Μεταφραστές (ΜΥΥ802)

Τμήμα Μηχανικών Η/Υ και Πληροφορικής

Ακαδημαϊκό έτος:2019-2020

Κωνσταντίνος Θεμελιώτης : 2443 , cse32443

Πανεπιστήμιο Ιωαννίνων

Περιεχόμενα

Περιεχόμενα 2
1.Εισαγωγή 4
2.Η γλώσσα προγραμματισμού Minimal++ 5
-Λεκτικές μονάδες 5
-Μορφή προγράμματος 6
-Τύποι και δηλώσεις μεταβλητών 7
-Τελεστές και εκφράσεις 7
-Δομές της γλώσσας 7
-Υποπρογράμματα 9
3.Λεκτική Ανάλυση 11
-Αυτόματο: 11
-Υλοποίηση του lex() 16
-Διάγραμμα καταστάσεων 17
4.Συντακτική Ανάλυση 18
-Γραμματική της Minimal ++ 18
-Λειτουργία και υλοποίηση του Συντακτικού Αναλυτή 20
-Παράδειγμα 20
5.Ενδιάμεσος Κώδικας 23
-Υλοποίηση: 25
-Αριθμητικές Παραστάσεις 26
-Λογικές Παραστάσεις - OR 26
-Λογικές Παραστάσεις – AND 27
-Λογικές Παραστάσεις 27
-Κλήση υποπρογραμμάτων 27
-Εντολή return 28
-Εκχώρηση 28
-Δομή While 28
-Δομή if 28
-Είσοδος – Έξοδος 28
-Βοηθητικές Συναρτήσεις 29
6.Πίνακας Συμβόλων 29
-Εγγράφημα Δραστηριοποίησης 30
-Ενέργειες στον Πίνακα Συμβόλων 32

-Υλοποίηση: 32
7.Σημασιολογική Ανάλυση 34
-Υλοποίηση 35
8.Παραγωγή τελικού κώδικα 35
-Η αρχιτεκτονική MIPS: 36
-Βοηθητικές Συναρτήσεις 37
Παραγωγή από τετράδες 39
9.Εκτέλεση 43
-Παράδειγμα: 44
10.Επίλογος 47

1.Εισαγωγή

Η παρούσα αναφορά αφορά την υλοποίηση ενός μεταφραστή (compiler) γραμμένο σε γλώσσα προγραμματισμού Python, για την γλώσσα προγραμματισμού Minimal++ όπου πραγματοποιήθηκε στα πλαίσια του μαθήματος “Μεταφραστές”, το Εαρινό εξάμηνο του Ακαδημαϊκού έτους 2019-2020

Σκοπός του μεταφραστή είναι να δέχεται στην είσοδο του ένα αρχείο κώδικα στην γλώσσα Minimal++ και να παράγει στην έξοδο του ένα ισοδύναμο πρόγραμμα στην γλώσσα Assembly του επεξεργαστή MIPS.

Η ανάπτυξη του μεταφραστή υλοποιήθηκε στις εξής 4 φάσεις.

-1η φάση

-Λεκτική ανάλυση

-Συντακτική ανάλυση.

-2η φάση

-Παραγωγή ενδιάμεσου κώδικα

-3η φάση

-Σημασιολογική ανάλυση

- Πίνακας συμβόλων

4η φάση

-Παραγωγή τελικού κώδικα

Παρακάτω παρουσιάζεται η γλώσσα προγραμματισμού Minimal++

2.Η γλώσσα προγραμματισμού Minimal++

Η minimal++ είναι μια μικρή γλώσσα προγραμματισμού φτιαγμένη με βάση τις ανάγκες της προγραμματιστικής άσκησης του μαθήματος. Παρόλο που οι προγραμματιστικές της ικανότητες είναι μικρές, η εκπαιδευτική αυτή γλώσσα περιέχει πλούσια στοιχεία και η κατασκευή του μεταγλωττιστή της έχει να παρουσιάσει αρκετό ενδιαφέρον, αφού περιέχονται σε αυτήν πολλές εντολές που χρησιμοποιούνται από άλλες γλώσσες, καθώς και κάποιες πρωτότυπες. Η minimal++ υποστηρίζει συναρτήσεις και διαδικασίες, μετάδοση παραμέτρων με αναφορά και τιμή, αναδρομικές κλήσεις και άλλες ενδιαφέρουσες δομές. Επίσης, επιτρέπει φώλιασμα στη δήλωση συναρτήσεων κάτι που λίγες γλώσσες υποστηρίζουν (το υποστηρίζει η Pascal, δεν το υποστηρίζει η C). Από την άλλη όμως πλευρά, η minimal++ δεν υποστηρίζει βασικά προγραμματιστικά εργαλεία όπως η δομή for, ή τύπους δεδομένων όπως οι πραγματικοί αριθμοί και οι συμβολοσειρές. Οι παραλήψεις αυτές έχουν γίνει ώστε να απλουστευτεί η διαδικασία κατασκευής του μεταγλωττιστή, μία απλούστευση όμως που έχει να κάνει μόνο με τη μείωση των γραμμών κώδικα και όχι με τη δυσκολία κατασκευής του ή την εκπαιδευτική αξία της άσκησης. Παρακάτω παρουσιάζεται μία περιγραφή της γλώσσας:

-Λεκτικές μονάδες

Το αλφάβητο της minimal++ αποτελείται από:

• τα μικρά και κεφαλαία γράμματα της λατινικής αλφαβήτου («A»,…,«Z» και «a»,…,«z»),

• τα αριθμητικά ψηφία («0»,…,«9»),

• τα σύμβολα των αριθμητικών πράξεων («+», «-», «*», «/»),

• τους τελεστές συσχέτισης «<», «>», «=», «<=», «>=», «<>»,

• το σύμβολο ανάθεσης «:=»,

• τους διαχωριστές («;», «,», «:»)

• καθώς και τα σύμβολα ομαδοποίησης («(»,«)»,«[»,«]»,«{»,«}»)

• και διαχωρισμού σχολίων («/*»,«*/»,«//»).

Τα σύμβολα «[» και «]» χρησιμοποιούνται στις λογικές παραστάσεις όπως τα σύμβολα «(» και «)»

στις αριθμητικές παραστάσεις.

Οι δεσμευμένες λέξεις είναι:

program **declare**

if **then** **else**

while

forcase **when** **default**

not **and** **or**

function **procedure** **call** **return** **in** **inout**

input **print**

Οι λέξεις αυτές δεν μπορούν να χρησιμοποιηθούν ως μεταβλητές. Οι σταθερές της γλώσσας είναι ακέραιες σταθερές που αποτελούνται από προαιρετικό πρόσημο και από μία ακολουθία αριθμητικών ψηφίων. Τα αναγνωριστικά της γλώσσας είναι συμβολοσειρές που αποτελούνται από γράμματα και ψηφία, αρχίζοντας όμως από γράμμα. Ο μεταγλωττιστής λαμβάνει υπόψη του μόνο τα τριάντα πρώτα γράμματα. Οι λευκοί χαρακτήρες (tab, space, return) αγνοούνται και μπορούν να χρησιμοποιηθούν με οποιονδήποτε τρόπο χωρίς να επηρεάζεται η λειτουργία του μεταγλωττιστή, αρκεί βέβαια να μην βρίσκονται μέσα σε δεσμευμένες λέξεις, αναγνωριστικά, σταθερές. Το ίδιο ισχύει και για τα σχόλια, τα οποία πρέπει να βρίσκονται μέσα στα σύμβολα /* και */ ή να βρίσκονται μετά το σύμβολο // και ως το τέλος της γραμμής. Απαγορεύεται να ανοίξουν δύο φορές σχόλια, πριν τα πρώτα κλείσουν. Δεν υποστηρίζονται εμφωλευμένα σχόλια.

-Μορφή προγράμματος

program id

{

declarations

subprograms

sequence of statements

}

-Τύποι και δηλώσεις μεταβλητών

Ο μοναδικός τύπος δεδομένων που υποστηρίζει η minimal++ είναι οι ακέραιοι αριθμοί. Οι ακέραιοι αριθμοί πρέπει να έχουν τιμές από –32767 έως 32767. Η δήλωση γίνεται με την εντολή **declare.** Ακολουθούν τα ονόματα των αναγνωριστικών χωρίς καμία άλλη δήλωση, αφού γνωρίζουμε ότι πρόκειται για ακέραιες μεταβλητές και χωρίς να είναι αναγκαίο να βρίσκονται στην ίδια γραμμή. Οι μεταβλητές χωρίζονται μεταξύ τους με κόμματα. Το τέλος της δήλωσης αναγνωρίζεται με το ελληνικό ερωτηματικό. Επιτρέπεται να έχουμε περισσότερες των μία συνεχόμενες χρήσεις της **declare.**

-Τελεστές και εκφράσεις

Η προτεραιότητα των τελεστών από τη μεγαλύτερη στη μικρότερη είναι:

(1) Μοναδιαίοι λογικοί: «not»

(2) Πολλαπλασιαστικοί: «*», «/»

(3) Μοναδιαίοι προσθετικοί: «+», «-»

(4) Δυαδικοί προσθετικοί: «+», «-»

(5) Σχεσιακοί «=», «<», «>», «<>», «<=», «>=»

(6) Λογικό «and»,

(7) Λογικό «or»

-Δομές της γλώσσας

Εκχώρηση

Id := expression

Χρησιμοποιείται για την ανάθεση της τιμής μιας μεταβλητής ή μιας σταθεράς, ή μιας έκφρασης σε μία μεταβλητή.

Απόφαση if

if (condition)

statements1

[**else**

statements2]

Η εντολή απόφασης **if** εκτιμάει εάν ισχύει η συνθήκη condition και εάν πράγματι ισχύει, τότε εκτελούνται οι εντολές statements1 που το ακολουθούν. Το **else** δεν αποτελεί υποχρεωτικό τμήμα της εντολής και γι' αυτό βρίσκεται σε αγκύλη. Οι εντολές statements2 που ακολουθούν το else εκτελούνται εάν η συνθήκη condition δεν ισχύει.

Επανάληψη while

while (condition)

statements

Η εντολή επανάληψης **while** επαναλαμβάνει συνεχώς τις εντολές statements, όσο η συνθήκη condition ισχύει. Αν την πρώτη φορά που θα αποτιμηθεί η condition, το αποτέλεσμα της αποτίμησης είναι ψευδές, τότε οι statements δεν εκτελούνται ποτέ.

