

# Μεταφραστές

Γλώσσα Cimple

Βασίλειος Βογιάννου 3193 cs03193

Δήμητρα Μαχαιρίδου 4108 cs04108

## Λεκτική ανάλυση

Αρχικά για να ξεκινήσει ο μεταγλωττιστής μας χρειάζεται τις λεκτικές μονάδες του προγράμματος. Αυτές μπορεί να είναι λέξεις, δεσμευμένες ή απλά αλφαριθμητικά, αριθμοί και σύμβολα διαφόρων τύπων. Αυτό υλοποιείται με την συνάρτηση `lektikosAnalitis()`.

Πιο συγκεκριμένα ξεκινάμε πάντα από την κατάσταση 0. Αν το input είναι:

- γράμμα τότε πηγαίνουμε στην κατάσταση 1
- αριθμός τότε πηγαίνουμε στην κατάσταση 2
- '<' τότε πηγαίνουμε στην κατάσταση 3 όπου και ελέγχουμε αν ακολουθεί = ή >
- '>' τότε πηγαίνουμε στην κατάσταση 4 όπου και ελέγχουμε αν ακολουθεί =
- ':' τότε πηγαίνουμε στην κατάσταση 5 όπου και ελέγχουμε αν ακολουθεί =
- '.' τότε πηγαίνουμε στην κατάσταση 7 όπου και ελέγχουμε αν ακολουθεί κάτι. Αυτό συμβαίνει επειδή με την '.' έχουμε end of program και δεν θέλουμε να ακολουθεί κάτι.
- '#' τότε πηγαίνουμε στην κατάσταση 6 όπου και ελέγχουμε αν ακολουθεί κάτι., εδώ γίνεται ο έλεγχος για τα σχόλια. Οτιδήποτε και να ακολουθεί μας διατηρεί στην κατάσταση και μόλις λάβουμε πάλι '#' τότε το σχόλιο κλείνει και επιστρέφουμε στην κατάσταση 0.
- '+', '-', '\*', '/', ';', ',', '[', ']', '(', ')', '{', '}' οδηγούμαστε στις αντίστοιχες καταστάσεις.

Αυτή η διαδικασία πραγματοποιείται μέχρι να βρεθούμε σε τελική κατάσταση. Ταυτόχρονα διατηρούμε και τον τύπο της μονάδας.

Στην συνέχεια ελέγχουμε το μήκος των αλφαριθμητικών το οποίο θέλουμε να είναι μικρότερο των 30 χαρακτήρων αλλά και την ύπαρξη δεσμευμένων λέξεων.

Επιπλέον γίνεται έλεγχος για την οριοθέτηση των αριθμών.

## Συντακτική ανάλυση

Το επόμενο κομμάτι στην υλοποίηση του προγράμματος διαπραγματεύεται την σύνταξη του πηγαίου κώδικα. Αυτό σημαίνει ότι γίνεται έλεγχος σχετικά με την δομή των κανόνων, δηλαδή κοιτάμε τις λεκτικές μονάδες που ακολουθούν και ελέγχουμε αν βρίσκονται σε σωστή θέση. Σε αυτό το σημείο δημιουργούμε τον κατάλληλο κανόνα για κάθε τέτοια δομή.

Πιο συγκεκριμένα:

- `program : program ID block`

Αφού ελέγχουμε ότι βρισκόμαστε όντως στην `program` τότε ελέγχουμε αν ακολουθεί αναγνωριστικό ώστε να καλέσουμε την `block`.

- `block : declarations subprograms statements`

Σε αυτή την περίπτωση απλά καλούμε τις `declarations()`, `subprograms()`, `statements()`.

- `declarations : ( declare varlist ; )*`

Εδώ χρειάζεται να ελέγχουμε για ύπαρξη διαδοχικών-πολλαπλών `declare` ώστε μόλις συμβεί αυτό να καλέσουμε την `varlist()` για να γίνει σωστά η δήλωση. Η διαδικασία αυτή συνεχίζεται όσο βρίσκουμε `';` δηλαδή ολοκλήρωση της δήλωσης.

- `varlist : ID ( , ID )* | ε`

Αφού ελέγχουμε για το πρώτο αναγνωριστικό τότε αν αυτό υπάρχει και όσο βρίσκουμε `'` αναζητούμε για το επόμενο.

- `subprograms : ( subprogram )*`

Εδώ ελέγχουμε για συναρτήσεις ή διαδικασίες και όσο υπάρχουν καλούμε την `subprogram()`.

- `subprogram : function ID ( formalparlist ) block | procedure ID ( formalparlist ) block`

Εφόσον βρεθήκαμε εδώ από συνάρτηση ή διαδικασία πρέπει να βρούμε και το αναγνωριστικό τους για το οποίο και ελέγχουμε. Στην συνέχεια κοιτάμε για ορίσματα καλώντας την `formalparlist()` (αυτό πρέπει να υπάρχει εντός παρενθέσεων τις οποίες και αναζητούμε) και έπειτα την `block()`.

