Νείρος Κωνσταντίνος 2503
Παπαγιάννη Ιωάννα 2790

Μέρος 1: Λεκτικός και συντακτικός αναλυτής

Το μέρος αυτό αναφέρεται στην υλοποίηση του λεκτικού και του συντακτικού αναλυτή του προγράμματος. Ο λεκτικός αναλυτής διαβάζει το αρχείο εισόδου και επιστρέφει κάθε λεκτική μονάδα. Ο συντακτικός αναλυτής ελέγχει εάν το πρόγραμμα ακολουθεί την γραμματική της γλώσσας.
Ο λεκτικός αναλυτής έγινε με την βοήθεια ενός πεπερασμένου αυτόματου. Η αρχική κατάσταση είναι η 0. Ο λεκτικός αναλυτής διαβάζει χαρακτήρα χαρακτήρα το αρχείο εισόδου και κάνει την αντίστοιχη μετάβαση. Για παράδειγμα από την κατάσταση 0 εάν διαβαστεί γράμμα μεταβαίνουμε στην κατάσταση 1. Οι καταστάσεις 0,1.. έως 9 είναι ενδιάμεσες και όλες οι υπόλοιπες είναι οι τελικές. Οι παρακάτω μεταβλητές και συναρτήσεις χρησιμοποιήθηκαν για τον λεκτικό αναλυτή:

transitionDiagram: Η λίστα αυτή περιέχει όλες τις πιθανές μεταβάσεις από μια ενδιάμεση κατάσταση με ένα συγκεκριμένο σύμβολο

word: Η λίστα αυτή περιέχει την λέξη που διαβάζουμε από το αρχείο. Στο τέλος του λεκτικού αναλυτή μετατρέπουμε την λίστα σε αλφαριθμητικό.

state: Η μεταβλητή state περιέχει την τρέχουσα κατάσταση που βρισκόμαστε. Στο τέλος του λεκτικού αναλυτή περιέχει την κατάσταση στην οποία αντιστοιχεί η λέξη.

finalStates: Η λίστα περιέχει όλες τις τελικές καταστάσεις που έχουν δημιουργηθεί και αναγνωρίζονται από το αυτόματο.

charMap: Το λεξικό περιέχει μια αντιστοίχηση χαρακτήρων με αριθμούς. Οι χαρακτήρες αντιπροσωπεύουν τους χαρακτήρες που διαβάζουμε από το αρχείο ενώ οι αριθμοί είναι η στήλη του πίνακα μεταβάσεων στον οποίο αντιστοιχίζουμε τον αριθμό.

letters: Περιέχει όλα τα κεφαλαία και μικρά γράμματα. Η λίστα χρησιμοποιείται για να ελέγξουμε εάν ένας χαρακτήρας είναι γράμμα ή όχι

digits: Περιέχει όλα τα ψηφία. Η λίστα χρησιμοποιείται για να ελέγξουμε εάν ένας χαρακτήρας είναι ψηφίο ή όχι

statesRet: Η λίστα περιέχει καταστάσεις όπως την idtk και την constanttk. Οι καταστάσεις αυτές διαβάζουν έναν χαρακτήρα τον οποίο δεν χρησιμοποιούν και ο οποίος ανήκει σε άλλη λεκτική μονάδα. Επομένως πρέπει να επιστραφεί ο χαρακτήρας στο αρχείο και να διαγραφεί από την τρέχουσα λεκτική μονάδα.

bindWords: Η λίστα περιέχει όλες τις δεσμευμένες λέξεις του προγράμματος, όπως την program και την if.

lines: Ο μετρητής lines μετράει της γραμμές του προγράμματος για να τυπώσουμε κατάλληλα μηνύματα λάθους.