Επανάληψη forcase

forcase

(**when**: (condition): statements1)*

default: statements2

Η δομή επανάληψης **forcase** ελέγχει τις condition που βρίσκονται μετά τα **when.** Μόλις μία από αυτές βρεθεί αληθής, τότε εκτελούνται οι statements1 που ακολουθούν. Μετά ο έλεγχος μεταβαίνει στην αρχή της **forcase**. Αν καμία από τις **when** δεν ισχύει, τότε ο έλεγχος μεταβαίνει στη **default** και εκτελούνται οι statements2 . Στη συνέχεια ο έλεγχος μεταβαίνει έξω από την **forcase.**

Επιστροφή τιμής συνάρτησης

return (expression)

Χρησιμοποιείται μέσα σε συναρτήσεις για να επιστραφεί το αποτέλεσμα της συνάρτησης.

Έξοδος δεδομένων

print (expression)

Εμφανίζει στην οθόνη το αποτέλεσμα της αποτίμησης του expression

Είσοδος δεδομένων

input (id)

Ζητάει από τον χρήστη να δώσει μία τιμή μέσα από το πληκτρολόγιο

Κλήσης διαδικασίας

call function_name(actual_parameters)

Καλεί μία διαδικασία

-Υποπρογράμματα

Η minimal++ υποστηρίζει συναρτήσεις.

function id (formal_pars)

{

declarations

subprograms

statements

}

Η formal_pars είναι η λίστα των τυπικών παραμέτρων. Οι συναρτήσεις μπορούν να φωλιάσουν η μία μέσα στην άλλη και οι κανόνες εμβέλειας είναι όπως της PASCAL. Η επιστροφή της τιμής μιας συνάρτησης γίνεται με την **return.**

Η κλήση μιας συνάρτησης, γίνεται από τις αριθμητικές παραστάσεις σαν τελούμενο. π.χ.

D = a + f(**in** x)

όπου f η συνάρτηση και x παράμετρος που περνάει με τιμή.

Οι διαδικασίες συντάσσονται ως εξής:

procedure id (formal_pars)

{

declarations

subprograms

statements

}

Η κλήση μιας διαδικασίας, γίνεται με την **call**. π.χ.

call f(**inout** x)

όπου f η διαδικασία και x παράμετρος που περνάει με αναφορά.

Μετάδοση παραμέτρων

Η minimal++ υποστηρίζει δύο τρόπους μετάδοσης παραμέτρων:

- με σταθερή τιμή. Δηλώνεται με τη λεκτική μονάδα **in**. Αλλαγές στην τιμή της δεν επιστρέφονται στο πρόγραμμα που κάλεσε τη συνάρτηση.
- με αναφορά. Δηλώνεται με τη λεκτική μονάδα **inout**. Κάθε αλλαγή στη τιμή της μεταφέρεται αμέσως στο πρόγραμμα που κάλεσε τη συνάρτηση.

Στην κλήση μίας συνάρτησης οι πραγματικοί παράμετροι συντάσσονται μετά από τις λέξεις κλειδιά

in και **inou**t, ανάλογα με το αν περνάνε με τιμή ή αναφορά.

3.Λεκτική Ανάλυση

Η πρώτη φάση ανάπτυξης του μεταφραστή ξεκινάει με την λεκτική ανάλυση. Σκοπός του λεκτικού αναλυτή είναι να αναγνωρίσει τις λεκτικές μονάδες του πηγαίου κώδικα. Καλείται ως συνάρτηση από τον συντακτικό αναλυτή και ξεκινάει διαβάζοντας γράμμα-γράμμα τον πηγαίο κώδικα. Μόλις αναγνωρίσει λεκτική μονάδα, τότε επιστρέφει στον συντακτικό αναλυτή έναν ακέραιο που χαρακτηρίζει τη λεκτική μονάδα (π.χ. δεσμευμένη λέξη) και την ίδια την λεκτική μονάδα. Την επόμενη φορά που θα καλεστεί ο λεκτικός αναλυτής θα επιστρέψει την επόμενη λεκτική μονάδα.

Η λειτουργία του λεκτικού αναλυτή υλοποιείται ως ένα αυτόματο καταστάσεων το όποιο ξεκινάει από μία αρχική κατάσταση και αναλόγως την είσοδο του, μεταβαίνει σε μία επόμενη κατάσταση, έως ότου φτάσει σε μία τελική κατάσταση. Με το που βρεθεί σε μία τελική κατάσταση σημαίνει ότι έχει αναγνωρίσει μία λεκτική μονάδα.

Το αυτόματο καταστάσεων αναγνωρίζει:

-δεσμευμένες λέξεις
    -πχ**: if, print, for**

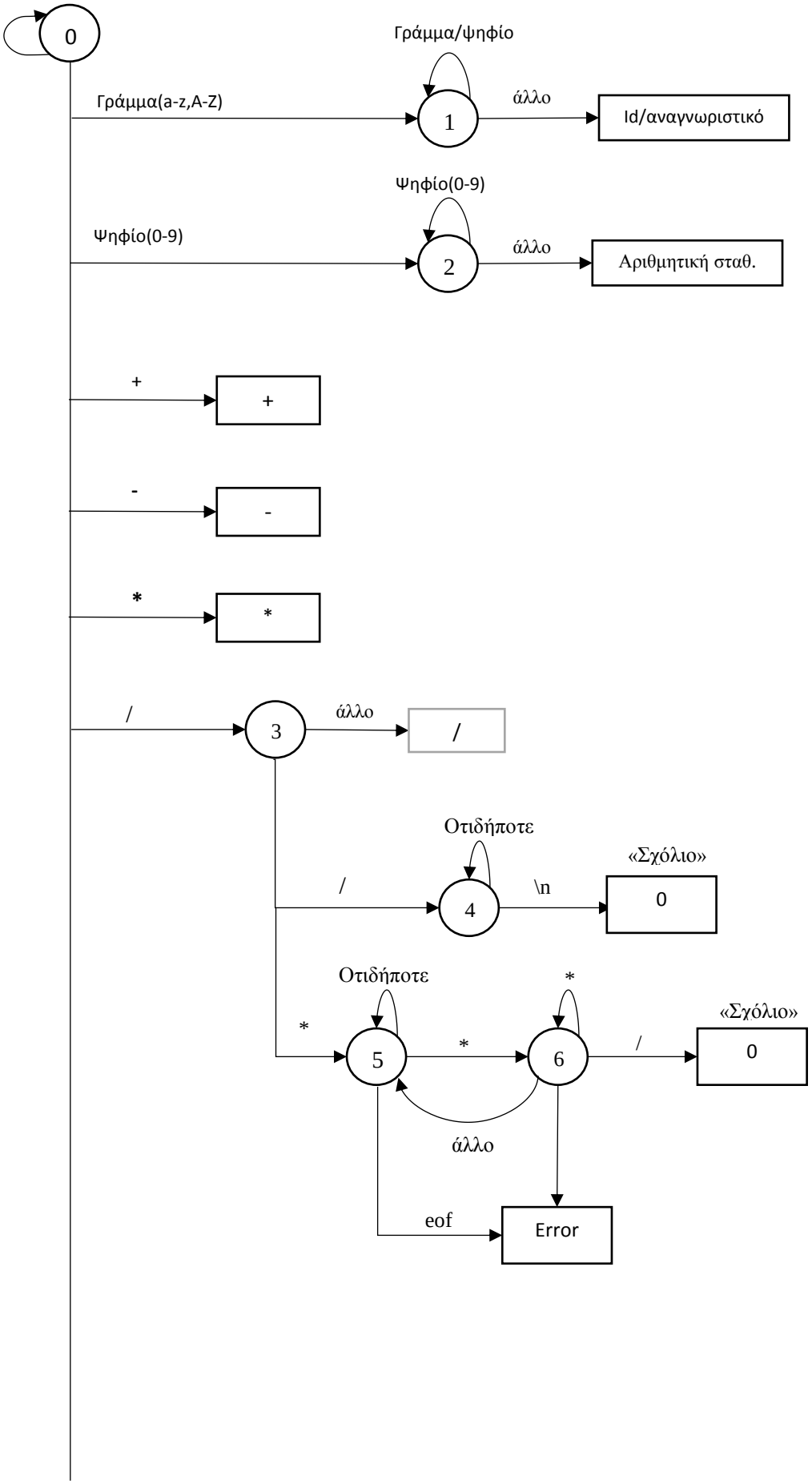
-σύμβολα της γλώσσας
    -πχ: **+, :=, =**

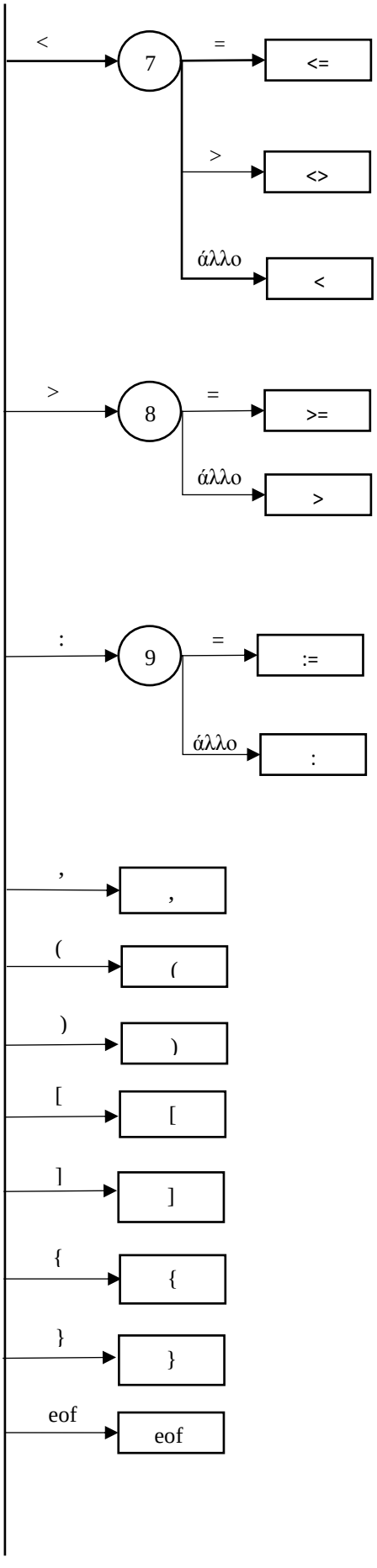
-αναγνωριστικά και σταθερές
    -πχ: **quarantine, covid19, 2020**

-λάθη
    πχ: μη άνοιγμα άγκιστρων του προγράμματος

-Αυτόματο:

Παρακάτω παρουσιάζεται το αυτόματο καταστάσεων που σχεδιάστηκε για την υλοποίηση του λεκτικού αναλυτή της γλώσσας προγραμματισμού Minimal++


