣ ΚΩΔΙΚΑΣ

Η τελευταία φάση της παραγωγής κώδικα είναι η παραγωγή του τελικού κώδικα. Ο τελικός κώδικας προκύπτει από τον ενδιάμεσο κώδικα με τη βοήθεια του πίνακα συμβόλων. Συγκεκριμένα, από κάθε εντολή ενδιάμεσου κώδικα προκύπτει μία σειρά εντολών τελικού

κώδικα, η οποία για να παραχθεί ανακτά πληροφορίες από τον πίνακα συμβόλων.

Έτσι δημιουργήσαμε την κλάση Final η είναι υπεύθυνη για τη **μετατροπή του ενδιάμεσου κώδικα (quads) σε τελικό κώδικα assembly**. Παράλληλα, διαχειρίζεται την αναζήτηση οντοτήτων στον πίνακα συμβόλων και τον υπολογισμό διευθύνσεων/offsets για μεταβλητές, παραμέτρους και προσωρινές μεταβλητές.

Κύρια χαρακτηριστικά και μέλη

- **scopesList**: Λίστα με όλα τα scopes (πεδία ορατότητας) του προγράμματος, όπως δημιουργήθηκαν από τον πίνακα συμβόλων.
- **listOfAllQuads**: Λίστα με όλες τις τετράδες (quads) που παράχθηκαν από το ενδιάμεσο στάδιο.
- **asc_file**: Το αρχείο εξόδου όπου γράφεται ο τελικός assembly κώδικας.

Βασικές μέθοδοι

1. search_entity(self, n)

Αναζητά μια οντότητα (μεταβλητή, παράμετρο, συνάρτηση κλπ) με όνομα n στον πίνακα συμβόλων, ξεκινώντας από το πιο εσωτερικό scope προς τα έξω. Επιστρέφει το scope και την οντότητα αν βρεθεί, αλλιώς τερματίζει με μήνυμα λάθους.

2. gnlvcode(self, name)

Υπολογίζει τη διεύθυνση (offset) μιας μεταβλητής ή παραμέτρου που βρίσκεται σε εξωτερικό scope (δηλαδή, όταν έχουμε εμφωλευμένες

συναρτήσεις/διαδικασίες). Γράφει τις κατάλληλες εντολές assembly για να φτάσει στη σωστή διεύθυνση στη στοίβα.

3. loadvr(self, v, r)

Φορτώνει την τιμή της μεταβλητής/παραμέτρου/προσωρινής ή σταθεράς v στον καταχωρητή t{r}.

Αν το v είναι σταθερά, απλά κάνει li t{r},v.

Αν είναι μεταβλητή/παράμετρος/προσωρινή, βρίσκει το offset και κάνει lw t{r},-offset(sp/gp) ή ακολουθεί τις κατάλληλες εντολές για παραμέτρους με αναφορά ή εμφωλευμένα scopes.

4. storerv(self, r, v)

Αποθηκεύει την τιμή του καταχωρητή t{r} στη μεταβλητή/παράμετρο/προσωρινή v.

Αν το v είναι τοπική μεταβλητή, κάνει sw t{r},-offset(sp).

Αν είναι σε εξωτερικό scope, χρησιμοποιεί gnlvcode για να βρει τη σωστή διεύθυνση.

Αν είναι παράμετρος με αναφορά, κάνει τα κατάλληλα lw και sw.

5. search_list_for_call(self, i)

Βοηθητική μέθοδος που ψάχνει στη λίστα quads για να βρει το όνομα του υποπρογράμματος που καλείται σε μια εντολή call.

6. is_constant(self, v)

Επιστρέφει True αν το v είναι αριθμητική σταθερά.

7. is_valid_identifier(self, v)

Επιστρέφει True αν το v είναι έγκυρο όνομα μεταβλητής (identifier) και όχι αριθμός.

Η μέθοδος **final_code** της κλάσης Syntaktikos καλεί τις μεθόδους της Final για να μετατρέψει τις quads σε assembly.

Για κάθε quad, η Final παράγει τις αντίστοιχες εντολές assembly, χρησιμοποιώντας τις μεθόδους loadvr, storerv, gnlvcode κλπ, ώστε να διαχειρίζεται σωστά τα offsets, τα nesting levels και τις παραμέτρους.

Η Final διασφαλίζει ότι ο τελικός κώδικας που παράγεται είναι σωστός ως προς τη διαχείριση της στοίβας, των scopes και των παραμέτρων, ακόμα και σε περιπτώσεις εμφωλευμένων υποπρογραμμάτων.