- `formalparlist : formalparitem ( , formalparitem )* | ε`

Εδώ ελέγχουμε και τους τύπους των ορισμάτων οι οποίοι μπορεί να είναι `in`, `inout`. Σε αυτή τη περίπτωση καλούμε την `formalparitem()` όσο βρίσκουμε `'` για να πάρουμε όλα τα πιθανά ορίσματα.

- `formalparitem : in ID | inout ID`

Εδώ ελέγχουμε σε ποια από τις δύο παραπάνω περιπτώσεις βρισκόμαστε και παίρνουμε το αντίστοιχο αναγνωριστικό.

- `statements : statement ; | { statement ( ; statement )* }`

Εδώ κάνουμε μια συλλογή από `statement` τα οποία χωρίζονται μεταξύ τους με `';` και εντάσσονται σε `'{ , '}'`. Αλλιώς έχουμε ένα μοναδικό `statement`.

- statement : assignStat | ifStat | whileStat | switchcaseStat | forcaseStat | incaseStat | callStat | returnStat | inputStat | printStat | ε

Σε αυτή την περίπτωση ελέγχουμε σε ποιο statement βρισκόμαστε και καλούμε την αντίστοιχη συνάρτηση υλοποίησης του.

- assignStat : ID := expression

Εδώ κάνουμε εκχώρηση έχοντας ως όρισμα το αναγνωριστικό και ελέγχοντας αν όντως υπάρχει το σύμβολο της εκχώρησης ':='.

- ifStat : if ( condition ) statements elsepart

Σε αυτή την περίπτωση θέλουμε να έχουμε μια συνθήκη (condition) μέσα σε παρενθέσεις. Όταν αυτό ισχύσει πρέπει να καλέσουμε τις statements(), elsepart() ώστε να εκτελεστεί το κατάλληλο κομμάτι κώδικα.

- elsepart : else statements | ε

Εδώ ελέγχουμε αν όντως υπάρχει else ώστε να καλέσουμε την statements ().

- whileStat : while ( condition ) statements

Σε αυτή την περίπτωση θέλουμε να έχουμε μια συνθήκη (condition) μέσα σε παρενθέσεις. Όταν αυτό ισχύσει πρέπει να καλέσουμε την statements() ώστε να εκτελεστεί το κατάλληλο κομμάτι κώδικα.

- switchcaseStat: switchcase

( case ( condition ) statements )\*

default statements

Σε αυτή την περίπτωση θέλουμε να έχουμε μια συνθήκη (condition) μέσα σε παρενθέσεις για κάθε case. Όσο αυτό ισχύει πρέπει να καλέσουμε την statements() ώστε να εκτελεστεί το κατάλληλο κομμάτι κώδικα. Αν υπάρξει default πρέπει ομοίως να καλέσουμε τον αντίστοιχο κώδικα υλοποίησης.

- forcaseStat : forcase

( case ( condition ) statements )\*

default statements

Σε αυτή την περίπτωση θέλουμε να έχουμε μια συνθήκη (condition) μέσα σε παρενθέσεις για κάθε case. Όσο αυτό ισχύει πρέπει να καλέσουμε την statements() ώστε να εκτελεστεί το κατάλληλο κομμάτι κώδικα. Αν υπάρξει default πρέπει ομοίως να καλέσουμε τον αντίστοιχο κώδικα υλοποίησης.

- incaseStat : incase

( case ( condition ) statements )\*

Σε αυτή την περίπτωση θέλουμε να έχουμε μια συνθήκη (condition) μέσα σε παρενθέσεις για κάθε case. Όσο αυτό ισχύει πρέπει να καλέσουμε την statements() ώστε να εκτελεστεί το κατάλληλο κομμάτι κώδικα.

- returnStat : return( expression )

Εδώ αναζητούμε ένα expression μέσα σε παρενθέσεις ώστε να ξέρουμε τι θα επιστρέψουμε.

- callStat : call ID( actualparlist )

Εδώ γίνεται η κλήση οπότε χρειαζόμαστε το αναγνωριστικό για το οποίο ελέγχουμε και ορίσματα (actualparlist) μέσα σε παρενθέσεις ώστε να επιτύχει.

- printStat : print( expression )

Εδώ αναζητούμε ένα expression μέσα σε παρενθέσεις ώστε να ξέρουμε τι θα εκτυπώσουμε.

- inputStat : input( ID )

Εδώ αναζητούμε ένα αναγνωριστικό μέσα σε παρενθέσεις ώστε να ξέρουμε τι θα πάρουμε ως είσοδο.

- actualparlist : actualparitem ( , actualparitem )\* | ε

Εδώ έχουμε μια λίστα από παραμέτρους (actualparitem) χωρισμένους με ','. Για να ενταχθούν στην λίστα πρέπει να έχουν τύπο.

- actualparitem : in expression | inout ID

Εδώ ελέγχουμε τον τύπο των παραμέτρων και καλούμε τον αντίστοιχο κανόνα (in) ή αναζητούμε το αναγνωριστικό του (inout).

- condition : boolterm ( or boolterm )\*

Εδώ χειριζόμαστε τις boolean εκφράσεις. Όσο δίνεται 'or' καλούμε την boolterm () ώστε να ολοκληρωθεί η έκφραση.