0
Γράμμα/ψηφίο
Γράμμα(a-z,A-Z)
1
άλλο
Id/αναγνωριστικό
Ψηφίο(0-9)
Ψηφίο(0-9)
2
άλλο
Αριθμητική σταθ.
+
+
-
-
*
*
/
3
άλλο
/
Οτιδήποτε
«Σχόλιο»
/
4
\n
0
Οτιδήποτε
*
«Σχόλιο»
*
5
*
6
/
0
άλλο
eof
Error


< 7 = <=
> <>
άλλο <
> 8 = >=
άλλο >
: 9 = :=
άλλο :
, ,
((
))
[[
]]
{ {
} }
eof eof

-Παρουσίαση Λεκτικού Αναλυτή - Αυτομάτου:

Ο λεκτικός αναλυτής έχει υλοποιηθεί μέσα στην συνάρτηση lex(). Η κλήση της συνάρτησης lex() γίνεται μέσα από τον συντακτικό αναλυτή κάθε φορά που αυτός θα χρειαστεί την επόμενη λεκτική μονάδα ώστε να εξετάσει την συντακτική ορθότητα του αρχείου εισόδου. Η συνάρτηση lex() κάθε φορά που θα αναγνωρίζει την επόμενη λεκτική μονάδα, ενημερώνει την global μεταβλητή tokens = [“ ” ,” ”] όπου είναι μία λίστα 2 στοιχείων. Το πρώτο στοιχείο της λίστας είναι η λεκτική μονάδα που αναγνώρισε και το δεύτερο στοιχείο είναι το είδος του token είναι. Για παράδειγμα, αν το αυτόματο αναγνωρίσει το αναγνωριστικό – id “*temp*” το οποίο είναι μια μεταβλητή του αρχείου εισόδου, τότε η μεταβλητή tokens θα περιέχει με το πέρας του lex: [“temp”,”idtk”]. Πολλές φορές για την αναγνώριση μιας λεκτικής μονάδας, καταναλώνεται χαρακτήρας, ο οποίος ανήκει στην επόμενη λεκτική μονάδα. Σε αυτήν την περίπτωση, μόλις αναγνωριστεί η λεκτική μονάδα, γίνεται οπισθοδρόμηση, δηλαδή η κεφαλή ανάγνωσης του αρχείου πηγαίνει μία θέση πίσω
Πέρα από τι λεκτικές μονάδες του αρχείου εισόδου, ο λεκτικός αναλυτής διαβάζει και τα σχόλια του αρχείου εισόδου. Καθώς όμως τα σχόλια δεν χρειάζονται κατά την φάση της μεταγλώττισης, ο lex μόλις τα αναγνωρίσει τα απορρίπτει, και επιστρέφει στην αρχική κατάσταση.

-Από την κατάσταση 0 :

-Παραμένει στην κατάσταση 0 όσο διαβάζει το κενό χαρακτήρα.

-Αν διαβάσει γράμμα μεταβαίνει στην κατάσταση 1 περιμένοντας να αναγνωρίσει Id/αναγνωριστικό.

-Μεταβαίνει στην κατάσταση 2 όταν διαβάσει ψηφίο περιμένοντας να αναγνωρίσει αριθμητική σταθερά.

-Αν διαβάσει κάποιον από τους χαρακτήρες :
[“+” , “-” , “*” , “,” , “(“ , “)” , “[“ , “]” , “{“, “}”]
θα μεταβεί σε τελική κατάσταση και θα επιστρέψει τον χαρακτήρα και το token του.

-Αν διαβάσει τον χαρακτήρα “/” θα μεταβεί στην κατάσταση 3 καθώς μπορεί είναι χαρακτήρας διαίρεσης καθώς και άνοιγμα σχολίων.

-Αν διαβάσει “<” μεταβαίνει στην κατάσταση 7 καθώς μπορεί στη συνέχεια να αναγνωρίσει “<=” ή <>.

-Αν διαβάσει “>” μεταβαίνει στην κατάσταση 8 καθώς στη συνέχεια μπορεί να αναγνωρίσει “>=” .

-Αν διαβάσει “:” μεταβαίνει στην κατάσταση 9 καθώς στη συνέχεια μπορεί να αναγνωρίσει το χαρακτήρα ανάθεσης “:=”.

-Από την κατάσταση 1 :

-Όσο διαβάζει γράμμα ή ψηφίο παραμένει στην κατάσταση 1 έως ότου διαβάσει κάποιον άλλον χαρακτήρα. Όταν συμβεί αυτό, θα μεταβεί σε τελική κατάσταση και θα επιστρέψει το αναγνωριστικό id/αναγνωριστικό.

-Από την κατάσταση 2 :

-Όσο διαβάζει ψηφίο παραμένει στην κατάσταση 2 έως ότου διαβάσει κάτι άλλο. Όταν γίνει αυτό θα μεταβεί σε τελική κατάσταση και θα επιστρέψει την αριθμητική σταθερά.

-Από την κατάσταση 3 :

-Αν διαβάσει "/" σημαίνει ότι ξεκινάει σχόλιο γραμμής και μεταβαίνει στην κατάσταση 4.

-Αν διαβάσει "*" σημαίνει ότι ξεκινάει σχόλιο πολλαπλών γραμμών και μεταβαίνει στην κατάσταση 5.

-Αν διαβάσει κάποιον άλλον χαρακτήρα σημαίνει ότι έχει αναγνωρίσει τον τελεστή της διαίρεσης και τον επιστρέφει.

-Από την κατάσταση 4 :

-Στην κατάσταση αυτή, ξέρουμε ότι έχει ξεκινήσει η αναγνώριση σχολίου γραμμής. Έτσι το αυτόματο καταναλώνει κάθε χαρακτήρα που θα έρθει ως είσοδο και μόλις συναντήσει χαρακτήρα αλλαγής γραμμής, επιστρέφει στην αρχική κατάσταση δίχως να τερματίσει ή επιστρέψει κάτι ο Lex ώστε απορρίψει το σχόλιο και να βρει την επόμενη λεκτική μονάδα.

-Από την κατάσταση 5 :

-Στην κατάσταση αυτή, έχει ξεκινήσει η αναγνώριση σχολίου πολλαπλής γραμμής. Έτσι το αυτόματο καταναλώνει κάθε χαρακτήρα στην είσοδο μέχρι να ξεκινήσει το κλείσιμο του σχολίου με τον χαρακτήρα "*" και να μεταβεί στην κατάσταση 6. Αν κατά την ανάγνωση το αυτόματο συναντήσει EOF τότε σημαίνει ότι ανοίγει σχόλιο πολλαπλής γραμμής δίχως να κλείσει,το αυτόματο πηγαίνει σε κατάσταση error και η μετάφραση τερματίζει.

-Από την κατάσταση 6 :

-Στην κατάσταση αυτή έχει ξεκινήσει το κλείσιμο του σχολίου πολλαπλής γραμμής. Αν στην είσοδο έρθει ο χαρακτήρας "*" τότε το αυτόματο παραμένει στην κατάσταση 6 αναμένοντας να έρθει ο χαρακτήρας "/"

ώστε να κλείσει το σχόλιο Αν έρθει "/" τότε αυτό σημαίνει κλείσιμο σχολίου και επιστροφή του αυτόματου στην κατάσταση 0. Αν έρθει κάποιος άλλος χαρακτήρας τότε σημαίνει ότι το τελευταίο "*" που διαβάστηκε δεν ήταν για κλείσιμο του σχολίου και έτσι το αυτόματο επιστρέφει στην κατάσταση 5.Αν στη είσοδο έρθει eof δίχως να γίνει κλείσιμο του σχολίου τότε το αυτόματο πηγαίνει σε κατάσταση error και τερματίζει.

-Από την κατάσταση 7 :

-Αν έρθει στην είσοδο ο χαρακτήρας "=" τότε το αυτόματο επιστρέφει την μονάδα "<=" και επιστρέφει στον συντακτικό αναλυτή.

-Αν έρθει στην είσοδο ο χαρακτήρας ">" τότε το αυτόματο αναγνώρισε την μονάδα "<>" και επιστρέφει στον συντακτικό αναλυτή.

-Αν έρθει κάποιος άλλος χαρακτήρας τότε το αυτόματο επιστρέφει την μονάδα "<".

-Από την κατάσταση 8 :

-Αν έρθει στην είσοδο ο χαρακτήρας "=" τότε το αυτόματο επιστρέφει την μονάδα ">=".

-Αν έρθει στην είσοδο κάποιος άλλος χαρακτήρας τότε το αυτόματο επιστρέφει την μονάδα ">".

-Από την κατάσταση 9 :

-Αν έρθει στην είσοδο ο χαρακτήρας "=" τότε το αυτόματο επιστρέφει την μονάδα "=".

-Αν έρθει στην είσοδο κάποιος άλλος χαρακτήρας τότε το αυτόματο επιστρέφει τον χαρακτήρα ":".

-Υλοποίηση του lex()

Για την μετάβαση του αυτομάτου στην επόμενη κατάσταση μετά την ανάγνωση κάθε χαρακτήρα, υλοποιήθηκε ένα διάγραμμα καταστάσεων. Οι στήλες του διαγράμματος αντιπροσωπεύουν τους χαρακτήρες εισόδου του αυτόματου και οι γραμμές τις καταστάσεις του. Το περιεχόμενο του διαγράμματος είναι η κατάσταση που μεταβαίνει το αυτόματο μετά την ανάγνωση του χαρακτήρα.