ΤΕΣΤ ΤΡΙΤΟΥ ΜΕΡΟΥΣ

Η μέθοδος final_code μεταφράζει με επιτυχία τον ενδιάμεσο κώδικα σε τελικό assembly όπως παρουσιάζω και παρακάτω, καλύπτοντας όλες τις βασικές λειτουργίες της γλώσσας. Το αρχείο final.asm που παράγεται είναι πλήρες και δομημένο, έτοιμο για εκτέλεση σε κατάλληλο περιβάλλον. Η διαδικασία testing επιβεβαίωσε τη σωστή λειτουργία τόσο της μετάφρασης quads σε assembly όσο και της διαχείρισης των scopes και των μεταβλητών μέσω του πίνακα συμβόλων.

Στη συγκεκριμένη περίπτωση χρησιμοποίησα για το αρχείο test1.gre :

```
πρόγραμμα τεστ
    δήλωση α,β
    δήλωση γ

    συνάρτηση αύξηση(α,β)
        διαπροσωπεία
        είσοδος α
        έξοδος β
        αρχή_συνάρτησης
            β := α + 1 ;
            αύξηση := α + 1 { δεν μπαίνει ερωτηματικό
                              είναι τέλος block }
        τέλος_συνάρτησης
      αρχή_προγράμματος
            α := 1 ;
            β := 2 + α * α / (2 - α - (2*α));
            γ := αύξηση(α,%β);
            για α:=1 έως 8 με_βήμα 2 επανάλαβε
                εκτέλεσε τύπωσε_συν_1(α)
            για_τέλος;
            β := 1 ;
            όσο β<10 επανάλαβε
                εάν β<>22 ή [β>=23 και β<=24] τότε
                    β := β+1
                εάν_τέλος { όχι ερωτηματικό, είναι τέλος block }
            όσο_τέλος; { θέλει ερωτηματικό
                         χωρίζει εκτελέσιμες εντολές }
            διάβασε β;
            επανάλαβε
                β := β + 1
            μέχρι β<-100
τέλος_προγράμματος
```

```
1    j L7
2
3
4
5    L1:
6    sw ra,(sp)
7    L2:
8    lw t1,-12(sp)
9    li t2,1
10   add,t1,t1,t2
11   sw t1,-20(sp)
12   L3:
13   lw t1,-20(sp)
14   lw t0,-16(sp)
15   sw t1,(t0)
16   L4:
17   lw t1,-12(sp)
18   li t2,1
19   add,t1,t1,t2
20   sw t1,-24(sp)
21   L5:
22   lw t1,-24(sp)
23   lw t0,-8(sp)
24   sw t1,(t0)
25   L6:
26   lw ra,(sp)
27   jr ra
28   L7:
29   addi sp,sp,64
30   mv gp,sp
31   L8:
32   li t1,1
33   sw t1,-12(gp)
34   L9:
35   lw t1,-12(gp)
36   lw t2,-12(gp)
37   mul,t1,t1,t2
38   sw t1,-24(sp)
39   L10:
40   li t1,2
41   lw t2,-12(gp)
42   sub,t1,t1,t2
43   sw t1,-28(sp)
44   L11:
45   li t1,2
46   lw t2,-12(gp)
47   mul,t1,t1,t2
48   sw t1,-32(sp)
49   L12:
50   lw t1,-28(sp)
51   lw t2,-32(sp)
52   sub,t1,t1,t2
53   sw t1,-36(sp)
54   L13:
55   lw t1,-24(sp)
56   lw t2,-36(sp)
57   div,t1,t1,t2
58   sw t1,-40(sp)
59   L14:
60   li t1,2
61   lw t2,-40(sp)
62   add,t1,t1,t2
63   sw t1,-44(sp)
64   L15:
65   lw t1,-44(sp)
66   sw t1,-16(gp)
67   L16:
68   addi fp,sp,28
69   lw t0,-12(gp)
70   sw t0,-12(fp)
71   L17:
72   addi t0,sp,-16
73   sw t0,-20(fp)
74   L18:
75   addi t0,sp,-48
76   sw t0,-8(fp)
77   L19:
78   lw t0,-4(sp)
79   sw t0,-4(fp)
80   addi sp,sp,28
81   jal L1
82   addi sp,sp,-28
83   L20:
84   lw t1,-48(sp)
85   sw t1,-20(gp)
86   L21:
87   li t1,1
88   sw t1,-12(gp)
89   L22:
90   li t1,2
91   li t2,0
92   bgt,t1,t2,L25
93   L23:
94   li t1,2
95   li t2,0
96   blt,t1,t2,L27
97   L24:
98   li t1,2
99   li t2,0
100  beq,t1,t2,L29
101  L25:
102  lw t1,-12(gp)
103  li t2,8
104  bge,t1,t2,L33
105  L26:
106  b L29
107  L27:
108  lw t1,-12(gp)
109  li t2,8
110  ble,t1,t2,L33
```

ΤΕΛΟΣ ΑΝΑΦΟΡΑΣ