- boolfactor : not [ condition ] | [ condition ] | expression REL\_OP expression

Εδώ χειριζόμαστε τους παράγοντες της παραπάνω έκφρασης οι οποίοι μπορεί να είναι άρνηση μιας συνθήκης ή μια συνθήκη ή μια σχέση. Ανάλογα με το αποτέλεσμα του ελέγχου μας οδηγούμαστε στην κατάλληλη επιλογή.

- expression : optionalSign term ( ADD\_OP term )\*

Εδώ χειριζόμαστε τις αριθμητικές εκφράσεις..Όσο δίνεται '+','-' καλούμε τις term ( ) ώστε να ολοκληρωθεί η έκφραση.

- factor : INTEGER | ( expression ) | ID idtail

Εδώ χειριζόμαστε τους παράγοντες της παραπάνω έκφρασης.Ανάλογα με το αποτέλεσμα του ελέγχου μας οδηγούμαστε στην κατάλληλη επιλογή.

- optionalSign

Εδώ ελέγχουμε αν έχουμε '+','-'.

- REL\_OP

Εδώ ελέγχουμε αν έχουμε '=',<='','>','<','>','<','>'.

### **Παραγωγή ενδιάμεσου κώδικα**

Ο μεταγλωττιστής μας μεταφράζει αρχικά το πηγαίο πρόγραμμα σε ένα ισοδύναμο γραμμένο σε μια ενδιάμεση γλώσσα χαμηλότερου επιπέδου.Το κομμάτι αυτό ονομάζεται ενδιάμεσος κώδικας και υλοποιείται κυρίως μέσα στον συντακτικό αναλυτή.

Η βασική δομή του κώδικα είναι μια πεντάδα (label+τετράδα) η οποία αποτελείται από μια ετικέτα (label) που την αριθμεί,ένα σύμβολο (x) που χαρακτηρίζει την επιθυμητή ενέργεια,δύο μεταβλητές ή σταθερές (y,z) στις οποίες αναφέρεται η ενέργεια και μια μεταβλητή (w) στην οποία αποθηκεύεται το αποτέλεσμα.Η μορφή της είναι label,x,y,z,w.Σημειώνουμε ότι εμφανίζονται τα begin\_block και end\_block τα οποία λειτουργούν ως flags έναρξης και τερματισμού αντίστοιχα,το par που χαρακτηρίζει μια παράμετρο και τα CV,REF,RET τα οποία είναι τύποι παραμέτρων.

Για την υλοποίηση του κώδικα χρησιμοποιήσαμε κάποιες βοηθητικές συναρτήσεις:

- nextquad():επιστρέφει την ετικέτα της επόμενης τετράδας που πρόκειται να παραχθεί.
- genquad(op,x,y,z):παράγει την επόμενη τετράδα.
- newtemp():παράγει την επόμενη προσωρινή μεταβλητή και την επιστρέφει.
- emptylist():παράγει μια κενή λίστα
- makelist(x):επιστρέφει μια λίστα η οποία εμπεριέχει μόνο την ετικέτα x.
- merge(l1,l2):επιστρέφει την λίστα l1 αφού πρώτα την συνενώσει με την l2.
- backpatch(list,z):αλλάζει το τελευταίο στοιχείο των τετράδων που βρίσκονται στην λίστα list και τοποθετεί το z.

Αναφορικά με τους κανόνες:

- `block()`:μόλις βρούμε το σημείο έναρξης και τερματισμού του προγράμματος ή της συνάρτησης ή της διαδικασίας χρησιμοποιούμε την `genquad()` ώστε να παράξουμε την ανάλογη τετράδα.Επίσης αν βρισκόμαστε στη `main` τότε πρέπει να προσθέσουμε τετράδα για το `halt` τερματισμού πριν από αυτόν.
- `assignStat()`:μόλις βρούμε τα ονόματα των μεταβλητών-σταθερών που χρειάζονται για την ανάθεση τότε τα περνάμε σε τετράδα με την `genquad()`.
- `ifStat()`:αν η συνθήκη ισχύει τότε στην λίστα `true` του `condition` παράγουμε την επόμενη τετράδα με την `backpatch()`.Εφόσον πραγματοποιηθούν τα `statements` παράγουμε μια λίστα ώστε να οδηγηθούμε στο τέλος της `if`.Αυτό γίνεται με τις συναρτήσεις `makelist()`,`genquad()`.Αν όμως η συνθήκη δεν ισχύει τότε στην λίστα `false` του `condition` παράγουμε την επόμενη τετράδα με την `backpatch()`.
- `whileStat()`:αν η συνθήκη ισχύει τότε στην λίστα `true` του `condition` παράγουμε την επόμενη τετράδα με την `backpatch()`.Εφόσον πραγματοποιηθούν τα `statements` οδηγούμαστε στην αρχή της `while`.Αυτό γίνεται με την συνάρτηση `genquad()`.Αν όμως η συνθήκη δεν ισχύει τότε στην λίστα `false` του `condition` παράγουμε την επόμενη τετράδα με την `backpatch()`.
- `switchcaseStat()`:μόλις μπορούμε στην `switchcase` παράγουμε μια κενή λίστα ώστε να τα χρησιμοποιήσουμε αργότερα.Αυτό γίνεται με την συνάρτηση `emptylist()`.Αν η συνθήκη ισχύει τότε στην λίστα `true` του `condition` παράγουμε την επόμενη τετράδα με την `backpatch()`.Εφόσον πραγματοποιηθούν τα `statements` μαζεύουμε πληροφορίες σχετικά με τα σημεία τα οποία πηγαίνει το `default`.Αυτά γίνονται με τις συναρτήσεις `makelist()`, `genquad()`,`merge()`.Αν όμως η συνθήκη δεν ισχύει τότε στην λίστα `false` του `condition` παράγουμε την επόμενη τετράδα με την `backpatch()`.
- `forcaseStat()`:μόλις μπορούμε στην `forcase` παράγουμε την επόμενη τετράδα και μια κενή λίστα ώστε να τα χρησιμοποιήσουμε αργότερα.Αυτά γίνονται με τις συναρτήσεις `nextquad()`,`emptylist()`.Αν η συνθήκη ισχύει τότε στην λίστα `true` του `condition` παράγουμε την επόμενη τετράδα με την `backpatch()`.Εφόσον πραγματοποιηθούν τα `statements` μαζεύουμε πληροφορίες σχετικά με τα σημεία τα οποία πηγαίνει το `default`.Αυτά γίνονται με τις συναρτήσεις `makelist()`, `genquad()`,`merge()`.Αν όμως η συνθήκη δεν ισχύει τότε στην λίστα `false` του `condition` παράγουμε την επόμενη τετράδα με την `backpatch()`.Στην περίπτωση που βρεθούμε στην `default` πρέπει να μεταβούμε στην επόμενη τετράδα οπότε κάνουμε `jump` με την `genquad()`.
- `returnStat()`:μόλις βρούμε το όνομα της μεταβλητής-σταθεράς που χρειάζεται για την επιστροφή τότε το περνάμε σε τετράδα με την `genquad()`.
- `callStat()`:μόλις βρούμε το όνομα της συνάρτησης που χρειάζεται για την κλήση τότε το περνάμε σε τετράδα με την `genquad()`.Το όνομα της προσωρινής μεταβλητής που χρησιμοποιεί το παράγουμε με την `newtemp()`.
- `printStat()`:μόλις βρούμε το όνομα της μεταβλητής-σταθεράς που χρειάζεται για την εκτύπωση τότε το περνάμε σε τετράδα με την `genquad()`.

- `inputStat()`:μόλις βρούμε το όνομα της μεταβλητής-σταθεράς που χρειάζεται για την εισαγωγή τότε το περνάμε σε τετράδα με την `genquad()`.
- `actualparitem()`:μόλις βρούμε το όνομα της μεταβλητής-σταθεράς που χρειάζεται για το πέρασμα παραμέτρου και αφού ελέγχουμε τον τύπο της τότε το περνάμε σε τετράδα με την `genquad()`.
- `boolfactor()`:στην τελευταία περίπτωση αυτής της δομής παράγουμε λίστα για τα `true` με την `makelist()`,παράγουμε την επόμενη τετράδα με την `genquad()`,παράγουμε λίστα για τα `false` με την `makelist()` και παράγουμε την επόμενη τετράδα με την `genquad()` ώστε να γίνεται `jump`.
- `expression()`:μόλις βρούμε το σύμβολο και τα ονόματα των μεταβλητών-σταθερών που χρειάζονται τότε τα περνάμε σε τετράδα με την `genquad()`.Το όνομα της προσωρινής μεταβλητής που χρησιμοποιεί το παράγουμε με την `newtemp()`.
- `term()`:μόλις βρούμε το σύμβολο και τα ονόματα των μεταβλητών-σταθερών που χρειάζονται τότε τα περνάμε σε τετράδα με την `genquad()`.Το όνομα της προσωρινής μεταβλητής που χρησιμοποιεί το παράγουμε με την `newtemp()`.
- `idtail()`:μόλις βρούμε το όνομα της συνάρτησης που χρειάζεται για την κλήση τότε το περνάμε σε τετράδα με την `genquad()`.Το όνομα της προσωρινής μεταβλητής που χρησιμοποιεί το παράγουμε με την `newtemp()`.

Η εκτύπωση του ενδιάμεσου κώδικα και του προγράμματος σε C γίνεται με τις συναρτήσεις `get_intermediate_program()` και `get_program_C()` αντίστοιχα.Οι εκτυπώσεις γίνονται σε αρχεία όταν αυτό είναι επιτρεπτό,δηλαδή παράγουμε αρχείο σε C όταν μέσα στον πηγαίο κώδικα δεν υπάρχει συνάρτηση ή διαδικασία.