-Από 0 έως 9 έχουν οριστεί οι αντίστοιχες καταστάσεις του αυτομάτου

-Ως -1 έχει οριστεί η κατάσταση σφάλματος όπου καταλήγει το αυτόματο όταν ανιχνεύεται σφάλμα

-Ως -2 έχει οριστεί ως η τελική κατάσταση

-Ως -3 έχει οριστεί η End Of File κατάσταση όπου καταλήγει το αυτόματο όταν το αυτόματο διαβάσει όλο το αρχείο

-Διάγραμμα καταστάσεων

State/Input	blank	a-z	0-9	+	-	*	/	=	<	>	:	,	;	(	)	[	]	Eof	\n	{	}
0	0	1	2	-2	-2	-2	3	-2	7	8	9	-2	-2	-2	-2	-2	-2	-3	0	-2	-2
1	-2	1	1	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2
2	-2	-2	2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2
3	-2	-2	-2	-2	-2	5	4	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	0	0	4	4
5	5	5	5	5	5	6	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
6	5	5	5	5	5	5	0	5	5	5	5	5	5	5	5	5	5	5	5	5	5
7	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2
8	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2
9	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2

4.Συντακτική Ανάλυση

Η πρώτη φάση του μεταφραστή τελειώνει με την συντακτική ανάλυση. Σκοπός του συντακτικού αναλυτή είναι να ελέγξει και να διαπιστώσει εάν το αρχείο εισόδου-πηγαίο πρόγραμμα ακολουθεί την γραμματική της γλώσσας και συνεπώς αν ανήκει στη γλώσσα. Επιπρόσθετα δημιουργεί κατάλληλο δομές μέσα από τις οποίες αργότερα θα προστεθούν εντολές όπου θα δομήσουν τις επόμενες φάσεις της μετάφρασης.
Για την κατασκευή ενός συντακτικού αναλυτή υπάρχουν ποικίλοι τρόποι. Για την γραμματική της Minimal++ θα χρησιμοποιήσουμε την συντακτική ανάλυση με αναδρομική κατάβαση η οποία βασίζεται σε γραμματική LL(1). Η γραμματική LL(1) αναγνωρίζει από αριστερά στα δεξιά, την αριστερότερη δυνατή παραγωγή και όταν δεν είναι ξεκάθαρο ποιον κανόνα θα καλέσει στην συνέχεια, μπορεί να το προσδιορίσει κοιτώντας τον επόμενο χαρακτήρα εισόδου, τον οποίο τον κοιτάει με την βοήθεια του λεκτικού αναλυτή. Τα παραπάνω θα φανούν καλύτερα με ένα παράδειγμα, αλλά ας, πρώτα παρουσιαστεί παρακάτω η γραμματική της γλώσσας.

-Γραμματική της Minimal ++

<program>	::=	**program** id **{** <block> **}**
<block>	::=	<declarations> <subprograms> <statements>
<declarations>	::=	(**declare** <varlist>;)*
<varlist>	::=	ε | id (, id)*
<subprograms>	::=	(<subprogram>)*
<subprogram>	::=	**function** id <funcbody> | **procedure** id
<funcbody>	::=	<formalpars> **{** <block> **}**
<formalpars>	::=	**(** <formalparlist> **)**
<formalparlist>	::=	<formalparitem> (, <formalparitem>)* | ε
<formalparitem>	::=	**in** id | **inout** id
<statements>	::=	<statement> | **{** <statement> (; <statement>)* **}**
<statement>	::=	<assignment-stat> |
		<if-stat> |
		<while-stat> |
		<doublewhile-stat> |
		<loop-stat> |

<exit-stat> |

<forcase-stat> |

<incase-stat> |

<call-stat> |

<return-stat> |

<input-stat> |

<print-stat>

<assignment-stat> ::= id := <expression>

<if-stat> ::= **if (**<condition>**) then** <statements> <elsepart>

<elsepart> ::= ε | **else** <statements>

<while-stat> ::= **while (**<condition>**)** <statements>

<forcase-stat> ::= **forcase**

(**when (**<condition>**)**: <statements>)*

default: <statements>

<return-stat> ::= **return** <expression>

<call-stat> ::= **call** id <actualpars>

<print-stat> ::= **print (**<expression>**)**

<input-stat> ::= **input (**id**)**

<actualpars> ::= **(** <actualparlist> **)**

<actualparlist> ::= <actualparitem> (, <actualparitem>)* | ε

<actualparitem> ::= **in** <expression> | **inout** id

<condition> ::= <boolterm> (**or** <boolterm>)*

<boolterm> ::= <boolfactor> (**and** <boolfactor>)*

<boolfactor> ::= **not [**<condition>**]** | **[**<condition>**]** |

<expression> <relational-oper> <expression>

<expression> ::= <optional-sign> <term> (<add-oper> <term>)*

<term> ::= <factor> (<mul-oper> <factor>)*

<factor> ::= constant | **(**<expression>**)** | id <idtail>

<idtail> ::= ε | <actualpars>

<relational-oper> :: = | <= | >= | > | < | <>

<add-oper> ::= + | -

<mul-oper> ::= * | /

<optional-sign> ::= ε | <add-oper>

-Λειτουργία και υλοποίηση του Συντακτικού Αναλυτή

Για κάθε έναν από τους κανόνες της παραπάνω γραμματικής, στον Compiler υλοποιείται μία συνάρτηση υποπρόγραμμα . Όταν συναντάται μη τερματικό σύμβολο καλείται η αντίστοιχη συνάρτηση.
Όταν συναντάται τερματικό σύμβολο τότε :

-Αν ο λεκτικός αναλυτής – lex() επιστρέφει λεκτική μονάδα που αντιστοιχεί στο τερματικό σύμβολο αυτό έχουμε αναγνωρίσει επιτυχώς την λεκτική μονάδα.

-Αν ο λεκτικός αναλυτής – lex() δεν επιστρέψει τη λεκτική μονάδα που περιμένει ο συντακτικός αναλυτής, έχουμε λάθος και καλείται ο διαχειριστής σφαλμάτων και τερματίζει η μετάφραση

Όταν αναγνωριστεί και η τελευταία λέξη του πηγαίου αρχείου εισόδου, τότε η συντακτική ανάλυση έχει ολοκληρωθεί και το πρόγραμμα ανήκει στην γλώσσα.

-Παράδειγμα

Παρακάτω παρουσιάζεται ένα παράδειγμα υλοποίησης κάποιων από τους πρώτους κανόνες της γραμματικής της γλώσσας που υλοποιεί ο μεταφραστής.

Κανόνας → <program> ::= **program** id **{** <block> **}**

Υλοποίηση:

```
def program() :
    if tokens[1] == 'programtk' : #Πρώτη λεκτική μονάδα
        lex()
        if tokens[1] == 'idtk' :  #Αναγνωριστικό του προγράμματος
            lex()
            if tokens[1] == '{tk' : #Άνοιγμα αγκίστρων προγράμματος
                lex()
                block()
                if tokens[1] == '}tk' :
                    lex()
                    if tokens[1] == 'eoftk' :
                        print("")
                    else :
                        print("Some Kind Of Error")
                        sys.exit(1)
                else :
                    print(tokens[0],tokens[1])
                    print("Syntax error: } is missing to close program
bracklet ")
                    sys.exit(1)
            else:
                print("Syntax error: expected { to open keyword 'progra
m' brackelt ")
                sys.exit(1)
        else :
            print("Syntax error: program ID is missing")
            sys.exit(1)
    else :

        print("Syntax error: 'program' statement is missing ")
        sys.exit(1)
```

Η συντακτική ανάλυση ξεκινάει με την κλήση του πρώτου κανόνα της γραμματικής <program>. Όπως αναφέρθηκε παραπάνω θα χρησιμοποιήσουμε την συντακτική ανάλυση με αναδρομική κατάβαση η οποία βασίζεται σε γραμματική LL(1). Η γραμματική LL(1) αναγνωρίζει από αριστερά στα δεξιά, την αριστερότερη δυνατή παραγωγή και όταν δεν είναι ξεκάθαρο ποιον κανόνα θα καλέσει στην συνέχεια, μπορεί να το προσδιορίσει κοιτώντας τον επόμενο χαρακτήρα εισόδου, τον οποίο τον κοιτάει με την βοήθεια του λεκτικού αναλυτή Η πρώτη λεκτική μονάδα που αναμένεται να επιστρέψει ο lex() από αριστερά προς δεξιά είναι η “program”. Αν διαβαστεί σωστά, συνεχίζει στην επόμενη λεκτική μονάδα που είναι το αναγνωριστικό του προγράμματος. Συνεχίζοντας με την βοήθεια του λεκτικού αναλυτή, η επόμενη μονάδα αναμένεται είναι το άγκιστρο “{“ που ανοίγει το μπλοκ του προγράμματος. Εφόσον έχουν αναγνωριστεί όλες οι παραπάνω μονάδες επιτυχώς,

καλείται ο κανόνας μπλοκ, ο οποίος με την σειρά του θα καλέσει του επόμενους κανόνες και ύστερα αυτοί τους επόμενους για όσο διαβάζονται οι αντίστοιχες λεκτικές μονάδες και ικανοποιούνται οι κανόνες. Επιστρέφοντας επιτυχώς πίσω στον αρχικό κανόνα, ο συντακτικός αναλυτής περιμένει να αναγνωρίσει το κλείσιμο του αρχικού μπλοκ με τον χαρακτήρα "}". Αν συμβεί αυτό και στην συνέχεια δεν διαβαστεί άλλος χαρακτήρας σημαίνει πως το πρόγραμμα εισόδου πέρασε την συντακτική ανάλυση και συνεπώς ανήκει στην γλώσσα. Αν κατά την ανάγνωση των λεκτικών μονάδων, ο συντακτικός αναλυτής συναντήσει μη αναμενόμενη μονάδα -πχ περιμένει το αναγνωριστικό του προγράμματος και διαβάσει άνοιγμα άγκιστρων- τότε σημαίνει πως το πρόγραμμα δεν περνάει την συντακτική ανάλυση καθώς δεν ικανοποιείται ο κανόνας (στην συγκεκριμένη περίπτωση το πρόγραμμα δεν έχει αναγνωριστικό) και η διαδικασία της μετάφρασης τερματίζει

Κανόνας → <block> ::= <declarations> <subprograms> <statements>

Υλοποίηση :

```
def block() :
    declarations()
    subprograms()
    statements()
```

Ο κανόνας <block> είναι ο δεύτερος κανόνας που καλείται και καλείται μέσα από τη συνάρτηση program. Αρχικά καλεί την συνάρτηση declarations όπου είναι υπεύθυνη για την αναγνώριση και δήλωση των μεταβλητών στο πρώτο σκέλος του προγράμματος. Αφού επιστρέψει επιτυχώς, στην συνέχεια καλείται η subprograms η οποία είναι υπεύθυνη για τον εντοπισμό τυχόν υποπρογραμμάτων στο 2^{ο} σκέλος του προγράμματος και την συντακτική τους ορθότητα. Επιστρέφοντας καλείται η statements όπου είναι υπεύθυνη για αναγνώριση και ανάλυση των επόμενων συντακτικών δομών του προγράμματος.

5.Ενδιάμεσος Κώδικας

Η δεύτερη φάση της ανάπτυξης του μεταφραστή, έρχεται με τον ενδιάμεσο κώδικα. Σε αυτή τη φάση, μεταφράζεται το αρχικό πρόγραμμα σε μία ενδιάμεση γλώσσα η οποία είναι κατώτερου επιπέδου από την αρχική αλλά ψηλότερου επιπέδου από την τελική. Σκοπός της παραγωγής του ενδιάμεσου κώδικα είναι η διευκόλυνση της διαδικασίας της μετάφρασης στην τελική γλώσσα όπως και επίσης η διευκόλυνση της διαδικασίας βελτιστοποίησης του κώδικα. Με το πέρας της φάσης, δημιουργείται ένα δυαδικό αρχείο το οποίο περιέχει τον ενδιάμεσο κώδικα και ένα αρχείο C το οποίο έχει φτιαχτεί με βάση τον ενδιάμεσο κώδικα που έχει δημιουργηθεί.

