Ομάδα : Μάριος Γαβριήλ Α.Μ : 4244

Μηνάς Ελευθερίου Α.Μ : 4245

Η εργασία αυτή έχει να κάνει με την μεταγλώττιση ενός προγράμματος γραμμένο στην Cimple γλώσσα , στην γλώσσα μηχανής για την αρχιτεκτονική τύπου MIPS. Για την παραγωγή του τελικού αυτού του κώδικα περνάμε το κείμενο του προγράμματος αυτού από τα στάδια της :

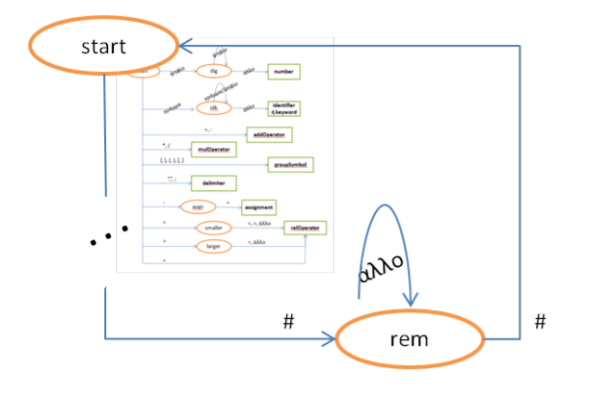
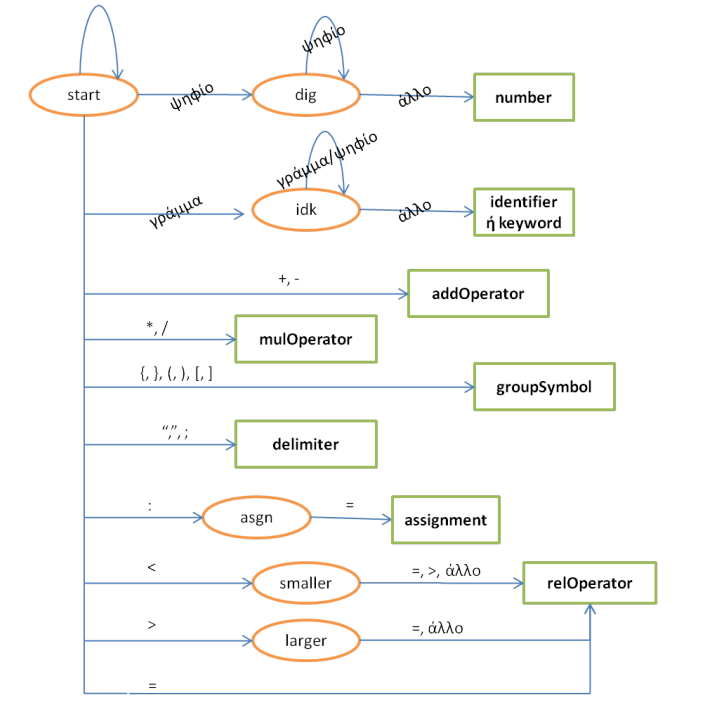
1. Λεκτικής Ανάλυσης
2. Συντακτικής ανάλυσης,
3. Παραγωγής ενδιάμεσου κώδικα
4. Παραγωγής πίνακα συμβόλων
5. Παραγωγής τελικού κώδικα στην αρχιτεκτονική τύπου MIPS.

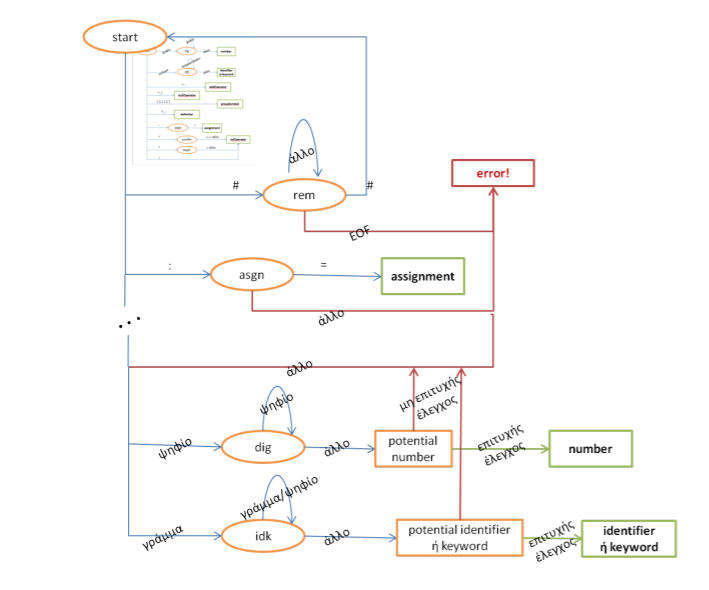
**Λεκτική Ανάλυση**

Στον λεκτικό αναλυτή παίρνουμε το αρχείο .ci αναλύουμε και ξεχωρίζουμε tokens από το αρχείο αυτό . Τα tokens αυτά χωρίζονται σε :