### Πίνακας συμβόλων

Αυτή η δομή καταγράφει τα ονόματα των συναρτήσεων και των μεταβλητών διατηρώντας και κάποιες βοηθητικές πληροφορίες για αυτά ώστε να γνωρίζουμε ποιά δομή μπορεί να διαχειριστεί κάποια άλλη.

Για τον λόγο αυτό χρησιμοποιούμε το `scope` το οποίο δημιουργείται κάθε φορά που συναντάμε τις λέξεις `program,function,procedure`.Επιπλέον κρατάμε για αυτό:

- τον βαθμό φωλιάσματος ο οποίος είναι η αρίθμηση του
- μια λίστα με `entities` τα οποία θα αναλύσουμε παρακάτω
- το 12 το οποίο είναι το αρχικό `offset` από το οποίο ξεκινάνε να μετράνε οι μεταβλητές
- το όνομα του

Η διαχείριση αυτής της δομής γίνεται με τις συναρτήσεις `record_scope()`,`delete_scope()` οι οποίες εισάγουν και εξάγουν αντίστοιχα.Σημειώνουμε ότι κάθε φορά που χειριζόμαστε μια συνάρτηση δημιουργείται ένα νέο εγγράφημα δραστηριοποίησης.

Επίσης χρησιμοποιούμε την δομή entity η οποία δημιουργείται κάθε φορά που συναντάμε συνάρτηση, διαδικασία, μεταβλητή. Για αυτήν κρατάμε το όνομα της και μια λίστα από argument. Σε περίπτωση που αυτά δεν είναι συναρτήσεις ή διαδικασίες μόνο τότε αυξάνουμε το offset των μεταβλητών κατά 4. Η διαχείριση αυτής της δομής γίνεται με τις συναρτήσεις record\_entity().

Για να υπολογίσουμε το framelength μιας συνάρτησης ολοκληρώνουμε πρώτα το πάνω scope και έπειτα αθροίζουμε το μέγιστο offset του με το 4. Αυτό συμβαίνει γιατί υπολογίζουμε αποστάσεις από την αρχή του τρέχοντος εγγραφήματος.

Επιπλέον χρησιμοποιούμε και την συνάρτηση search\_entity() για εύρεση ενός entity.

Για την ολοκλήρωση αυτής της υλοποίησης έχουμε προσθέσει κώδικα στις εξής συναρτήσεις:

- program(): μόλις βρούμε το όνομα του προγράμματος τότε το περνάμε ως scope με την record\_scope(). Αφού υλοποιηθεί η block διαγράφουμε το scope με την delete\_scope().
- block(): σε περίπτωση που έχω συνάρτηση ή διαδικασία τότε πρέπει να περάσω πληροφορία προς τα πάνω. Αυτό σημαίνει ότι βάζουμε με την nextquad() το startquad στο entity της συναρτησης. Επιπλέον προσθέτουμε το framelength.
- varlist(), formalpitem(): μόλις βρούμε το όνομα και τον τύπο της μεταβλητής αναζητούμε το offset ώστε να δημιουργήσουμε το entity με την record\_entity().
- subprogram(): μόλις βρούμε το όνομα της συνάρτησης ή της διαδικασίας τότε το περνάμε ως scope με την record\_scope() και ως entity με την record\_entity(). Αφού υλοποιηθεί η block διαγράφουμε το scope με την delete\_scope(). Σε αυτό το σημείο δεν γνωρίζουμε το startquad και το framelength της οπότε συμπληρώνουμε με κενό.

Η εκτύπωση του πίνακα συμβόλων εμφανίζεται στο αρχείο txt.

### Σημασιολογική ανάλυση

Στόχος είναι στο πρόγραμμα να ισχύουν τα εξής:

- κάθε συνάρτηση έχει τουλάχιστον ένα return
- δεν υπάρχει return έξω από συνάρτηση
- μια διαδικασία δεν μπορεί να έχει return

Για την υλοποίηση αυτών δημιουργήσαμε στην block μια global μεταβλητή η οποία μας βοηθά να ελέγξουμε την ύπαρξη των return στον πηγαίο κώδικα σε συνεργασία με την αναζήτηση του επιθυμητού entity.

Επίσης δεν πρέπει να υπάρχουν κοινά ονόματα. Για αυτό στην record\_entity() πριν εισάγουμε το νέο entity κάνουμε έλεγχο αν προϋπάρχει.

Επιπλέον δεν μπορώ να χρησιμοποιήσω μια συνάρτηση ως μεταβλητή, δηλαδή δεν γίνεται  $\alpha=f()$  ή  $\alpha=\alpha+f()$ . Για αυτό στην assignStat() αναζητώ το επιθυμητό entity και αν είναι συνάρτηση ή διαδικασία τότε υπάρχει λάθος. Ανάλογα λειτουργούμε και στις idtail(), callStat().



Σημειώνουμε ότι δεν έχουμε υλοποιήσει το ερώτημα στ.

### Παραγωγή τελικού κώδικα

Με τις πληροφορίες που έχουμε από τον ενδιάμεσο κώδικα και τον πίνακα συμβόλων δημιουργούμε τον τελικό κώδικα.