Ο ενδιάμεσος κώδικας είναι ένα σύνολο από τετράδες. Αυτές οι τετράδες ουσιαστικά περιγράφουν τις αριθμητικές πράξεις, τα άλματα, τις συνθήκες, τις επαναλήψεις και γενικά όλες τις «ενέργειες» του προγράμματος. Οι τετράδες αποτελούνται από έναν τελεστή και τρία τελούμενα. Για παράδειγμα μία τετράδα που θα παραχθεί για το άθροισμα 2 αριθμών είναι η :

+, a, b, t_1

Όπου ο πρώτος τελεστής είναι ο τελεστής της πρόσθεσης, τα τελούμενα a, b είναι οι μεταβλητές που προστίθενται και το τρίτο τελούμενο είναι μια προσωρινή μεταβλητή όπου αποθηκεύεται το αποτέλεσμα της πράξης.

Κάθε μία από τις τετράδες που δημιουργούνται έχουν μπροστά έναν μοναδικό αριθμό που την χαρακτηρίζει. Μόλις εκτελεστεί μία τετράδα, τότε εκτελείται η αμέσως επόμενη με τον μεγαλύτερο αριθμό που την χαρακτηρίζει, εκτός κι αν η τετράδα που εκτελείται υποδείξει αλλιώς (πχ μία εντολή jump).

Υπάρχουν διαφορετικές μορφές τετράδων αναλόγως την ενέργεια. Αυτές είναι :

Τελεστές αριθμητικών πράξεων:

Τετράδες της μορφής :

op, x, y, z

όπου op ένας από τους τελεστές : +, -, *, /

όπου x,y ονόματα μεταβλητών η αριθμητικές σταθερές

όπου z μεταβλητή που τοποθετείται το αποτέλεσμα

Πχ η τετράδα του z = x * y → *, x, y, z

Τελεστής εκχώρησης :

Τετράδες της μορφής :

:=, x, _, z

Η τιμή του x εκχωρείται στο z

Τελεστής άλματος χωρίς συνθήκη :

jump, _, _, z

μεταπήδηση χωρίς όρους στη θέση z

Τελεστής άλματος με συνθήκη :

relop, x, y, z

όπου relop ένας από τους τελεστές :

=, >, <, <>, >=, <=

μεταπήδηση στη θέση z αν ισχύει η x relop y

Αρχή και τέλος ενότητας

begin_block, name, _, _

αρχή υποπρογράμματος με όνομα name

end_block, name, _, _

τέλος υποπρογράμματος με όνομα name

halt, _, _, _

τερματισμός προγράμματος

Συνάρτησεις – Διαδικασίες :

par, x, m, _

όπου x παράμετρος και m ο τρόπος μετάδοσης :

CV : μετάδοση με τιμή

REF : μετάδοση με αναφορά

RET : επιστροφή τιμής συνάρτησης

call, name, _, _

κλήση συνάρτησης name

ret, x, _, _

επιστροφή τιμής συνάρτησης

-Υλοποίηση:

Οι διαφορετικού τύπου τετράδες που παρουσιάστηκαν παραπάνω, κατά την μετάφραση δημιουργούνται μέσα στον συντακτικό αναλυτή στις ανάλογες συναρτήσεις. Πριν όμως γίνει αυτό, υλοποιούνται κάποιες βοηθητικές συναρτήσεις – υπορουτίνες που είναι τα βασικά εργαλεία για την δημιουργία τους. Αυτές οι συναρτήσεις είναι οι:

- def nextQuad() :
  -Επιστρέφει τον αριθμό της επόμενης τετράδας που πρόκειται να παραχθεί

- def genQuad(op,x,y,z) :
  -Δημιουργεί την επόμενη τετράδα που δέχεται σαν όρισμα και την προσθέτει στην λίστα των τετράδων

- def newTemp() :
  -Δημιουργεί και επιστρέφει μια νέα προσωρινή μεταβλητή της μορφής T_1, T_2, T_3 …
  -Επίσης δημιουργεί ένα αντικείμενο τύπου Entity το οποίο βέβαια ανήκει στην τελευταία φάση παραγωγής -την παραγωγή του τελικού κώδικα.

- def emptyList() :
  -Δημιουργεί μια κενή λίστα ετικετών τετράδων

- def makeList(x) :
  -Δημιουργεί μια λίστα ετικετών τετράδων που περιέχει μόνο το x

- def merge(list_1,list_2) :
  -Δημιουργεί μια λίστα ετικετών τετράδων από την συνένωση των λιστών list_1, list_2 που δέχεται σαν ορίσματα.

- def backPatch(list,z) :
  -Η λίστα list που δέχεται ως όρισμα, αποτελείται από τετράδες των οποίων το τελευταίο τελούμενο δεν είναι συμπληρωμένο. Η συνάρτηση αυτή επισκέπτεται τις τετράδες αυτές και τις συμπληρώνει με την ετικέτα z.

Πέρα από τις συναρτήσεις χρησιμοποιούνται και 2 global μεταβλητές η ***quads_list*** όπου είναι η λίστα που περιέχει όλες τις τετράδες του ενδιάμεσου κώδικα και η

temp_var_counter που είναι μετρητής των προσωρινών μεταβλητών που δημιουργούνται.

Συνεχίζοντας, για την παραγωγή των τετράδων του ενδιάμεσου κώδικα χρησιμοποιούμε τις παραπάνω συναρτήσεις, οι οποίες καλούνται μέσα από τον συντακτικό αναλυτή. Για να καθοριστεί πού θα γίνει η κλήση των συναρτήσεων και να παραχθούν σωστά οι τετράδες χρησιμοποιούμε τους παρακάτω κανόνες ανάλογα την παράσταση.

-Αριθμητικές Παραστάσεις :

E ->T1(+ T2{P1})* {P2}

{P1}: w = newTemp()

genQuad("+",T1.place,T2.place,w)

T1.place = w

{P2}:

T ->F1(x F2{P1})* {P2}

{P1}: w =newTemp()

genQuad("x",F1.place,F2.place,w)

F1.place=w

{P2}: T.place=F1.place

F->(E){P1}

{P1}: F.place=E.place

F->id{P1}

{P1}:F.place=id.place

-Λογικές Παραστάσεις - OR

B->Q1{P1}(or{P2}Q2{P3})*

{P1}: B.true=Q1.true

B.false=Q1.false

{P2}: backpatch(B.false,nextQuad())

{P3}: B.true=merge(B.true,Q2.true)

B.false=Q2.false

-Λογικές Παραστάσεις – AND

Q->R1{P1}(and{P2}R2{P3})*

{P1}: Q.true=R1.true

Q.false=R1.false

{P2}: backpatch(Q.true,nextquad())

{P3}: Q.false = merge(Q.false,R2.false)

Q.true=R2.true

-Λογικές Παραστάσεις:

R->(B){P1}

{P1}: R.true=B.true

R.false=B.false

P->E1 relop E2{P1}

{P1}: R.true=makelist(nextquad())

genQuad(relop,E1.place,E2.place,"_")

R.false=makelist(nextQuad())

genQuad("jump", "_","_","_")

-Κλήση υποπρογραμμάτων:

Έστω κλήση διαδικασίας : call add(in a, in b)

genQuad("par",a, "CV", "_")

genQuad("par",b, "REF","_")

genQuad("call",add,"_","_")

Έστω κλήση συνάρτησης: result = add(in a, inout b) :

genQuad("par", a, "CV", "_")

genQuad("par", b, "REF, "_")

w=newTemp()

genQuad("par", w, "RET", "_")

genQuad("call", "add", "_", "_")

-Εντολή return

S-> return(E){P1}

{P1}: genquad(“retv:,E.place,”_”,”_”)

-Εκχώρηση :

S-> id:= E{P1};

{P1}: genQuad(“:=”, E.place, “_”, id)

-Δομή While :

S-> while{P1} B do {P2} S1 {P3}

{P1}: Bquad:=nextQuad())

{P2}: backpatch(B.true,nextQuad())

{P3}: genQuad(“jump”, “_”, “_”, Bquad)

Backpatch(B.false,nextQuad())

-Δομή if:

S-> if B then {P1} S1 {P2} TAIL {P3}

{P1}: backpatch(B.true,nextQuad())

{P2}: ifList=makelist(nextQuad())

genQuad(“jump”,”_”,”_”,”_”,”_”)

backpatch(B.false,nextQuad())

{P3}: backpatch(ifList,nextQuad())

TAIL-> else S2 | TAIL -> ε

-Είσοδος –Έξοδος:

S->input(id){P1}

{P1}: genQuad(“inp”,id.place,”_”,”_”)

S->print(E){P2}

{P2}: genQuad(“out”,E.place,”_”,”_”)

-Βοηθητικές Συναρτήσεις

Εφόσον πλέον έχουν δημιουργηθεί οι τετράδες του ενδιάμεσου κώδικα , χρησιμοποιούνται άλλες 3 βοηθητικές συναρτήσεις με σκοπό την δημιουργία του δυαδικού αρχείου που περιέχει τον ενδιάμεσο κώδικα και τον αρχείου C που περιέχει τον ισοδύναμο κώδικα σε C.

```
def createInt_File() :
```

Δημιουργεί ένα δυαδικό αρχείο και «γράφει» την λίστα quads_list που περιέχει τις τετράδες σε αυτό

```
def createC_File():
```

Δημιουργεί ένα αρχείο C, γράφει τις δομές, καλεί την output_Var(c_File) οι οποία ε αρχικοποιεί τις μεταβλητές του προγράμματος και συνεχίζει γράφοντας τις τετράδες δημιουργώντας έτσι το αρχείο C

```
def output_Var(c_file):
```

Βοηθητική συνάρτηση όπου αναγνωρίζει τις μεταβλητές μέσα από τις τετράδες, δημιουργεί μια λίστα μεταβλητών varlist και τις γράφει - αρχικοποιεί στο αρχείο.

def get(quad):

Βοηθητική συνάρτηση η οποία δημιουργεί την δομή στην γλώσσα C αναλόγως την τετράδα που θα αναγνωρίζει και την επιστρέφει στην createC_File η οποία και την καλεί. Έστω ότι αναγνωρίσει την τετράδα out τότε την μεταφράζει στην δομή της printf της C.