lex(): Η συνάρτηση lex() υλοποιεί τον λεκτικό αναλυτή. Αρχικά βάζουμε την λέξη στο κενό και την κατάσταση στην 0. Στην συνέχεια όσο δεν είμαστε σε τελική κατάσταση διαβάζουμε τον επόμενο χαρακτήρα, αυξάνουμε τον μετρητή γραμμών εάν έχουμε διαβάζει αλλαγή γραμμής και προσθέτουμε στην λέξη τον χαρακτήρα. Ακολούθως βρίσκουμε τον αριθμό στήλης που ανήκει ο χαρακτήρας που διαβάσαμε βλέποντας εάν ανήκει στην λίστα των αριθμών, στην λίστα των ψηφίων και τέλος στο λεξικό. Αφού γίνει αυτό κάνουμε την μετάβαση με βάση τον πίνακα μεταβάσεων. Σε περίπτωση που η νέα κατάσταση είναι η 0 σβήνουμε τα περιεχόμενα της λέξης αφού δεν μας δίνουν κάποια πληροφορία. Όταν βρεθούμε σε τελική κατάσταση, αρχικά ελέγχουμε άν η κατάσταση ανήκει στις statesRet και εάν ναι, επιστρέφουμε τον χαρακτήρα στο αρχείο. Ακολούθως φτιάχνουμε την λέξη σε αλφαριθμητικό, αφού την είχαμε σε λίστα, και ελέγχουμε για περιπτώσεις λάθους. Τέλος ελέγχουμε εάν η λέξη ανήκει στις δεσμευμένες και εάν ναι τότε αλλάζουμε κατάλληλα τον κωδικό της λέξης.

Ο συντακτικός αναλυτής ελέγχει την σωστή γραφή του προγράμματος σε γλώσσα EEL. Αυτό σημαίνει πως ελέγχει εάν οι προτάσεις ακολουθούν τους κανόνες τις γραμματικής. Οι κανόνες τις γραμματικής υλοποιούνται με συναρτήσεις στις οποίες ελέγχουμε εάν οι λεκτικές μονάδες δίνονται με την αναμενόμενη μορφή. Η εύρεση των λεκτικών μονάδων γίνεται με την κλήση του λεκτικού αναλυτή. Η συντακτική ανάλυση ξεκινάει με την κλήση της συνάρτησης program().

program(): Αρχικά καλεί τον λεκτικό αναλυτή για να διαβάσει την πρώτη λεκτική μονάδα. Εάν δεν είναι η κατάσταση “programtk” το πρόγραμμα τερματίζει με μήνυμα λάθους the “first word must

be program” στο οποίο αναφέρεται και η γραμμή του λάθους. Στην συνέχεια καλεί λεκτικό αναλυτή για να βρει την επόμενη λεκτική μονάδα. Αυτή πρέπει να είναι το όνομα του προγράμματος επομένως πρέπει να έχει κατάσταση "idtk". Εάν δεν ισχύει αυτό τυπώνεται μήνυμα λάθους και το πρόγραμμα τερματίζει. Ο λεκτικός αναλυτής καλείται ξανά και στην συνέχεια καλείται η συνάρτηση block(). Μετά την επιστροφή από την block() αναμένουμε την λεκτική μονάδα με κατάσταση "endprogramtk".

block(): Η συνάρτηση block() με την σειρά καλεί τις συναρτήσεις:

declarations()
subprograms()
statements()

declarations(): Η declarations() ελέγχει εάν η λεκτική μονάδα έχει κωδικό “declaretk”. Στην περίπτωση αυτή καλείται ο λεκτικός αναλυτής και στην συνέχεια η συνάρτηση varlist(). Μετά την varlist() περιμένουμε την λεκτική μονάδα με κωδικό "enddeclaretk". Εάν δεν την βρούμε τότε τερματίζουμε τυπώνοντας κατάλληλο μήνυμα λάθους. Εδώ πρέπει να τονίσουμε ότι με βάση την γραμματική εάν δεν βρεθεί η λεκτική μονάδα έχει κωδικό “declaretk” στην αρχή της συνάρτησης δεν τυπώνουμε μήνυμα λάθους καθώς μπορούν να παραλειφθούν οι δηλώσεις μεταβλητών. Αυτό σημαίνει πως η πρώτη λεκτική μονάδα που βλέπει η declarations() ανήκει ενδεχομένως στην subprograms(). Επομένως και όταν υπάρχει το “declaretk” η συνάρτηση θα πρέπει να διαβάζει και την πρώτη λεκτική μονάδα, ενδεχομένως, του subprograms(). Επειδή η γραμματική έχει και άλλες τέτοιες περιπτώσεις κανόνων αποφασίσαμε πως κάθε συνάρτηση θα καλείται έχοντας διαβασμένη μια λεκτική μονάδα και πριν τερματίσει θα διαβάζει την επόμενη λεκτική μονάδα για να μην χρειάζεται να σκεφτόμαστε εάν πρέπει να κληθεί λεκτικός αναλυτής ή όχι.