* Identifiers : Τα οποία μπορεί να είναι ονόματα μεταβλητών ,διαδικασιών ,συναρτήσεων και δεσμευμένες λέξεις.
* Numbers : Ακέραιοι αριθμοί
* Add Operators : Αριθμητικές πράξεις πρόσθεσης και αφαίρεσής
* Mul Operators : Αριθμητικές πράξεις πολλαπλασιασμού και διαίρεσής
* Rel Operators : Λογικές πράξεις ανισοτήτων

Διαχωρίζουμε τα tokens με βάση του αυτόματου λειτουργίας του λεκτικού αναλυτή διαβάζοντας το κείμενο ανα χαρακτήρα .





Για την υλοποίηση του αυτομάτου αυτού υλοποιήσαμε ένα πίνακα μεταβάσεων με γραμμές της καταστάσεις του αυτόματου:

0->Start 1->dig(ψηφία) 2->idk(αναγνωριστικά) 3->asgn(ανάθεση) 4->smaller(μικρότερο) 5->greater(μεγαλύτερο) 6->rem(σχόλια)

και στήλες όλες τις δυνατές εισόδου που μπορεί να πάρει το αυτόματο Όπου white\_characters είναι το διάστημα και το tab ,character όλοι οι χαρακτήρες του λατινικού αλφαβήτου κεφαλαία και μικρά γράμματα ,digits τα ψηφία 0-9 ,real\_op τα λογικά σύμβολα ,mult\_op πολλαπλασιασμός και διαίρεση ,add\_op πρόσθεση και αφαίρεση ,coma ,semi\_colon , full\_stop και colon, το κόμμα , το ερωτηματικό ,η τελεία,και η άνω και κάτω τελεία αντίστοιχα ,left/right\_parenthesis οι παρενθέσεις ,left/right\_brackets οι αγκύλες ’[‘ ’]’ ,open/close\_code\_block άνοιγμα και κλείσιμο κομματιών κώδικα ‘{‘ ‘}’, Eof το τέλος του αρχείου ,not\_acceptable\_character μη αποδεκτός χαρακτήρας, comment σχόλια.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Σχήμα1 Πίνακας Καταστάσεων.

Το tableofStates με βάση το αρχική κατάσταση και τον κάθε χαρακτήρα που διαβάζουμε από το κείμενο αποφασίζει την επόμενη κατάσταση ή token ή το μήνυμα λάθους που θα εμφανιστεί στην οθόνη. Το αυτόματο υλοποιείται στην συνάρτηση lex() το οποίο δημιουργεί και επιστρέφει μια λίστα που περιέχει το είδος του token , την κατάσταση στην οποία ανήκει αυτό , την λέξη του token αυτού και την γραμμή που βρίσκετε αυτό. Μέσα στην lex ελέγχουμε ανα χαρακτήρα το κείμενο σε ποια από της παραπάνω κατηγορίες ανήκει .Μέσα στον λεκτικό αναλυτή γίνεται ο έλεγχος για το μέγεθος της λέξης κάθε token έτσι ώστε να είναι μικρότερη από 30 χαρακτήρες και σε περίπτωση αριθμού έλεγχος για το αν αυτός ο αριθμός είναι μεταξύ του διαστήματος [-232-1 , 232-1].Έλεγχοι μέσα στο lex γίνονται κα για μη αποδεχτούς χαρακτήρες , αν μετά από ψηφίο υπάρχει χαρακτήρας ,αν υπάρχει τελεία πριν το τέλος του κειμένου ,αν δεν κλείνουν τα σχόλια ή βρεθεί τέλος αρχείου μέσα στα σχόλια και τέλος αν το token είναι αναγνωριστικό τότε γίνεται και έλεγχος για δεσμευμένες λέξεις.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Συντακτική Ανάλυση

Στην φάση της συντακτικής ανάλυσης παίρνουμε χρησιμοποιούμε τον λεκτικό αναλυτή και τα tokens που παράγει για να υλοποιήσουμε την γραμματική της γλώσσας Cimple που δίνει την ακριβή περιγραφή της γλώσσας. Η γραμματική της Cimple :