6.Πίνακας Συμβόλων

Η 3ή φάση της ανάπτυξης του μεταφραστή είναι ο πίνακας συμβόλων. Ο πίνακας συμβόλων είναι μια δομή, όπου αποθηκεύεται πληροφορία σχετικά με τα σύμβολα του προγράμματος (μεταβλητές και συναρτήσεις).

Οι εγγραφές - οντότητες του πίνακα ορίζοντα ως εξής:

- Εγγραφή Οντότητας – Record Entity.

Αυτή η οντότητα μπορεί να είναι:

-Μεταβλητή , με χαρακτηριστικά:

Name
Type
offset : Απόσταση από την αρχή του εγγραφήματος δραστηριοποίησης

-Συνάρτηση, με χαρακτηριστικά

Name
Type
startQuad(ετικέτα της πρώτης τετράδας του κώδικα της συνάρτησης
argument : λίστα παραμέτρων
framelength : μήκος εγγραφήματος δραστηριοποίησης

-Σταθερά, με χαρακτηριστικά :

Name
value

-Παράμετρος, με χαρακτηριστικά:

Name
value

-Προσωρινή μεταβλητή, με χαρακτηριστικά:

Name
offset

Εγγραφή εμβέλειας – Record Scope:

Χαρακτηριστικά:

List Entity(λίστα από Entities)

Int nestingLevel : βάθος φωλιάσματος

Εγγραφή Ορίσματος – Record Argument

Int parMode : πέρασμα με αναφορά η με τιμή

Int type: τύπος μεταβλητής

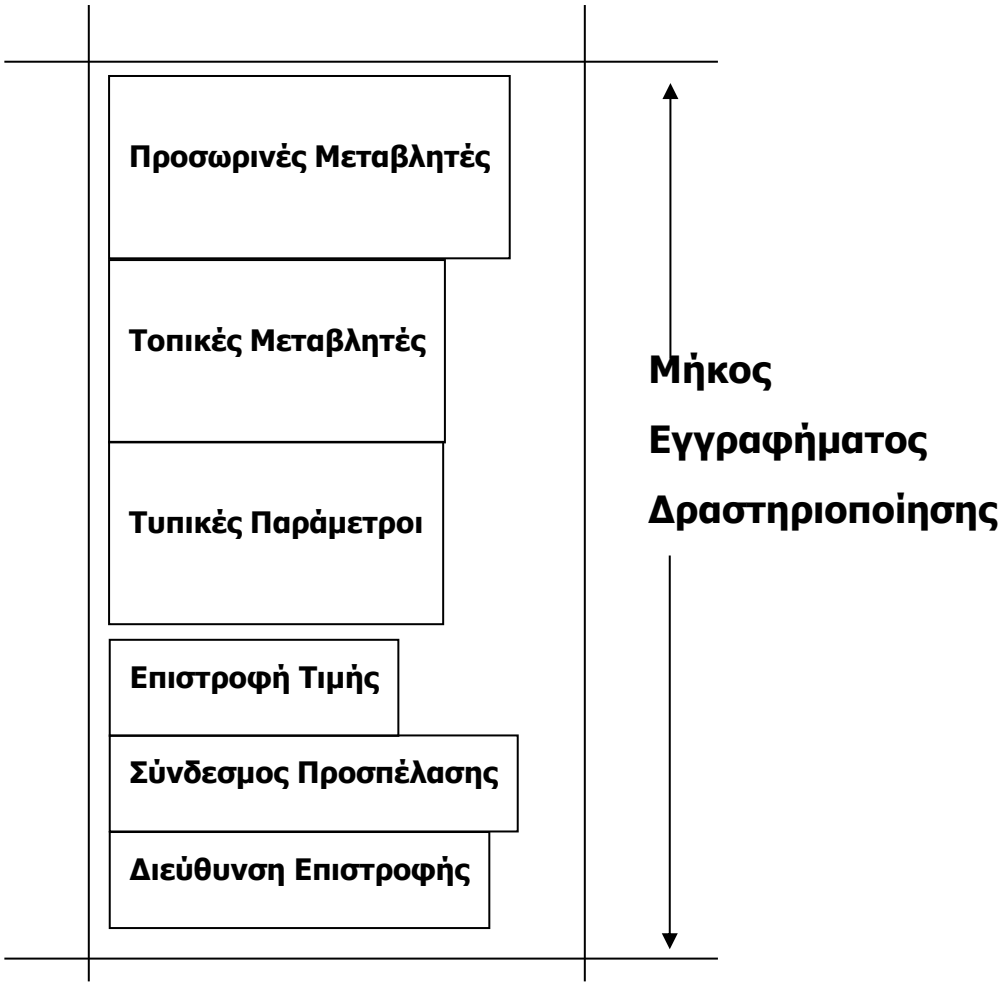
-Εγγράφημα Δραστηριοποίησης

Δημιουργείται για κάθε συνάρτηση από αυτήν που την καλεί

Όταν αρχίζει η εκτέλεση της συνάρτησης ο δείκτης στοίβας μεταφέρεται στην αρχή του εγγρασήματος δραστηριοποίησης

Περιέχει πληροφορίες που χρησιμέυουν για την εκτέλεση και τον τερματισμό της συνάρτησης καθώς και πληροφορίες που σχετίζονται με τις μεταβλητές που χρησιμοποιεί

Όταν τερματίζεται η συνάρτηση ο χώρος που καταλαμβάνει το εγγράφημα δραστηριοποίησης επιστρέφεται στο σύστημα



Διεύθυνση επιστροφής: η διεύθυνση στην οποία θα μεταβεί η ροή του προγράμματος όταν ολοκληρωθεί η εκτέλεση της συνάρτησης

Σύνδεσμος Προσπέλασης: δείχνει στο εγγράφημα δραστηριοποίσης που πρέπει να αναζητηθούν μεταβλητές οι οποίες δεν είναι τοπικές αλλά η συνάρτηση έχει δικαίωμα να χρησιμοποιήσει

Επιστροφή τιμής: η διεύθυνση στην οποία θα γραφεί το αποτέλεσμα της συνάρτησης όταν αυτό υπολογιστεί

-Ενέργειες στον Πίνακα Συμβόλων**:**

-Προσθήκη νέου Scope: όταν ξεκινάμε τη μετάφραση μιας νέας συνάρτησης

-Διαφραφή Scope: όταν τελειώνουμε τη μετάφραση μιας συνάρτησης - με τη διαγραφή διαγράφουμε την εγγραφή (record) του Scope και όλες τις λίστες με τα Entity και τα Argument που εξαρτώνται από αυτήν

-Προσθήκη νέου Entity

όταν συναντάμε δήλωση μεταβλητής

όταν δημιουργείται νέα προσωρινή μεταβλητή

όταν συναντάμε δήλωση νέας συνάρτησης

όταν συναντάμε δήλωση τυπικής παραμέτρου συνάρτησης

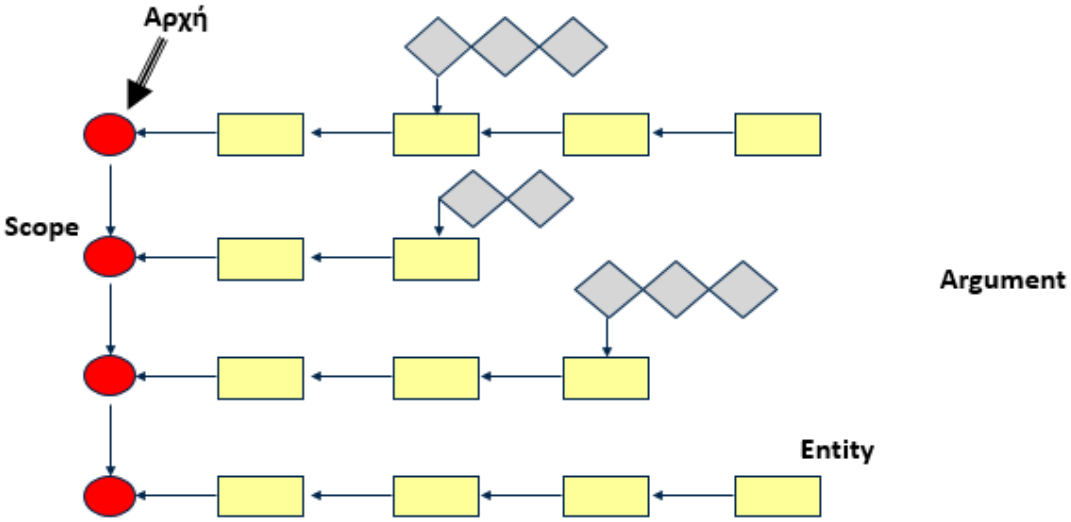
Προσθήκη νέου Argument:

όταν συναντάμε δήλωση τυπικής παραμέτρου συνάρτησης

Αναζήτηση: μπορεί να αναζητηθεί κάποιο entity με βάση το όνομά του. Η αναζήτηση ενός entity γίνεται ξεκινώντας από την αρχή του πίνακα και την πρώτη του γραμμή. Αν δε βρεθεί πηγαίνουμε στην επόμενη γραμμή έως ότου βρεθεί το entity ή τελειώσουν όλα τα entities οπότε επιστρέφουμε και μήνυμα λάθους. Αν με το ζητούμενο όνομα υπάρχει πάνω από ένα entity τότε επιστρέφουμε το πρώτο που θα συναντήσουμε

-Υλοποίηση:

Όπως προκύπτει και από τα παραπάνω, ο πίνακας συμβόλων είναι μία λίστα από Scopes όπου κάθε Scope αποτελείται από Entities, όπως παρουσιάζεται και στο παρακάτω σχήμα :



Οι οντότητες Entity και Argument έχουν υλοποιηθεί ως κλάσεις όπου τα αντικείμενα αρχικοποιούνται κάθε φόρα κατά της δημιουργία αντικειμένων με τον default constructor __init__ .

Στην κλάση Entity κατά την αρχικοποίηση αναλόγως το αν η εγγραφή που πρόκειται να δημιουργηθεί είναι συνάρτηση, μεταβλητή η παράμετρος, αρχικοποιούνται και τα ανάλογα πεδία όπως αναφέρονται και παραπάνω.

Στην κλαση Scope υλοποιούνται 2 συναρτήσεις. Η add_Entity(self,entity) η οποία προσθέτει ένα αντικείμενο τύπου entity στην λίστα με τα entities και την add_Argument(self, par_mode), όπου προσθέτει σε ένα νέο argument στην λίστα των argument που έχει ένα entity.