Μέρος 2: Μετατροπή σε ενδιάμεσο κώδικα

Μετά την ολοκλήρωση του λεκτικού και του συντακτικού αναλυτή ακολουθεί η γραφή του ενδιάμεσου κώδικα. Σε αυτό το μέρος πρέπει να γίνεται μετατροπή των εκφράσεων της γλώσσας EEL σε κάταλληλες εκφράσεις του ενδιάμεσου κώδικα. Ο ενδιάμεσος κώδικας αποτελείται από τετράδες της μορφής [quadLabel, op, x, y, z] όπου το quadLabel είναι ο αριθμός της τετράδας, το op ο τελεστής ενώ τα x, y και z εξαρτώνται από την μορφή του op. Εαν για παράδειγμα το op είναι αριθμητικός τελεστής τότε τα x, y είναι μεταβλητές ή σταθερές ενώ το z είναι μεταβλητή. Η τετράδα τότε αναπαριστά την πράξη z = x op y. Η παραγωγή του ενδιάμεσου κώδικα γίνεται παράλληλα με την συντακτικη τοποθετώντας σε κατάλληλα σημεία της γραμματικής κλήσεις των συναρτήσεων παραγωγής του ενδιάμεσου κώδικα. Οι συναρτήσεις αυτές είναι οι παρακάτω:

1. nextquad(): Επιστρέφει τον αριθμό της επόμενης τετράδας που πρόκειται να παραχθεί
2. newtemp(): Δημιουργεί και επιστρέφει μια νέα προσωρινή μεταβλητή. Οι προσωρινές μεταβλητές είναι της μορφής T_1, T_2, κ.ο.κ.
3. genquad(): Παράγει την τετράδα op, x, y, z
4. emptylist(): Δημιουργεί μια κενή λίστα ετικετών τετράδων
5. makelist(int label): Δημιουργεί μια λίστα ετικετών τετράδων που περιέχει μόνο το label.
6. merge(list1,list2): Δημιουργεί μια λίστα ετικετών τετράδων από την συνένωση των λιστών list1 και list2.
7. backpatch(list, z): Η λίστα list αποτελείται απο δείκτες σε τετράδες των οποίων το τελευταίο τελούμενο δεν ειναι συμπληρωμένο. Η backpatch επισκέπτεται μία μία τις τετράδες αυτές και τις συμπληρώνει με την ετικέτα z.

Οι παρακάτω μεταβλητές χρησιμοποιήθηκαν για την παραγωγή του ενδιάμεσου κώδικα:

quadList: Κρατάει όλες τις τετράδες που παράγονται για τον ενδιάμεσο κώδικα
temp: Δείχνει τον αριθμό της επόμενης προσωρινής μεταβλητής που θα παραχθεί
quadLabel: Δείχνει τον αριθμό της επόμενης τετράδας που θα παραχθεί
name: Κρατάει το όνομα του block
programName: Κρατάει το όνομα του προγράμματος
place: Κρατάει το αποτέλεσμα ενός κανόνα όταν αυτό είναι μια μεταβλήτη ή μια σταθερά
assignmentVariable: κρατάει το όνομα μιας μεταβλητής για την οποία έχει κληθεί η assignmentStat(). Αυτό χρειάζεται γιατί μέσα στην συνάρτηση το χρειαζόμαστα αλλά καλούμε τον λεκτικό ανάλυτη πριν καλέσουμε την συνάρτηση.

exitList: Κρατάει τα jump που προκύπτουν από κλήσεις exit
cList: Κρατάει τις μεταβλητές που δηλώνονται στο πρόγραμμα. Χρειάζεται για την μετατροπή του αρχείου EEL σε αρχείο C.

Στην συνέχεια για κάθε συνάρτηση δίνουμε το σχέδιο ενδιάμεσου κώδικα για την παραγωγή της σε τελικό

block():

declarations()
subprograms()
genquad("begin block", name, "_","_")
statements()
if blockName == programName
genquad("halt", "_","_","_")
genquad("end block", name, "_","_")

expression():

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

{P1}: w=newTemp()
genquad("+",T1.place, T2.place, w)
T1.place = w

{P2}: E.place = T1.place

term():

F1(x F2(){P1}){P2}

{P1}: w=newTemp()
genquad("+",F1.place, F2.place, w)
F1.place = w

{P2}: T.place = F1.place

condition():

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

assignmentStat():

S-> id:=E {P1}

{P1}: genquad(":=",E.place,"_",id)

ifstat():

IF-> if condition then{P1} statements {P2}
else {P3} statements endif{P4}

{P1}: backpatch(condition.true, nextquad())

{P2}: jump = makeList(nextquad())
genquad("jump","_","_","_")

{P3}: backpatch(condition.false, nextquad())