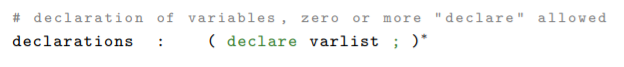
Εικόνα που περιέχει κείμενο

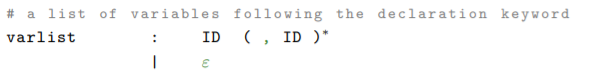
Περιγραφή που δημιουργήθηκε αυτόματα

Αφού καλεστεί η συνάρτηση Syntax\_analyzer() όπου και υλοποιείται η συντακτική ανάλυση ο πρώτος κανόνας της γραμματικής που καλείται είναι το program όπου και καλείται η lex() και παράγεται το πρώτο token που είναι η δεσμευμένη λέξη program ,ξανακαλείται η lex() και δημιουργείται ένα identifier\_token με λέξη του token αυτού, ένα αποδεκτό αναγνωριστικό που προσδιορίζει το όνομα του main program .Ακολουθεί ο κανόνας block και η χαρακτηριστική τελεία για το τέλος του κειμένου. Παράγονται μηνύματα λάθους αν το κείμενο δεν αρχίζει με την λέξη program , το αναγνωριστικό που ακολουθεί μετά το program δεν είναι αποδεκτό και εφόσον ολοκληρωθεί η συντακτική ανάλυση του κειμένου αν βρούμε Eof πριν από την χαρακτηριστική τελεία.



Ο κανόνας block υλοποιεί τους κανόνες declarations subprograms και statements.

 Μέσα στον κανόνα declarations ,ψάχνουμε μέσα σε ένα βρόγχο καλώντας την lex για το token της δεσμευμένης λέξης declare. Αν υπάρχει έστω και ένα καλείται ο κανόνας varlist ,ακολουθεί η lex που ψάχνει για ένα ερωτηματικό token. Το ερωτηματικό απαιτείται μετά το τέλος του κανόνα varlist είτε υπάρχει ένα ή περισσότερα από ένα declare. Παράγεται μήνυμα λάθους αν δεν υπάρχει ερωτηματικό μετά από κάθε varlist.



Το varlist υλοποιείται ως εξής , καλείται η lex γίνεται συνεχής έλεγχος για την εμφάνιση ενός αποδεκτού αναγνωριστικού μέσα σε ένα βρόγχο και αν υπάρχουν περισσότερα από ένα αναγνωριστικό , διαχωρίζονται με κόμματα .Ένα varlist μπορεί να είναι και κενό. Παράγεται μήνυμα λάθους στην περίπτωση που υπάρχουν περισσότερα από ένα αναγνωριστικά και δεν διαχωρίζονται από κόμμα .

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Αφού τελειώσει ο κανόνας declarations καλείται το subprograms όπου καλείται η lex και εφόσον υπάρχουν οι δεσμευμένες λέξεις function και procedure καλείται και υλοποιείται ο κανόνας subprogram . Μέσα στο block μπορεί να μην υπάρχει κάποια συνάρτηση ή διαδικασία.

Εικόνα που περιέχει κείμενο

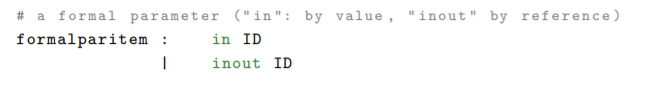
Περιγραφή που δημιουργήθηκε αυτόματα

Αναλόγως με το αν το είναι function ή procedure αν υπάρχει ένα αποδεκτό αναγνωριστικό για το συγκεκριμένο υπο πρόγραμμα μέσα σε παρενθέσεις καλείται ο κανόνας formalparlist και αφού τελειώσει η υλοποίηση του καλείται το block για το σώμα του υπο προγράμματος. Παράγονται μηνύματα λάθους για μη αποδεκτό αναγνωριστικό ,για αριστερή παρένθεση πριν από την υλοποίηση του formalparlist και για δεξιά παρένθεση μετά την υλοποίηση.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Εδώ καλείται ο κανόνας formalparItem τουλάχιστον μια φορά ή και περισσότερες από μία με διαχωριστικό ένα κόμμα.



Με βάση τις δεσμευμένες λέξεις που βρίσκουμε ,“in” για παράμετρο περασμένη με τιμή και “inout” για παράμετρο περασμένη με αναφορά ψάχνουμε για αποδεκτό αναγνωριστικό και αν δεν υπάρχει εμφανίζουμε μήνυμα λάθους .Παράγεται επίσης μήνυμα λάθους αν δεν υπάρχουν πριν το αναγνωριστικό οι δύο δεσμευμένες λέξεις.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Μετά την ολοκλήρωση του κανόνα subprograms καλείται το statements . Όπου και υλοποιείται το σώμα του συγκεκριμένου block. Στον κανόνα αυτό υπάρχει τουλάχιστον ένα statement ή και περισσότερα .Στην περίπτωση που υπάρχουν περισσότερα από ένα statement απαιτείται το άνοιγμα και κλείσιμο κομματιών κώδικα . Μετά από κάθε statement απαιτείται ερωτηματικό . Παράγονται μηνύματα λάθους αν δεν έχουν διατυπωθεί σωστά τα statements ή αν δεν υπάρχει ερωτηματικό.