Υλοποιήσαμε τις εξής συναρτήσεις:

- `gencode(entity)`: μεταφέρει στον καταχωρητή `t0` την διεύθυνση μιας μη τοπικής μεταβλητής.
- `loadvr(v,r)`: μεταφέρει δεδομένα στον καταχωρητή `r`.
- `storevr(r,v)`: μεταφέρει δεδομένα από τον καταχωρητή `r` στη μεταβλητή `v`.
- `get_MIPS_program()`: με την συνάρτηση αυτή γίνεται ο βασικός έλεγχος και παράγωγή του τελικού αρχείου. Οι αντιστοιχίες είναι οι εξής:
  - `jump, “_”, “_”, label`  
`b label`
  - `relop(?),x,y,z`  
`loadvr(x,$t1)`  
`loadvr(y, $t2)`  
`branch(?),$t1,$t2,z`  
`branch(?) : beq,bne,bgt,blt,bge,ble`
  - `:=, x, “_”, z`  
`loadvr(x, $t1)`  
`storevr($t1, z)`
  - `op x,y,z`  
`loadvr(x, $t1)`  
`loadvr(y, $t2)`  
`op $t1,$t1,$t2`  
`op: add,sub,mul,div storevr($t1,z)`

- out “\_”, “\_”, x

li \$v0,1

loadvr(x,\$a0)

syscall

- in “\_”, “\_”, x

li \$v0,5

syscall

storerv(\$v0,x)

- retv “\_”, “\_”, x

loadvr(x, \$t1)

lw \$t0,-8(\$sp)

sw \$t1,(\$t0)

- par,x,τύπος, \_

- ❖ Αν είναι η πρώτη παράμετρος τότε γράφουμε `addi $fp,$sp,frameLength`

- ❖ Αν ο τρόπος περάσματος-τύπος είναι CV δηλαδή με γίνεται με τιμή τότε

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

όπου i μετρητής παραμέτρων

- ❖ Αν ο τρόπος περάσματος-τύπος είναι REF δηλαδή γίνεται με αναφορά τότε ελέγχουμε το βάθος φωλιάσματος της μεταβλητής και της καλούσας συνάρτησης (1) και τον τρόπο περάσματος της παραμέτρου σε αυτήν (2).

Αν ισχύει το (1) και το (2) είναι τοπική ή με τιμή τότε `addi $t0,$sp,-offset sw $t0,-(12+4i)($fp)` αλλιώς `lw $t0,-offset($sp) sw $t0,-(12+4i)($fp)`.

Αν δεν ισχύει το (1) και το (2) είναι τοπική ή με τιμή τότε `gnlncode(x) sw $t0,-(12+4i)($fp)` αλλιώς `gnlncode(x) lw $t0,($t0) sw $t0,-(12+4i)($fp)`.

- ❖ Αν ο τρόπος περάσματος-τύπος είναι RET τότε `addi $t0,$sp,-offset sw $t0,-8($fp)`

- `call, _, f`

Ελέγχουμε το βάθος φωλιάσματος της καλούσας και της κληθείσας συνάρτησης και αν αυτά είναι ίσα τότε `lw $t0,-4($sp) sw $t0,-4($fp)` αλλιώς `sw $sp,-4($fp)`.

Σε κάθε περίπτωση `addi $sp,$sp,frameLength jal f addi $sp,$sp,-frameLength`.

- `begin_block`

Αν δεν είναι η `main` του κυρίως προγράμματος τότε `sw $ra,($sp)` αλλιώς `j Lmain addi $sp,$sp,frameLength move $s0,$sp`.

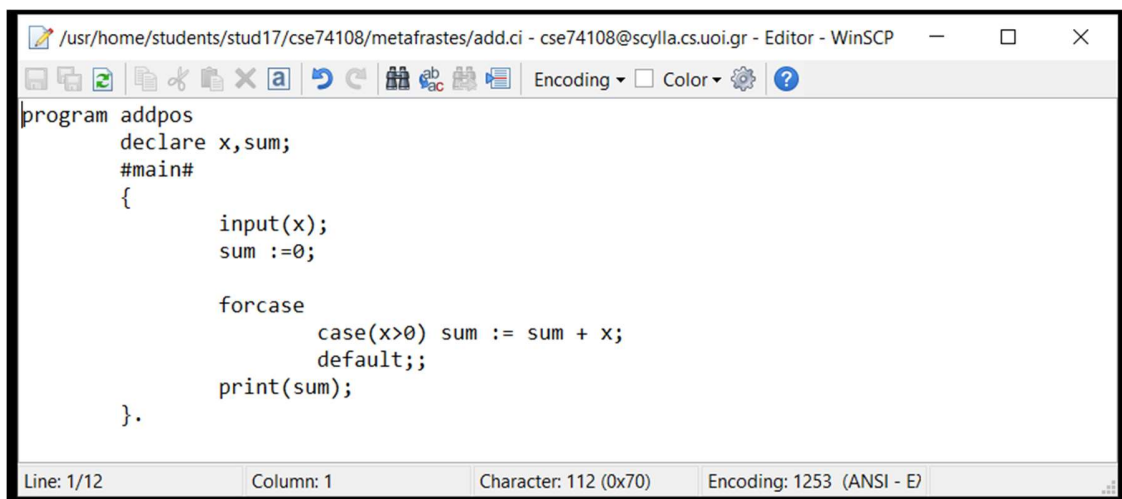
- `end_block`

Αν δεν είναι η `main` του κυρίως προγράμματος τότε `lw $ra,($sp) jr $ra`.