{P4}: backpatch(jump, nextquad())

whilestat():

while-> {P0} while condition {P1} statements {P2} endwhile {P3}

{P0}: start = nextquad()

{P1}: backpatch(condition.true, nextquad())

{P2}: genquad("jump","_","_",start)

{P3}: backpatch(condition.false, nextquad())

exitStat():

exit -> exit{P1}

{P1} list1 = makelist(nextquad())
genquad(“jump”,”_”,”_”,”_”)
exitList = merge(exitList, list1)

repeatStat():

repeat -> {P0} repeat statements {P1} endrepeat {P2}

{P0}: exitList = emptyList()
start = nextquad()
{P1}: genquad(“jump”,”_”,”_”,start)
{P2}: backpatch(exitList, nextquad())

switchstat():

switch -> {P4} switch exp1
(case exp2 {P0} : statements{P1} {P2})$^{+}$
endswitch {P3}

{P0}: cond = makeList(nextquad())
genquad(“<>”, exp1.place, exp2.place, “_”)

{P1}: list1 = makelist(nextquad())
genquad(“jump”,”_”,”_”,”_”)
jump = merge(jump, jump1)

{P4}: jump = emptyList()
{P2}: backpatch(cond, nextquad())
{P3}: backpatch(jump, nextquad())

forcasestat():

witch -> {P0} forcase exp1

(when cond{P0} : {P1}statements {P2})$^{+}$

endforcase {P3}

{P0}: start = nextQuad()

counter = newTemp()

genquad(“:=”, 0, “_”, counter)

{P1}: backpatch(cond.true, nextquad())

genquad(“+”, 1, counter, counter)

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

list1 = makelist(nextquad())

genquad(“jump”,”_”,”_”,”_”)

jump = merge(jump, jump1)

{P4}: jump = emptyList()

{P3}: genquad(“>=”, counter, 1, start)

Οι παρακάτω συναρτήσεις χρησιμοποιήθηκαν για την αποθήκευση των αποτελεσμάτων της παραγωγής ενδιάμεσου κώδικα:

printInt(): Αποθηκεύει τις τετράδες που παράγει ο ενδιάμεσος κώδικας σε αρχείο

printC(): Μετατρέπει τις τετράδες που παράγει ο ενδιάμεσος κώδικας σε αρχείο

Μέρος 3: Μετατροπή σε τελικό κώδικα

Μετά την ολοκλήρωση του ενδιάμεσου κώδικα ακολουθεί η παραγωγή τελικού κώδικα. Σε αυτό το μέρος πρέπει να γίνεται μετατροπή του ενδιάμεσου κώδικα σε εντολές assembly και να ελεγχθεί ο κώδικας για λάθη που αφορούν την χρήση και δήλωση μεταβλητών και συναρτήσεων. Αρχικά παράγεται ο πίνακας συμβόλων που κρατάει πληροφορίες για τις συναρτήσεις και τις μεταβλητές του προγράμματος και στην συνέχεια με την βοήθεια αυτού γίνονται οι απαραίτητοι έλεγχοι και η παραγωγή του τελικού κώδικα. Οι έλεγχοι που έχουν γίνει για την ορθότητα του προγράμματος είναι:

1. κάθε συνάρτηση έχει μέσα της τουλάχιστον ένα return
2. δεν υπάρχει return έξω από συνάρτηση (σε διαδικασία ή το κυρίως πρόγραμμα)
3. υπάρχει exit μόνο μέσα σε βρόχους repeat
4. κάθε μεταβλητή, συνάρτηση ή διαδικασία που έχει δηλωθεί να μην έχει δηλωθεί πάνω από μία φορά στο βάθος φωλιάσματος στο οποίο βρίσκεται
5. κάθε μεταβλητή, συνάρτηση ή διαδικασία που χρησιμοποιείται έχει δηλωθεί και μάλιστα με τον τρόπο που χρησιμοποιείται

Οι παρακάτω μεταβλητές χρησιμοποιήθηκαν για την παραγωγή του πίνακα συμβόλων και του τελικου κώδικα:

scopeList: κρατάει την λίστα με τα scope, δηλαδή τον πίνακα συμβόλων
scopeLabel: κρατάει το βάθος φωλιάσματος του κάθε scope
start: κρατάει την πρώτη τετράδα που πρέπει να μετατραπεί σε τελικό
end: κρατάει την τελευταία τετράδα που πρέπει να μετατραπεί σε τελικό
insertScope(scopeName,type): εισάγει στον πίνακα συμβόλων μία συνάρτηση ή διαδικάσια με βάση το όρισμα type

deleteScope(): διαγράφει ένα scope από τον πίνακα συμβόλων

printScopeList(): τυπώνει τον πίνακα συμβόλων

findEntity(entityName): ψάχνει στον πίνακα συμβόλων στην πρώτη γραμμή εάν υπάρχει entity με το όνομα που δίνεται σαν όρισμα ώστε να μην δηλώσουμε πάνω απο μια φορά στο ίδιο scope το ίδιο entity.