Εικόνα που περιέχει πίνακας

Περιγραφή που δημιουργήθηκε αυτόματα

Κάθε statement που καλείται μπορεί να έχει μόνο ένα από τους παραπάνω κανόνες ή να είναι κενό.

Εικόνα που περιέχει κείμενο

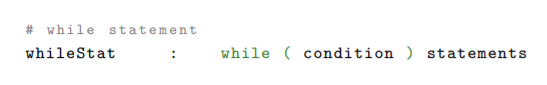
Περιγραφή που δημιουργήθηκε αυτόματα

Αν μετά την κλήση του statement βρεθεί αναγνωριστικό τότε υλοποιείται η assignStat, μετά το αναγνωριστικό αυτό απαιτείται το σήμα ανάθεσης (:=) και καλείται έπειτα ο κανόνας expression. Παράγονται μηνύματα λάθους στις περιπτώσεις οπού μετά το αναγνωριστικό δεν υπάρχει το σήμα ανάθεσης ή δεν υπάρχει αναγνωριστικό στη αρχή του κανόνα.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Αν βρεθεί η δεσμευμένη λέξη “if” καλείται ο κανόνας ifStat .Μετά το if απαιτείτε άνοιγμα και κλείσιμο παρενθέσεων ενδιάμεσα των οποίων καλείται τo condition .Έπειτα ξανακαλείται το statements για την υλοποίηση του σώματος της if και τέλος το elsepart. Παράγονται μηνύματα λάθους σε περίπτωση απουσίας των παρενθέσεων. Για το elsepart εφόσον βρίσκεται η δεσμευμένη λέξη else καλείται ξανά το statements υλοποίηση του σώματος του else. Ο κανόνας else μπορεί να είναι κενός.



Με την εύρεση της δεσμευμένης λέξης while καλείται η whilestat που έχει ενδιάμεσα παρενθέσεων το condition που αν είναι αληθές θα υλοποιεί τον βρόγχο και έπειτα καλείται το statements για το σώμα του βρόγχου. Μήνυμα λάθους παράγεται σε περίπτωση απουσίας παρενθέσεων.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Με την εύρεση της δεσμευμένης λέξης switch καλείται η switchcasestat όπου γίνεται έλεγχος για την δεσμευμένη λέξη case . Εφόσον υπάρχει ακολουθείται από την condition ενδιάμεσα παρενθέσεων και έπειτα καλείτε η statements. Αν στην switchcase δεν υπάρχουν cases τότε ακολουθεί η δεσμευμένη λέξη default όπου και καλείται η statements. Μηνύματα λάθους παράγονται στις περιπτώσεις απουσίας παρενθέσεων στα case ή απουσίας του default από το σώμα της switchcase.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Με την εύρεση της δεσμευμένης λέξης forcase καλείται η forcasestat όπου γίνεται έλεγχος για την δεσμευμένη λέξη case . Εφόσον υπάρχει ακολουθείται από την condition ενδιάμεσα παρενθέσεων και έπειτα καλείτε η statements. Αν στην forcase δεν υπάρχουν cases τότε ακολουθεί η δεσμευμένη λέξη default όπου και καλείται η statements. Μηνύματα λάθους παράγονται στις περιπτώσεις απουσίας παρενθέσεων στα case ή απουσίας του default από το σώμα της forcase.

Εικόνα που περιέχει κείμενο

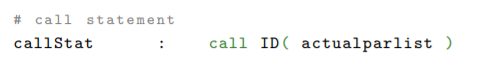
Περιγραφή που δημιουργήθηκε αυτόματα

Με την εύρεση της δεσμευμένης λέξης incase καλείται η incasestat όπου γίνεται έλεγχος για την δεσμευμένη λέξη case . Εφόσον υπάρχει ακολουθείται από την condition ενδιάμεσα παρενθέσεων και έπειτα καλείτε η statements . Μηνύματα λάθους παράγονται στις περιπτώσεις απουσίας παρενθέσεων στα case .

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Με την εύρεση της δεσμευμένης λέξης return καλείται η returnstat όπου και ενδιάμεσα παρενθέσεων απαιτείτε το expression που θα επιστραφεί. . Μηνύματα λάθους παράγονται στις περιπτώσεις απουσίας παρενθέσεων.