Σημειώνουμε ότι βασικό κριτήριο των παραπάνω συναρτήσεων είναι ο βαθμός φωλιάσματος.

### Παράδειγμα

Αρχικό πρόγραμμα



```

program addpos
declare x,sum;
#main#
{
    input(x);
    sum :=0;

    forcase
        case(x>0) sum := sum + x;
        default;;
    print(sum);
}.
  
```

Line: 1/12    Column: 1    Character: 112 (0x70)    Encoding: 1253 (ANSI - E)

## Ενδιάμεσος κώδικας

```
/usr/home/students/stud17/cse74108/metafrastes/add.int - cse74108@scylla.cs.uoi.gr - Editor - WinSCP
[[0, 'begin_block', 'addpos', '_', '_']
[1, 'inp', 'x', '_', '_']
[2, ':=', '0', '_', 'sum']
[3, '>', 'x', '0', 5]
[4, 'jump', '_', '_', 8]
[5, '+', 'sum', 'x', 'T_0']
[6, ':=', 'T_0', '_', 'sum']
[7, 'jump', '_', '_', 9]
[8, 'jump', '_', '_', '_']
[9, 'out', 'sum', '_', '_']
[10, 'halt', '_', '_', '_']
[11, 'end_block', 'addpos', '_', '_']

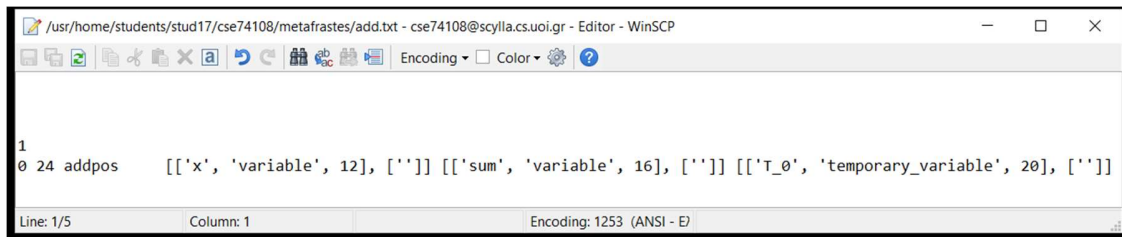
Line: 1/12      Column: 1      Character: 91 (0x5B)      Encoding: 1253 (ANSI - E)
```

Παράγεται η `begin_block` της `main`, έπειτα βρίσκει την `input` και σημειώνει την `x`, στην συνέχεια εκχωρεί το άθροισμα στο 0 και προχωρά στους ελέγχους. Ο σωστός έλεγχος `x>0` οδηγεί στην ετικέτα 5 ώστε να γίνει η άθροιση την οποία αρχικά περνάει σε μια προχωρινή μεταβλητή και έπειτα την εκχωρεί στο άθροισμα. Αν ο έλεγχος δεν είναι σωστός τότε υπάρχει η ακολουθία των `jump`. Στο τέλος υπάρχει το `halt` και το `end_block` της `main` ώστε να τερματίσει.

## Πρόγραμμα c

```
/usr/home/students/stud17/cse74108/metafrastes/add.c - cse74108@scylla.cs.uoi.gr - Editor - WinSCP
int main(){
int x
int sum
int T_0
//(inp,x,_,_)
L_1:scanf(x);
//(:=,0,_,sum)
L_2: sum=0;
//(>,x,0,5)
L_3: if(x>0) goto L5;
//(jump,_,_,8)
L_4: goto L_8;
//(+,sum,x,T_0)
L_5: T_0=sum+x;
//(:=,T_0,_,sum)
L_6: sum=T_0;
//(jump,_,_,9)
L_7: goto L_9;
//(jump,_,_,_)
L_8: goto L__;
//(out,sum,_,_)
L_9:printf(sum);
}
```