Η δημιουργία των παραπάνω γίνεται με κάποιες βοηθητικές συναρτήσεις.
Αυτές είναι:

def add_new_Scope() : Προσθήκη νέου Scope

def delete_Scope() : Διαγραφή Scope

def create_new_entity (name, Type, par_mode) : Προσθήκη νέου Entity

def add_new_argument(par_mode) : Προσθήκη νέου argument

def search_Entity(name) :Αναζήτηση Entity

def get_main_offset() : Επιστροφή Main offset

def get_Current_Scope() : Επιστρέφει το τρέχον Scope

def is_current_scope(name) : Ελέγχει αν ένα Entity ανήκει στο τρέχον Scope

def set_start_quad(quad) : Ορίζει την αρχική τετράδα

def set_FrameLength() : Ορίζει το framelength

Η κλήση των παραπάνω συναρτήσεων γίνεται μέσα από τον συντακτικό αναλυτή στις περιπτώσεις που αναφέρονται παραπάνω στις ενέργειες του πίνακα συμβόλων

7.Σημασιολογική Ανάλυση

Η 3η φάση ανάπτυξης του μεταφραστή τελειώνει με την σημασιολογική ανάλυση. Σκοπός της σημασιολογικής ανάλυσης είναι, πέρα από τον συντακτικό έλεγχο για τον εντοπισμό συντακτικών σφαλμάτων, ο εντοπισμός σημασιολογικών σφαλμάτων. Αυτά μπορεί να είναι η δήλωση 2 φορές της ίδιας μεταβλητής σε ένα μπλοκ, η μη επιστροφή τιμής από συνάρτηση κλπ.

Στην παρούσα ανάλυση έχουν υλοποιηθεί οι εξής περιπτώσεις:

- Δήλωση μεταβλητής μόνο μια φορά σε μια εμβέλεια – Κάθε μεταβλητή πρέπει να δηλώνεται μια φορά σε κάθε εμβέλεια

- Έλεγχος για μη δηλωμένη μεταβλητή – Κάθε μεταβλητή του προγράμματος πρέπει πρώτα να δηλωθεί με την εντολή declare

- Ύπαρξη της εντολής return μέσα σε συνάρτηση – Κάθε συνάρτηση(function) στο τέλος της πρέπει να έχει την εντολή return

- Μη ύπαρξη της εντολής return σε διαδικασία – Απαγορεύεται η ύπαρξη εντολής return μέσα σε μία διαδικασία (procedure).

-Υλοποίηση

Για την υλοποίηση των παραπάνω περιπτώσεων δημιουργήθηκαν οι εξής συναρτήσεις:

-Δήλωση μεταβλητής μόνο μια φορά σε μια εμβέλεια:

```
def is_declared (var)
```

- Έλεγχος για μη δηλωμένη μεταβλητή :

```
def is_undeclared(var) :
```

- Ύπαρξη της εντολής return μέσα σε συνάρτηση :

Boolean Μεταβλητή *has_return* η οποία αν κατά την διάρκεια της μετάφρασης μιας συνάρτησης εντοπίσει την μονάδα return γίνεται αληθής.

-Μη ύπαρξη της εντολής return σε διαδικασία :

Boolean Μεταβλητή *has_return* η οποία αν κατά την διάρκεια της μετάφρασης μιας διαδικασίας εντοπίσει την μονάδα return γίνεται αληθής.

8.Παραγωγή τελικού κώδικα

Η ανάπτυξη του μεταφραστή για την γλώσσα Minimal++ τελειώνει με την Παραγωγή του τελικού κώδικα. Σκοπός αυτής της φάσης είναι να παραχθεί ισοδύναμο πρόγραμμα με το αρχικό πρόγραμμα εισόδου στην γλώσσα assembly του επεξεργαστή MIPS

Για να παράγουμε τον τελικό κώδικα θα εκμεταλλευτούμε τον ενδιάμεσο κώδικα. Για κάθε μία εντολή του ενδιάμεσου κώδικα, για κάθε μία τετράδα θα παράγουμε αντίστοιχες τις αντίστοιχες εντολές του τελικού κώδικα.

Κύριες ενέργειες στη φάση αυτή είναι :

-Η απεικόνιση τον μεταβλητών στην μνήμη

-Το πέρασμα παραμέτρων και η κλήση συναρτήσεων

-Η αρχιτεκτονική MIPS:

Χρήσιμοι καταχωρητές:

-καταχωρητές προσωρινών τιμών:	$t0...$t7
-καταχωρητές οι τιμές των οποίων διατηρούνται ανάμεσα σε κλήσεις συναρτήσεων:	$s0...$s7
-καταχωρητές ορισμάτων:	$a0...$a3
-καταχωρητές τιμών:	$v0,$v1
-stack pointer	$sp
-frame pointer	$fp
- return address	$ra

Χρήσιμες εντολές για αριθμητικές:

add $t0,$t1,$t2	t0=t1+t2
sub $t0,$t1,$t2	t0=t1-t2
mul $t0,$t1,$t2	t0=t1*t2
div $t0,$t1,$t2t	0=t1/t2

Χρήσιμες εντολές για μετακίνηση δεδομένων:

move $t0,$t1	t0=t1	μεταφορά ανάμεσα σε καταχωρητές
li $t0, value	t0=value	σταθερά σε καταχωρητή
lw $t1,mem	t1=[mem]	περιεχόμενο μνήμης σε καταχωρητή
sw $t1,mem	[mem]=t1	περιεχόμενο καταχωρητή σε μνήμη
lw $t1,($t0)	t1=[t0]	έμμεση αναφορά με καταχωρητή
sw $t1,-4($sp)	t1=[$sp-4]	έμμεση αναφορά με βάση τον $sp

Χρήσιμες εντολές για άλματα:

b label	branch to label
beq $t1,$t2,label	jump to label if $t1=$t2

blt $t1,$t2,label	jump to label if $t1<$t2
bgt $t1,$t2,label	jump to label if $t1>$t2
ble $t1,$t2,label	jump to label if $t1<=$t2
bge $t1,$t2,label	jump to label if $t1>=$t2
bne $t1,$t2,label	jump to label if $t1<>$t2

Χρήσιμες εντολές για την χρήση συναρτήσεων:

Jlabel	jump to label
Jallabel	κλήση συνάρτησης
jr$ra	άλμα στη διεύθυνση που έχει ο καταχωρητής στο παράδειγμα είναι ο $raπου έχει τηνδιεύθυνση επιστροφής συνάρτησης

-Βοηθητικές Συναρτήσεις

Χρησιμοποιώντας λοιπόν τις τετράδες του ενδιάμεσου κώδικα και τις παραπάνω εντολές δομούμε τον τελικό κώδικα. Πριν όμως γίνει αυτό θα χρειαστούν ακόμα κάποιες βοηθητικές συναρτήσεις. Αυτές είναι:

def gnvlcode(v) :

Μεταφέρει στον $t0 την διεύθυνση μιας μη τοπικής μεταβλητής από τον πίνακα συμβόλων - βρίσκει πόσα επίπεδα επάνω βρίσκεται η μη τοπική μεταβλητή και μέσα από τον σύνδεσμο προσπέλασης την εντοπίζει

def loadvr(v, r) :

μεταφορά δεδομένων στον καταχωρητή r - η μεταφορά μπορεί να γίνει από τη μνήμη (στοίβα) ή να εκχωρηθεί στο r μία σταθερά

Για την μεταφορά χωρίζουμε περιπτώσεις:

-Αν v είναι σταθερά :

li $tr,v

-Αν v είναι καθολική μεταβλητή-ανήκει στο κυρίως πρόγραμμα:

lw $tr,-offset($s0)

-Αν v είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος ίσο με το τρέχον, ή προσωρινή μεταβλητή

lw $tr,-offset($sp)

- Αν v είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος ίσο με το τρέχον

lw $t0,-offset($sp)

lw $tr,($t0)

-Αν v είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος μικρότερο από το τρέχον

gnlvcode()

lw $tr,($t0)

-Αν v είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος μικρότερο από το τρέχον

gnlvcode()

lw $t0,($t0)

lw $tr,($t0)

def storerv(r, v) : μεταφορά δεδομένων από τον καταχωρητή r στη μνήμη (μεταβλητή v)

Όπως και προηγουμένως, χωρίζουμε περιπτώσεις:

- αν v είναι καθολική μεταβλητή –δηλαδή ανήκει στο κυρίως πρόγραμμα

sw $tr,-offset($s0)

-αν v είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος ίσο με το τρέχον, ή προσωρινή μεταβλητή

sw $tr,-offset($sp)

- αν v είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος ίσο με το τρέχον

lw$t0,-offset($sp)

sw $tr,($t0)

-αν v είναι τοπική μεταβλητή, ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος μικρότερο από το τρέχον

gnlvcode(v)

sw $tr,($t0)

- αν v είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος μικρότερο από το τρέχον

gnlvcode(v)

lw $t0,($t0)

sw $tr,($t0)

def get_branch(relop) :επιστρέφει το relop κατά την μετάφραση της αντίστοιχης τετράδας σε τελικό κώδικα

def get_op(operator) :επιστρέφει τον τελεστή κατά την μετάφραση της αντίστοιχης τετράδας σε τελικό κώδικα

Παραγωγή από τετράδες

Αφού πλέον έχουν κατασκευαστεί όλες οι βοηθητικές συναρτήσεις, στην συνάρτηση produce_final_code γίνεται η μετάφραση των εντολών ενδιάμεσου κώδικα, σε εντολές τελικού κώδικα γλώσσας assembly. Έτσι κοιτώντας μια – μια τις τετράδες με την βοήθεια των προηγούμενων συναρτήσεων έχουμε τα εξής:

Τετράδα : jump, “_”, “_”, label : Εντολές αλμάτων

j label

Τετράδα: relop(?),x,y,z : Εντολές αλμάτων

loadvr(x,1)

loadvr(y,2)

branch(?),$t1,$t2,z όπου branch μπορεί : beq,bne,bgt,blt,bge,ble

Τετράδα: :=, x, “_”, z : Εκχώρηση

loadvr(x,1)

storerv(1,z)

Τετράδα op x,y,z : Αριθμητικές πράξεις

loadvr(x,1)

loadvr(y,2)

op $t1,$t1,$t2 op: add,sub,mul,div

storerv(1,z)

Τετράδα : out “_”, “_”, x : Είσοδος έξοδος

Li $v0,1

li $a0, x

syscall

Τετράδα : in “_”, “_”, x : Είσοδος έξοδος

li $v0,5

syscall

Τετράδα : retv“_”, “_”, x : Επιστροφή τιμής συνάρτησης

loadvr(x,1)

lw $t0,-8($sp)

sw $t1,($t0)

αποθηκεύεται ο x στη διεύθυνση που είναι αποθηκευμένη στην 3ηθέση του εγγραφήματος δραστηριοποίησης