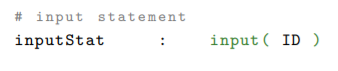


Με την εύρεση της δεσμευμένης λέξης call καλείται ο κανόνας callStat οπού και απαιτείται το αναγνωριστικό της διαδικασίας ή συνάρτησης που καλείται και μέσα σε παρενθέσεις καλείται ο κανόνας actualparlist για τη λίστα παραμέτρων που θα περαστούν στην καλούσα διαδικασία ή συνάρτηση . Μηνύματα λάθους παράγονται στις περιπτώσεις απουσίας παρενθέσεων ή σε περίπτωση μη αποδεκτού αναγνωριστικού για την καλούσα.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Με την εύρεση της δεσμευμένης λέξης print καλείται ο κανόνας printStat όπου και ενδιάμεσα παρενθέσεων καλείται το expression που απαιτείτε για τύπωση . Μηνύματα λάθους παράγονται στις περιπτώσεις απουσίας παρενθέσεων.



Με την εύρεση της δεσμευμένης λέξης input καλείται ο κανόνας inputStat όπου και ενδιάμεσα παρενθέσεων βρίσκεται ένα αναγνωριστικό που απαιτείτε για εισαγωγή. Μηνύματα λάθους παράγονται στις περιπτώσεις απουσίας παρενθέσεων ή και σε περίπτωση απουσίας του αναγνωριστικού αυτούΕικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Μετά την κλήση του callStat καλείτε η actualParList που δημιουργεί την λίστα των παραμέτρων για την καλούσα συνάρτηση ή διαδικασία καλώντας των κανόνα actualParItem. Η λίστα αυτή μπορεί να είναι κενή. Τα actualParItem διαχωρίζονται με κόμμα αν είναι περισσότερα από ένα .

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Η actualParItem διαβάζει τη δεσμευμένη λέξη “in” και καλεί την expression για το πέρασμα παραμέτρου με τιμή ή την δεσμευμένη λέξη “inout” και αναμένει ένα αποδεκτό αναγνωριστικό για την παράμετρο με αναφορά. Παράγονται μηνύματα λάθους για τις περιπτώσεις που δεν υπάρχουν οι δεσμευμένες λέξεις ή δεν υπάρχει αναγνωριστικό για το πέρασμα παραμέτρου με αναφορά.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Ο κανόνας condition καλεί τουλάχιστον μια φορά τον κανόνα boolterm για την διαμόρφωση μιας κατάστασης (αληθής ή ψευδής). Αν υπάρχουν περισσότερες από μια boolterm διαχωρίζονται από την δεσμευμένη λέξη or.

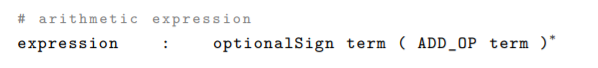
Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Ο κανόνας condition καλεί τουλάχιστον μια φορά τον κανόνα boolfactor . Αν υπάρχουν περισσότερες από μια boolfactor διαχωρίζονται από την δεσμευμένη λέξη andΕικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, πορτοκαλί, κλείσιμο

Περιγραφή που δημιουργήθηκε αυτόματα

Στον boolfactor ορίζουμε κατά πόσο θέλουμε η κατάσταση να είναι αληθής καλώντας την condition ανάμεσα σε αγκύλες ή ψευδής θέτοντας την δεσμευμένη λέξη not μπροστά από τις αγκύλες ή ορίζουμε μια λογική πράξη καλώντας την expression δύο φορές και βάζοντας ανάμεσα τους ένα λογικό σύμβολο . Παράγεται μήνυμα λάθους στην περίπτωση απουσίας των αγκυλών .



Ο κανόνας αποτελείται από ένα προαιρετικό πρόσημο που τοποθετείτε υλοποιώντας την optionalSign ακολουθείτε η κλήση του κανόνα term και αν υπάρχουν και άλλες κλήσεις του term χωρισμένες με αριθμητικούς συντελεστές(+ ή -).

Εικόνα που περιέχει κείμενο

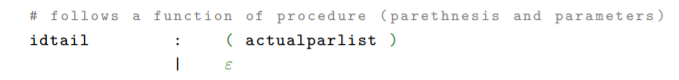
Περιγραφή που δημιουργήθηκε αυτόματα

Ο κανόνας term υλοποιεί μια αριθμητική έκφραση καλώντας τον factor μια ή περισσότερες φορές χωρισμένες από συντελεστές πολλαπλασίασης . (\* , /)

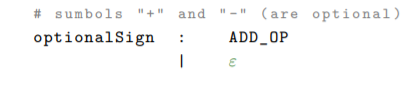
Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Ο κανόνας factor ψάχνει για έναν ακέραιο, ένα expression ή ένα αναγνωριστικό που ακολουθείται από την κλήση του idtail.



O κανόνας idtail αποτελείται από ένα actualparlist ή μπορεί να είναι και κενός .