insertEntity(entity): Εισάγει ένα entity στην λίστα με τα entity

findOffset(): υπολογίζει το offset στο οποίο θα μπεί μια νέα μεταβλητή

locateEntity(entityName): βρίσκει και επιστρέφει μια entity στον πίνακα συμβόλων. Η τιμή επιστροφής χρησιμοποιείται για να δούμε εάν μια μεταβλητή έχει χρησιμοποιηθεί σωστά.

findEntityAssembly(entityName): βρίσκει και επιστρέφει ένα entity μαζί με χρήσιμες πληροφορίες για τον τελικό κώδικα.

gnlvcode(offset, entityLabel): χρησιμοποιείται από τον τελικό κώδικα για να φορτώσει την διεύθυνση μιας μη τοπικής μεταβλητής στον καταχωρητή $t0

def loadvr(v,r): Βάζει στον καταχωρητή $tr την τιμή της μεταβλητής v γράφοντας τις κατάλληλες εντολές μετά τους ελέγχους για τον τύπο της μεταβλητής v

def storerv(r,v): Βάζει στην μεταβλητή v την τιμή του καταχωρητή $tr γράφοντας τις κατάλληλες εντολές μετά τους ελέγχους για τον τύπο της μεταβλητής v

areSiblings(f1, f2): Ελέγχει εάν δυο συναρτήσεις είναι αδέρφια δηλαδή εάν είναι δηλωμένες στο ίδιο βάθος φωλιάσματος για να γραφούν οι κατάλληλες εντολές για την μετατροπή της call

convertIntToAssembly(): μετατρέπει ενδιάμεσο κώδικα σε κώδικα Assembly. Η μετατροπή γίνεται όταν τελειώνει το block ενός υποπρογράμματος και πριν την διαγραφή του scope του από τον πίνακα συμβόλων.

Στην συνέχεια δίνεται ένα πρόγραμμα σε γλώσσα ELL, το αντίστοιχο του σε ενδιάμεσο κώδικα και σε κώδικα assembly καθώς και ο πίνακας συμβόλων όπως αυτός παράγεται πριν την διαγραφή ενός scope

Πρόγραμμα EEL

```
program prog
declare a,b,x,y,p enddeclare

procedure proc( in k, inout y)
    declare k1,l,g,aaaaaa,x enddeclare
    function f()
        declare a,b enddeclare
        return a
    endfunction
    function func(in k)
        declare o,a,x,n,t,c enddeclare
        while x<4
            aaaaaa:=0;
            a :=1;
            k:=4+n;
            x:=k
        endwhile;
        a:= 2+4/g-k*7;
        repeat endrepeat;
         if a+3*f(in t) = k  and [x<0] or not [ l>=0] and true  or false then
endif;

        if a>b then b:=90 else c:=100 endif;
        forcase
            when a > b : a:= 2+4/g-k*7;
            when a >= b : b:=3+4
        endforcase;

        switch
            k
            case l : o:=9
            case k : o:=10
        endswitch;
        return x
    endfunction
    input x;
    print x+2;
    repeat
        if x > 2 then exit; p:=0 else x:=3 endif;
        while x<2
            x:=x+1;
            repeat
                exit
            endrepeat;
            k:=0
        endwhile
    endrepeat

endprocedure
 function func(in k)
        declare o,a,x,n,t,c enddeclare
        return x
endfunction
call proc(in x, inout y);
input x;
a:=0
endprogram /*asdfasdfdsf sdf*/
```