εναλλακτικά μπορούμε να γράψουμε το αποτέλεσμα στον $v0, και μετά πρέπει να φροντίσουμε να το πάρουμε από εκεί

loadvr(x,1)

move $v0,$t1

Παράμετροι συνάρτησης :

πριν από την πρώτη παράμετρο, τοποθετούμε τον $fpνα δείχνει στην στοίβα της συνάρτησης που θα δημιουργηθεί

add $fp,$sp,framelength

Τετράδα par,x,CV, _

loadvr(x,0)

sw $t0, -(12+4i)($fp)

όπου i ο αύξων αριθμός της παραμέτρου

Τετράδα par,x,REF, _ :

αν η καλούσα συνάρτηση και η μεταβλητή x έχουν το ίδιο βάθος φωλιάσματος, η παράμετρος x είναι στην καλούσα συνάρτηση τοπική μεταβλητή ή παράμετρος που έχει περαστεί με τιμή

add $t0,$sp,-offset

sw $t0,-(12+4i)($fp)

αν η καλούσα συνάρτηση και η μεταβλητή x έχουν το ίδιο βάθος φωλιάσματος, η παράμετρος x είναι στην καλούσα συνάρτηση παράμετρος που έχει περαστεί με αναφορά

lw $t0,-offset($sp)

sw $t0,-(12+4i)($fp)

αν η καλούσα συνάρτηση και η μεταβλητή x έχουν διαφορετικό βάθος φωλιάσματος, η παράμετρος x είναι στην καλούσα συνάρτηση τοπική μεταβλητή ή παράμετρος που έχει περαστεί με τιμή

gnlvcode(x)

sw $t0,-(12+4i)($fp)

αν η καλούσα συνάρτηση και η μεταβλητή x έχουν διαφορετικό βάθος φωλιάσματος, η παράμετρος x είναι στην

καλούσα συνάρτηση παράμετρος που έχει περαστεί με αναφορά

gnlvcode(x)

lw $t0,($t0)

sw $t0,-(12+4i)($fp)

Τετράδα : par,x,RET, _

γεμίζουμε το 3οπεδίο του εγγραφήματοςδραστηριοποίησης της κληθείσας συνάρτησης με τη διεύθυνση της προσωρινής μεταβλητής στην οποία θα επιστραφεί η τιμή

add $t0,$sp,-offset

sw $t0,-8($fp)

Τετράδα call, _, _, f Κλήση συνάρτησης

αρχικά γεμίζουμε το 2οπεδίο του εγγραφήματος δραστηριοποίησης της κληθείσας συνάρτησης με την διεύθυνση του εγγραφήματος δραστηριοποίησης του γονέα της, ώστε η κληθείσα να γνωρίζει που να κοιτάξει αν χρειαστεί να προσπελάσει μία μεταβλητή την οποία έχει δικαίωμα να προσπελάσει, αλλά δεν της ανήκει

αν καλούσα και κληθείσα έχουν το ίδιο βάθος φωλιάσματος, τότε έχουν τον ίδιο γονέα

lw $t0,-4($sp)

sw $t0,-4($fp)

αν καλούσα και κληθείσα έχουν διαφορετικό βάθος φωλιάσματος, τότε η καλούσα είναι ο γονέας της κληθείσας

sw $sp,-4($fp)

στη συνέχεια μεταφέρουμε τον δείκτη στοίβας στην κληθείσα

add $sp,$sp,framelength

καλούμε τη συνάρτηση

jalf

και όταν επιστρέψουμε παίρνουμε πίσω τον δείκτη στοίβας στην καλούσα

add $sp,$sp,-framelength

μέσα στην κληθείσα :

στην αρχή κάθε συνάρτησης αποθηκεύουμε στην πρώτη θέση του εγγραφήματοςδραστηριοποίησης την διεύθυνση επιστροφής τηςτην οποία έχει τοποθετήσει στον $ran jal

sw $ra,($sp)

στην τέλος κάθε συνάρτησης κάνουμε το αντίστροφο, παίρνουμε από την πρώτη θέση του εγγραφήματος δραστηριοποίησης την διεύθυνση επιστροφής τηςσυνάρτησης και την βάζουμε πάλι στον $ra. Μέσω του $raεπιστρέφουμε στην καλούσα

lw $ra,($sp)

jr $ra

Αρχή προγράμματος και Κυρίως πρόγραμμα :

το κυρίως πρόγραμμα δεν είναι το πρώτο πράγμα που μεταφράζεται, οπότε στην αρχή του προγράμματος χρειάζεται ένα άλμα που να οδηγεί στην πρώτη ετικέτα του κυρίως προγράμματος

j Lmain

στη συνέχεια πρέπει να κατεβάσουμε τον $sp κατά framelengthτης main

add $sp,$sp,framelength

και να σημειώσουμε στον $s0 το εγγράφημα δραστηριοποίησης της main ώστε να έχουμε εύκολη πρόσβαση στις global μεταβλητές

move $s0,$sp

9.Εκτέλεση

Έχοντας υλοποιηθεί ορθά οι παραπάνω φάσεις, έχει ολοκληρωθεί η μετάφραση του αρχικού προγράμματος εισόδου σε Minimal++ σε ισοδύναμο assembly για τον επεξεργαστή MIPS.

Συνολικά, τα αρχεία που έχουν δημιουργηθεί κατά την εκτέλεση του μεταφραστή είναι:

<filename>_IntFile.int : Το αρχείο ενδιάμεσου κώδικα που περιέχει τις τετράδες

<filename>_CFile.c : Το αρχείο που περιέχει τον ισοδύναμο κώδικα σε C

<filename>_FinalCode.asm: Το αρχείο που περιέχει τον ισοδύναμο τελικό κώδικα

Όπου <filename> το όνομα του αρχείου εισόδου του μεταφραστή.

Για την εκτέλεση του μεταφραστή μεταφερόμαστε μέσω τερματικού στον φάκελο/directory που βρίσκεται ο μεταφραστής και το αρχείο εισόδου.
Το όνομα του μεταφραστή είναι mnmal.py

-Για εκτέλεση σε περιβάλλον Linux : python3 mnmal.py <filename>.min

-Για εκτέλεση σε περιβάλλον Windows: mnmal.py <filename>.min

Το αρχείο εισόδου πρέπει να έχει κατάληξη min και να δοθεί σαν όρισμα κατά την εκτέλεση. Μόνο ένα αρχείο εισόδου επιτρέπεται.

-Παράδειγμα:

```
program testi1 {
    declare x,y,z;
    {
        x := 10;
        y := 100;
        z := 1001;
        if (x>y and y<z) then {
            print(z)
        }else{
            print(x);
        }
    }
}
```

Παραπάνω παρουσιάζεται ένα παράδειγμα κώδικα στην γλώσσα Minimal++ με το όνομα testi1.min.
Για την μετάφραση του:

Περιβάλλον Linux :

Στο directory όπου βρίσκεται ο μεταφραστής και το αρχείο εισόδου δίνουμε στο τερματικό την εντολή:

```
python3 mnmal.py testi1.min
```

Περιβάλλον Windows :

Στο φάκελο όπου βρίσκεται ο μεταφραστής και το αρχείο εισόδου δίνουμε στο command line την εντολή :

```
mnmal.py testi1.min
```

Τα αποτελέσματα που προκύπτουν είναι τα εξής:

testi1_IntFile.int :

0: ['begin_block', 'testi1', '_', '_']

1: [':=', '10', '_', 'x']

2: [':=', '100', '_', 'y']

3: [':=', '1001', '_', 'z']

4: ['>', 'x', 'y', 6]

5: ['jump', '_', '_', 10]

6: ['<', 'y', 'z', 8]

7: ['jump', '_', '_', 10]

8: ['out', '_', '_', 'z']

9: ['jump', '_', '_', 11]

10: ['out', '_', '_', 'x']

11: ['halt', '_', '_', '_']

12: ['end_block', 'testi1', '_', '_']

testi1_CFile.c :

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
```

```
        int x;
        int y;
        int z;

        L_0: ; // ['begin_block', 'testi1', '_', '_']
        L_1: x = 10; // [':=', '10', '_', 'x']
        L_2: y = 100; // [':=', '100', '_', 'y']
        L_3: z = 1001; // [':=', '1001', '_', 'z']
        L_4: if (x > y) goto L_6; // ['>', 'x', 'y', 6]
        L_5: goto L_10; // ['jump', '_', '_', 10]
        L_6: if (y < z) goto L_8; // ['<', 'y', 'z', 8]
        L_7: goto L_10; // ['jump', '_', '_', 10]
        L_8: printf("%d\n", z); // ['out', '_', '_', 'z']
        L_9: goto L_11; // ['jump', '_', '_', 11]
        L_10: printf("%d\n", x); // ['out', '_', '_', 'x']
        L_11: ; // ['halt', '_', '_', '_']
        L_12: ; // ['end_block', 'testi1', '_', '_']
}
```

Testi1_FinalCode.asm:

```
j Lmain
L0:
sw $ra, ($sp)
L1:
li $t1, 10
sw $t1, -($s0)
L2:
li $t1, 100
sw $t1, -($s0)
L3:
li $t1, 1001
sw $t1, -($s0)
L4:
lw $t1, -($s0)
lw $t2, -($s0)
bgt $t1, $t2, L
L5:
```

```
j L10
L6:
lw $t1, -($s0)
lw $t2, -($s0)
blt $t1, $t2, L
L7:
j L10
L8:
li $v0, 1
li $t0, out
add $a0, $t1, 0
syscall
L9:
j L11
L10:
li $v0, 1
li $t0, out
add $a0, $t1, 0
syscall
L11:
L12:
lw $ra, ($sp)
jr $ra
```

10.Επίλογος

Η παραπάνω αναφορά παρουσιάζει τα βήματα με τα οποία έγινε η κατασκευή του μεταφραστή όπως και την υλοποίηση τους και κάποια παραδείγματα μέσα από τον κώδικα.

Επίσης εμπεριέχει χρήσιμες πληροφορίες και σχήματα από τις διαφάνειες που παρουσιάστηκαν στην διάρκεια του μαθήματος.

Αν και το μεγαλύτερο μέρος της υλοποίησης του πραγματοποιήθηκε σε περίεργους καιρούς, το πρότζεκτ αυτό παρουσιάζει και διδάσκει σε μεγάλο βαθμό την λειτουργία ενός μεταφραστή και την δομή μιας γλώσσας προγραμματισμο. Τυχόν λάθη θα υπάρξουν, αλλά η ανάπτυξη του και η γνώση που αποκομίστηκε αποτελεί σίγουρα ένα εργαλείο στη φαρέτρα κάθε προγραμματιστή