Ο κανόνας είναι προαιρετικός και καλείται για να προσθέσει πρόσημο στον αριθμό ή έκφραση που ακολουθεί.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

**Παραγωγή Ενδιάμεσου Κώδικα**

Στην παραγωγή του ενδιάμεσου κώδικα παράγουμε τετράδες που περιέχουν ένα τελεστή και τρία τελούμενα για κάθε expression ,condition και statement . Οι τετράδες αυτές φέρουν ως πρώτο όρισμα μια ετικέτα με μοναδικό αριθμό*(π.χ 100 : +,a,b,c)* η κάθε μια και θα φανούν χρήσιμες στην παραγωγή τελικού κώδικα .Ο ενδιάμεσος κώδικας παράγεται ταυτόχρονα όσο υλοποιείτε και ο συντακτικός αναλυτής. Για να παραχθούν οι τετράδες αυτές υλοποιούμε τις βοηθητικές συναρτήσεις nextQuad , genQuad , newtemp , emptyList , makeList, merge και backpatch.

**nextQuad**() : Επιστρέφει τον αριθμό της επόμενης τετράδας που πρόκειται να παραχθέι

**genQuad (op,x,y,z)** : Δημιουργεί την τετράδα με τα ορίσματα op , x , y ,z

**newtemp()** : Δημιουργεί μια νέα προσωρινή μεταβλητή της μορφής Τ\_1,Τ\_2 .. και την επιστρέφει

**emptyList ()** : Δημιουργεί μια κενή λίστα ετικετών τετράδων.

**makeList (χ):** Δημιουργεί μια λίστα ετικετών που περιέχει μόνο το χ

**merge(list1,list2) :** Δημιουργεί μια λίστα ετικετών τετράδων από την συνένωση των δύο λιστών list1 , list2.

**backpatch(list,z)** : Όπου το όρισμα list είναι λίστα από ετικέτες τετράδων οι οποίες έχουν το τελευταίο στοιχείο κενό. Η backpatch επισκέπτεται όλες τις τετράδες που αναγράφει η list και συμπληρώνει το τελευταίο στοιχείο με την ετικέτα z.

Σε κάθε καινούργιο block κώδικα που δημιουργείται είτε αυτό είναι το κυρίως πρόγραμμα είτε μια συνάρτηση ή διαδικασία δημιουργείται μια καινούργια τετράδα του τύπου (‘begin\_block’,το όνομα του μπλοκ αυτού ,\_,\_) το οποίο παίρνουμε σαν όρισμα στην κλήση του block όπως και μια σημαία (flag) που αναδεικνύει κατά πόσο το μπλόκ αυτό είναι το κυρίως πρόγραμμα. Αντίστοιχα στο τέλος κάθε μπλόκ δημιουργούμε την τετράδα ('end\_block',όνομα του μπλόκ,'\_','\_') και αν η σημαία είναι ίση με το 1 τότε το συγκεκριμένο μπλόκ είναι το κυρίως πρόγραμμα και δημιουργούμε την τετράδα ('halt','\_','\_','\_') πριν από την τετράδα end\_block.

Για αριθμητικές παραστάσεις υλοποιούμε την ακόλουθη γραμματική :

Expression -> Term1st( ‘+’ ή ‘-‘ Term2nd {P1})\* {P2}

{P1}: w= newTemp()

genQuad(addop,T1\_place,T2\_place,w)

T1\_place=w

{P2}: E\_place =T1\_place

To term1st υλοποιείτε πριν την επανάληψη αφου το Expression πρέπει να έχει τουλάχιστον ένα term το οποίο κρατάμε στην T1\_place . Μέσα στην επανάληψη ψάχνουμε για ‘+’ ή ’- ’ και αν βρούμε ξανακαλούμε την term την οποία κρατάμε ως T2\_place και υλοποιείτε το P1 οπού δημιουργείται μια προσωρινή μεταβλητή w την οποία θα χρησιμοποιήσουμε για αποθήκευση του αποτελέσματος της αριθμητικής πράξης και παράγεται η τετράδα που περιγράφει την αριθμητική πραξη αυτή όπου addop ειναι ‘+’ ή ’- ’. Τέλος αποθηκεύουμε την τελευταια προσωρινή μεταβλητη στο Τ1\_place για επιστροφή .Υλοποιείτε το P2 και επιστρέφεται το E\_place.

Term->Factor1(‘\*’ ή ‘/’ Factor2{P1})\*{P2}

{P1}: w=newTemp()

genQuad(mulOperator,F1\_place,F2\_place,w)

F1\_place=w

{P2}:T\_place=F1\_place

Εδώ η factor1 υλοποιείτε πριν την επανάληψη και αποθηκεύουμε το αποτέλεσμα του 1ου factor στην μεταβλητή F1\_place και αν υπάρχουν και αλλα factor2 τοτε ξανακαλείτε η factor και επείτα υλοποιείτε η P1 όπου και δημιουργείται μια προσωρινή μεταβλητή w την οποία θα χρησιμοποιήσουμε για αποθήκευση του αποτελέσματος της αριθμητικής πράξης και παράγεται η τετράδα που περιγράφει την αριθμητική πραξη αυτή όπου mulOperator ‘\*’ ή ‘/’ Τέλος αποθηκεύουμε την τελευταια προσωρινή μεταβλητη στο F1\_place για επιστροφή .Υλοποιείτε το P2 και επιστρέφεται το T\_place.