Ενδιάμεσος κώδικας

```
[0, 'beginblock', 'f', '_', '_']
[1, 'retv', 'a', '_', '_']
[2, 'endblock', 'f', '_', '_']
[3, 'beginblock', 'func', '_', '_']
[4, '<', 'x', '4', 6]
[5, 'jump', '_', '_', 12]
[6, ':=', '0', '_', 'aaaaaa']
[7, ':=', '1', '_', 'a']
[8, '+', '4', 'n', 'T_1']
[9, ':=', 'T_1', '_', 'k']
[10, ':=', 'k', '_', 'x']
[11, 'jump', '_', '_', 4]
[12, '/', '4', 'g', 'T_2']
[13, '+', '2', 'T_2', 'T_3']
[14, '*', 'k', '7', 'T_4']
[15, '-', 'T_3', 'T_4', 'T_5']
[16, ':=', 'T_5', '_', 'a']
[17, 'jump', '_', '_', 17]
[18, 'par', 't', 'CV', '_']
[19, 'par', 'T_6', 'RET', '_']
[20, 'call', 'f', '_', '_']
[21, '*', '3', 'T_6', 'T_7']
[22, '+', 'a', 'T_7', 'T_8']
[23, '=', 'T_8', 'k', 25]
[24, 'jump', '_', '_', 27]
[25, '<', 'x', '0', 29]
[26, 'jump', '_', '_', 27]
[27, '>=', 'l', '0', 29]
[28, 'jump', '_', '_', 29]
[29, 'jump', '_', '_', 30]
[30, '>', 'a', 'b', 32]
[31, 'jump', '_', '_', 34]
[32, ':=', '90', '_', 'b']
[33, 'jump', '_', '_', 35]
[34, ':=', '100', '_', 'c']
[35, ':=', 0, '_', 'T_9']
[36, '>', 'a', 'b', 38]
[37, 'jump', '_', '_', 44]
[38, '+', 'T_9', 1, 'T_9']
[39, '/', '4', 'g', 'T_10']
[40, '+', '2', 'T_10', 'T_11']
[41, '*', 'k', '7', 'T_12']
[42, '-', 'T_11', 'T_12', 'T_13']
[43, ':=', 'T_13', '_', 'a']
[44, '>=', 'a', 'b', 46]
[45, 'jump', '_', '_', 49]
[46, '+', 'T_9', 1, 'T_9']
[47, '+', '3', '4', 'T_14']
[48, ':=', 'T_14', '_', 'b']
[49, '>=', 'T_9', 1, 35]
[50, '<>', 'k', 'l', 53]
[51, ':=', '9', '_', 'o']
[52, 'jump', '_', '_', 56]
[53, '<>', 'k', 'k', 56]
[54, ':=', '10', '_', 'o']
[55, 'jump', '_', '_', 56]
[56, 'retv', 'x', '_', '_']
[57, 'endblock', 'func', '_', '_']
```

```
[58, 'beginblock', 'proc', '_', '_']
[59, 'inp', 'x', '_', '_']
[60, '+', 'x', '2', 'T_15']
[61, 'out', 'T_15', '_', '_']
[62, '>', 'x', '2', 64]
[63, 'jump', '_', '_', 67]
[64, 'jump', '_', '_', 77]
[65, ':=', '0', '_', 'p']
[66, 'jump', '_', '_', 68]
[67, ':=', '3', '_', 'x']
[68, '<', 'x', '2', 70]
[69, 'jump', '_', '_', 76]
[70, '+', 'x', '1', 'T_16']
[71, ':=', 'T_16', '_', 'x']
[72, 'jump', '_', '_', 74]
[73, 'jump', '_', '_', 72]
[74, ':=', '0', '_', 'k']
[75, 'jump', '_', '_', 68]
[76, 'jump', '_', '_', 62]
[77, 'endblock', 'proc', '_', '_']
[78, 'beginblock', 'func', '_', '_']
[79, 'retv', 'x', '_', '_']
[80, 'endblock', 'func', '_', '_']
[81, 'beginblock', 'prog', '_', '_']
[82, 'par', 'x', 'CV', '_']
[83, 'par', 'y', 'REF', '_']
[84, 'call', 'proc', '_', '_']
[85, 'inp', 'x', '_', '_']
[86, ':=', '0', '_', 'a']
[87, 'halt', '_', '_', '_']
[88, 'endblock', 'prog', '_', '_']
```

Πίνακας Συμβόλων

[2, 'f', [['a', 12], ['b', 16]]]
[1, 'proc', [['k', 12, 'CV'], ['y', 16, 'REF'], ['k1', 20], ['l', 24], ['g', 28], ['aaaaaa', 32], ['x', 36], ['f', 'function', 0, 20]]]
[0, 'prog', [['a', 12], ['b', 16], ['x', 20], ['y', 24], ['p', 28], ['proc', 'procedure', -1, -1]]]
['func', 'function', -1, -1]