## Πίνακας συμβόλων

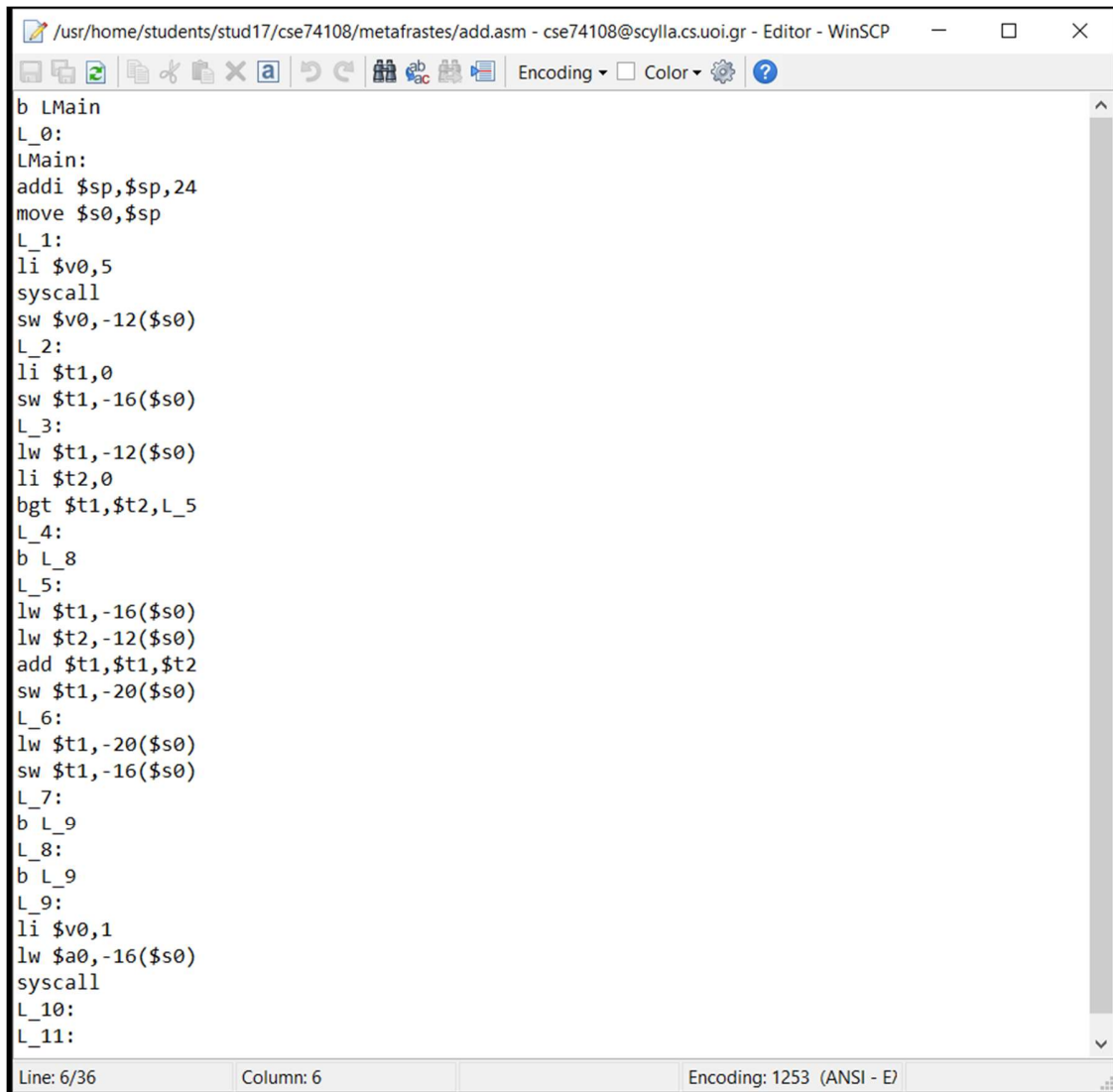


```
1
0 24 addpos      [['x', 'variable', 12], ['']] [['sum', 'variable', 16], ['']] [['T_0', 'temporary_variable', 20], ['']]
```

Line: 1/5      Column: 1      Encoding: 1253 (ANSI - E)

Το αποτέλεσμα είναι αναμενόμενο γιατί θέλουμε 1 scope με entities τα declare και την προσωρινή μεταβλητή που δημιουργήθηκε. Οι τιμές μεταβάλλονται ομαλά εφόσον αυξάνονται κατά 4 την φορά ξεκινώντας από 12.

## Τελικός κώδικας



```
b LMain
L_0:
LMain:
addi $sp,$sp,24
move $s0,$sp
L_1:
li $v0,5
syscall
sw $v0,-12($s0)
L_2:
li $t1,0
sw $t1,-16($s0)
L_3:
lw $t1,-12($s0)
li $t2,0
bgt $t1,$t2,L_5
L_4:
b L_8
L_5:
lw $t1,-16($s0)
lw $t2,-12($s0)
add $t1,$t1,$t2
sw $t1,-20($s0)
L_6:
lw $t1,-20($s0)
sw $t1,-16($s0)
L_7:
b L_9
L_8:
b L_9
L_9:
li $v0,1
lw $a0,-16($s0)
syscall
L_10:
L_11:
```

Line: 6/36      Column: 6      Encoding: 1253 (ANSI - E)

Αρχικά παράγεται κώδικας για το begin\_block της main στο LMain και για την input στο L\_1. Στην συνέχεια για την εκχώρηση του αθροίσματος στο L\_2 και για τον έλεγχο στο L\_3. Έπειτα στο L\_5 για την άθροιση και στο L\_6 για την εκχώρηση. Στο L\_9 υπάρχει κώδικας για την print. Οι τιμές που παίρνει από τον πίνακα συμβόλων έχουν ελεγχθεί σε προηγούμενο στάδιο.