Factor->(Expression){P1} : F\_place=E\_place

| id {P1}:F\_place = Id\_place

Αν το factor είναι expression τότε το Ρ1 επιστρέφει είτε έναν αριθμό είτε μιαν μεταβλήτη αν το factor είναι αριθμητικη πράξη , αν μεσα στο factor υπάρχεί αναγνωριστικό τοτε επιστρέφει το αναγνωριστικό αυτό

Για Λογικές Παραστάσεις υλοποιούμε την ακόλουθη γραμματική :

Condition->Boolterm1{P1}(or {P2}Boolterm2{P3})\*

{P1} : ConditionTrue=Bool\_term\_true

ConditionFalse=Bool\_term\_false

{P2}: backpatch(ConditionFalse,nextquad())

{P3}: ConditionTrue=merge(ConditionTrue,Bool\_term2\_true)

ConditionFalse = bool\_term2\_false

Μέσα στο condition καλείτε η Boolterm οπού επιστρέφει μια λίστα με πρώτο ορισμα την ετικέτα της τετράδας που πρέπει να μεταβούμε σε περίπτωση true και δεύτερο όρισμα την ετικέτα της τετράδας που πρέπει να μεταβούμε σε περίπτωση false{P1}. Μεσα σε επανάληψη ψάχνουμε για τυχόν επόμενα Boolterm που διαχωρίζοντεαι με or,μετα το or κανουμε backpatch και συνενώνουμε ολες της τετραδες που υπάρχουν στο ConditionFalse με την ετικέτα της επόμενης τετράδας που θα είναι το επόμενο boolterm{Ρ2} , συνενώνουμε στην λίστα με τις ετικέτες των true την ετικέτα που θα επιστρέψουν ολες οι κλήσεις των boolterm και θέτουμε την ConditionFalse με την ετικέτα της επόμενης boolterm σε περίπτωση false που θα υλοποιείθει {Ρ3}.

Boolterm->Boolfactor1{P1} (and {P2} Boolfactor2{P3})\*

{P1}: BooltermTrue = Boolfactor1\_true

BooltermFalse = Boolfactor1\_false

{P2} backpatch (Boolterm\_true, nextQuad())

{P3} BooltermFalse = merge(BooltermFalse, Boolfactor2\_false)

BooltermTrue = Boolfactor2\_true

Καλείτε η Boolfactor και επιστρέφει μια λίστα με πρώτο ορισμα την ετικέτα της τετράδας που πρέπει να μεταβούμε σε περίπτωση true και δεύτερο όρισμα την ετικέτα της τετράδας που πρέπει να μεταβούμε σε περίπτωση false {Ρ1} Μεσα σε επανάληψη ψάχνουμε για τυχόν επόμενα Boolfactor που διαχωρίζοντεαι με and,μετα το and κανουμε backpatch και συνενώνουμε ολες της τετραδες που υπάρχουν στο Boolterm\_true με την ετικέτα της επόμενης τετράδας που θα είναι το επόμενο boolfactor{Ρ2} έτσι ώστε να ελεγχθεί το επόμενο boolfactor αν είναι αληθές .Επείτα συνενώνουμε την λιστα BooltermFalse με τις υπόλοιπες ετικέτες που θα επιστρέψει η Boolfactor2 σε περίπτωση false και θέτουμε την BooltermTrue ως την ετικέτα της επόμενης boolfactor σε περίπτωση true {Ρ3}

Boolterm ->Condition{P1}: Boolterm\_true = Condition\_true

Boolterm\_false = Condition\_false

|Expression relOp Expression {P1} : Boolterm\_true = makelist(nextQuad())

genQuad(relop,E1\_palce,E2\_place,’\_’)

Boolterm\_false = makelist(nextQuad())

genQuad(“jump”,”\_”,”\_”,”\_”)

Αν το Boolterm είναι Condition τότε γίνετε μια απλή μεταφορά των ετικετών από το Condition\_true και Condition\_false στα αντίστοιχα Boolterm\_true και Boolterm\_false ενώ διαφορετικά η Boolterm\_true δημιουργεί μια λίστα με την ετικέτα της επόμενης τετράδας Δημιουργείτε η τετράδα που περιγράφει την λογική παράσταση και η Boolterm\_false δημιουργεί μια λιστα με την ετικέτα της επόμενης τετράδας που θα δημιουργηθεί και παράγεται η τετράδα οπου θα υλοποιείθεί το jump που γίνεται backpatch προηγουμενος για να προσθεθεί ο αριθμός της τετραδας στοχος που πρεπει να ακολουθηθεί.