[2, 'func', [['k', 12, 'CV'], ['o', 16], ['a', 20], ['x', 24], ['n', 28], ['t', 32], ['c', 36], ['T_1', 40], ['T_2', 44], ['T_3', 48], ['T_4', 52], ['T_5', 56], ['T_6', 60], ['T_7', 64], ['T_8', 68], ['T_9', 72], ['T_10', 76], ['T_11', 80], ['T_12', 84], ['T_13', 88], ['T_14', 92]]]
[1, 'proc', [['k', 12, 'CV'], ['y', 16, 'REF'], ['k1', 20], ['l', 24], ['g', 28], ['aaaaaa', 32], ['x', 36], ['f', 'function', 0, 20], ['func', 'function', 3, 96]]]
[0, 'prog', [['a', 12], ['b', 16], ['x', 20], ['y', 24], ['p', 28], ['proc', 'procedure', -1, -1]]]

[1, 'proc', [['k', 12, 'CV'], ['y', 16, 'REF'], ['k1', 20], ['l', 24], ['g', 28], ['aaaaaa', 32], ['x', 36], ['f', 'function', 0, 20], ['func', 'function', 3, 96], ['T_15', 40], ['T_16', 44]]]
[0, 'prog', [['a', 12], ['b', 16], ['x', 20], ['y', 24], ['p', 28], ['proc', 'procedure', 58, 48]]]
['func', 'function', -1, -1]

[1, 'func', [['k', 12, 'CV'], ['o', 16], ['a', 20], ['x', 24], ['n', 28], ['t', 32], ['c', 36]]]
[0, 'prog', [['a', 12], ['b', 16], ['x', 20], ['y', 24], ['p', 28], ['proc', 'procedure', 58, 48], ['func', 'function', 78, 40]]]

[0, 'prog', [['a', 12], ['b', 16], ['x', 20], ['y', 24], ['p', 28], ['proc', 'procedure', 58, 48], ['func', 'function', 78, 40]]]

Τελικός κώδικας

```
j L_main
L_0:
sw $ra, ($sp)
L_1:
lw $t1,-12($sp)
lw $t0, -8($sp)
sw $t1, ($t0)
L_2:
lw $ra, ($sp)
jr $ra
L_3:
sw $ra, ($sp)
L_4:
lw $t1,-24($sp)
li $t2, 4
blt $t1, $t2, L_6
L_5:
j L_12
L_6:
li $t1, 0
lw $t0, -4($sp)
lw $t0, -4($t0)
add $t0, $t0, -32
lw $t1,($t0)
L_7:
li $t1, 1
sw $t1,-20($sp)
L_8:
li $t1, 4
lw $t2,-28($sp)
add $t1, $t1, $t2
sw $t1,-40($sp)
L_9:
lw $t1,-40($sp)
sw $t1,-12($sp)
L_10:
lw $t1,-12($sp)
sw $t1,-24($sp)
L_11:
j L_4
L_12:
li $t1, 4
lw $t0, -4($sp)
lw $t0, -4($t0)
add $t0, $t0, -28
lw $t2,($t0)
div $t1, $t1, $t2
sw $t1,-44($sp)
L_13:
li $t1, 2
lw $t2,-44($sp)
add $t1, $t1, $t2
```

```
sw $t1,-48($sp)
L_14:
lw $t1,-12($sp)
li $t2, 7
mul $t1, $t1, $t2
sw $t1,-52($sp)
L_15:
lw $t1,-48($sp)
lw $t2,-52($sp)
sub $t1, $t1, $t2
sw $t1,-56($sp)
L_16:
lw $t1,-56($sp)
sw $t1,-20($sp)
L_17:
j L_17
L_18:
add $fp, $sp, 20
lw $t0,-32($sp)
sw $t0, --12($fp)
L_19:
add $t0, $sp, -60
sw $t0, -8($fp)
L_20:
sw $t0, -4($sp)
sw $t0, -4($fp)
sw $sp, -4($fp)
add $sp, $sp, 20
jal L_0
add $sp, $sp, -20
L_21:
li $t1, 3
lw $t2,-60($sp)
mul $t1, $t1, $t2
sw $t1,-64($sp)
L_22:
lw $t1,-20($sp)
lw $t2,-64($sp)
add $t1, $t1, $t2
sw $t1,-68($sp)
L_23:
L_24:
j L_27
L_25:
lw $t1,-24($sp)
li $t2, 0
blt $t1, $t2, L_29
L_26:
j L_27
L_27:
lw $t0, -4($sp)
lw $t0, -4($t0)
add $t0, $t0, -24
lw $t1,($t0)
li $t2, 0
bge $t1, $t2, L_29
L_28:
j L_29
L_29:
j L_30
```

```
L_30:
lw $t1,-20($sp)
lw $t2,-16($s0)
bgt $t1, $t2, L_32
L_31:
j L_34
L_32:
li $t1, 90
sw $t1,-16($s0)
L_33:
j L_35
L_34:
li $t1, 100
sw $t1,-36($sp)
L_35:
li $t1, 0
sw $t1,-72($sp)
L_36:
lw $t1,-20($sp)
lw $t2,-16($s0)
bgt $t1, $t2, L_38
L_37:
j L_44
L_38:
lw $t1,-72($sp)
li $t2, 1
add $t1, $t1, $t2
sw $t1,-72($sp)
L_39:
li $t1, 4
lw $t0, -4($sp)
lw $t0, -4($t0)
add $t0, $t0, -28
lw $t2,($t0)
div $t1, $t1, $t2
sw $t1,-76($sp)
L_40:
li $t1, 2
lw $t2,-76($sp)
add $t1, $t1, $t2
sw $t1,-80($sp)
L_41:
lw $t1,-12($sp)
li $t2, 7
mul $t1, $t1, $t2
sw $t1,-84($sp)
L_42:
lw $t1,-80($sp)
lw $t2,-84($sp)
sub $t1, $t1, $t2
sw $t1,-88($sp)
L_43:
lw $t1,-88($sp)
sw $t1,-20($sp)
L_44:
lw $t1,-20($sp)
lw $t2,-16($s0)
bge $t1, $t2, L_46
L_45:
j L_49
```

```
L_46:
lw $t1,-72($sp)
li $t2, 1
add $t1, $t1, $t2
sw $t1,-72($sp)
L_47:
li $t1, 3
li $t2, 4
add $t1, $t1, $t2
sw $t1,-92($sp)
L_48:
lw $t1,-92($sp)
sw $t1,-16($s0)
L_49:
lw $t1,-72($sp)
li $t2, 1
bge $t1, $t2, L_35
L_50:
L_51:
li $t1, 9
sw $t1,-16($sp)
L_52:
j L_56
L_53:
L_54:
li $t1, 10
sw $t1,-16($sp)
L_55:
j L_56
L_56:
lw $t1,-24($sp)
lw $t0, -8($sp)
sw $t1, ($t0)
L_57:
lw $ra, ($sp)
jr $ra
L_58:
sw $ra, ($sp)
L_59:
li $v0, 5
syscall
move $t0, $v0
sw $t0,-36($sp)
L_60:
lw $t1,-36($sp)
li $t2, 2
add $t1, $t1, $t2
sw $t1,-40($sp)
L_61:
li $v0, 1
lw $t0,-40($sp)
move $a0, $t0
syscall
L_62:
lw $t1,-36($sp)
li $t2, 2
bgt $t1, $t2, L_64
L_63:
j L_67
L_64:
```

```
j L_77
L_65:
li $t1, 0
sw $t1,-28($s0)
L_66:
j L_68
L_67:
li $t1, 3
sw $t1,-36($sp)
L_68:
lw $t1,-36($sp)
li $t2, 2
blt $t1, $t2, L_70
L_69:
j L_76
L_70:
lw $t1,-36($sp)
li $t2, 1
add $t1, $t1, $t2
sw $t1,-44($sp)
L_71:
lw $t1,-44($sp)
sw $t1,-36($sp)
L_72:
j L_74
L_73:
j L_72
L_74:
li $t1, 0
sw $t1,-12($sp)
L_75:
j L_68
L_76:
j L_62
L_77:
lw $ra, ($sp)
jr $ra
L_78:
sw $ra, ($sp)
L_79:
lw $t1,-24($sp)
lw $t0, -8($sp)
sw $t1, ($t0)
L_80:
lw $ra, ($sp)
jr $ra
L_81:
L_main:
add $sp, $sp, 32
move $s0, $sp
L_82:
add $fp, $sp, 48
lw $t0,-20($s0)
sw $t0, --12($fp)
L_83:
add $t0, $sp, -24
sw $t0, --16($fp)
L_84:
sw $sp, -4($fp)
add $sp, $sp, 48
```

```
jal L_58
add $sp, $sp, -48
L_85:
li $v0, 5
syscall
move $t0, $v0
sw $t0,-20($s0)
L_86:
li $t1, 0
sw $t1,-12($s0)
L_87:
L_88:
```