Σε περίπτωση κλήσης μιας διαδικασίας μεσα στην callStat() καλείται η actualParList μεσα στην οποία καλέιτε η actualParItem και δημιουργά τις τετράδες για όλες τις παραμέτρους της καλούσας ,για πέρασμα με τιμή genQuad('par',this\_Expression,'CV','\_') και για περασμα με αναφορά genQuad('par',this\_name,'REF','\_') όπου το this \_expression και this\_name τα ονόματα των παραμέτρων .Έπειτα δημιουργούμε την τετράδα genQuad('call',name,'\_','\_') οπου name το όνομα της καλούσας .

Στην περίπτωση κλήσης συνάρτησης που γίνετε μεσω του factor , καλείτε η idtail() και καλεί την actualParList οπου και δημιουργόυνται οι τετράδες για τις παραμέτρους με αναφορα η παραμέτρους με τιμή όπως και πριν ,επείτα δημιουργείτε μια καινούργια προσωρινή μεταβλητη η οποία θα είναι η επιστρεφόμενη τιμή της συνάρτησης και παραγοναι οι τετράδες genQuad('par',w,'RET','\_') και genQuad('call',name,'\_','\_') οπου w η προσωρινή μεταβλητή και ναμε το όνομα της συνάρτησης .

Για την εντολή return παράγεται μέσα στη return\_stat() η τετράδα genQuad ('retv',E\_Place,'\_','\_') οπου E\_place το επιστρεφόμενο expression.

Για εκχώρηση μεταβλητών παράγεται μεσα στην Assignment\_Stat() δημιουργείτε η τετράδα genQuad(':=',E\_Place,'\_',assignmentId) οπου E\_place το expression και assignmentId το αναγνωριστικό που θα γίνει η εκχώρηση σε αυτό.

Για την Δομή της επανάληψης while υλοποιούμε την γραμματική μεσα στην While\_Stat

S -> while {P1} Condition do {P2} Statement {P3}

{P1}: quadCounter = nextquad()

{P2}: backpatch(Conditiontrue,nextquad())

{P3}: genquad(“jump”,”\_”,”\_”, quadCounter)

backpatch(Condition false,nextquad())

Στην quadCounter κρατάμε την ετικέτα της επόμενης τετράδας που δηλαδή την ετικέτα του condition του while στη συνέχεια κάνουμε backpatch τις τετράδες που επέστρεψε το condition για την περίπτωση true και μετά την εκτέλεση των Statement του σώματος της επανάληψης δημιουργείτε η τετράδα του jump με τελευταίο όρισμα το quadCounter και έπειτα γίνεται backpatch στις τετράδες που επέστρεψε το condition για την περίπτωση false με την ετικέτα της επόμενης τετράδας δηλαδή της τετράδας ακριβώς έξω από την επανάληψη.

Για την δομή της if-else υλοποιούμε την γραμματική μέσα στην If\_Stat

S -> if Condition then {P1} Statement {P2} TAIL {P3}

TAIL -> else Statement | TAIL -> e

{P1}: backpatch(Conditiontrue,nextquad())

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

genquad(“jump”,”\_”,”\_”,”\_”)

backpatch(Condition false,nextquad())

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

Καλείτε η condition οπου και παράγει τις λίστες των true και false έπειτα κάνουμε backpatch την τετράδα από το Conditiontrue με την ετικέτα της επόμενης τετράδας όπου και υλοποιείτε το σώμα της if και στη συνέχεια δημιουργείται η μεταβλητή iflist όπου και δημιουργεί μια καινούργια λίστα με την ετικέτα της επόμενης τετράδας .Δημιουργείτε η τετράδα του jump και έπειτα κάνουμε backpatch με την ετικέτα της επόμενης τετράδας την λίστα με τις ετικέτες που επέστρεψε το Condition false ώστε να εκτελεστούν η εντολές που υπάρχουν στο TAIL. Αν αυτό δεν είναι κενό. Έπειτα κάνουμε backpatch με την ετικέτα της επόμενης τετράδας δηλαδή την τετράδα εχτός της if, την iflist εξασφαλίζοντας έτσι ότι εάν το condition της If είναι αληθές οι εντολές μέσα στο σώμα της else δεν θα εκτελεστούν .

Για την περίπτωση εισόδου (input) μέσα στην Input\_Stat δημιουργείτε η τετράδα genQuad('inp',Id\_Place,'\_','\_') όπου Id\_Place το αναγνωριστικό που θέλουμε να εισάγουμε .

Για την περίπτωση εκτύπωσης (print) μέσα στην Print\_Stat δημιουργείτε η τετράδα genQuad('out',E\_Place,'\_','\_')όπου Ε\_Place το expression που θέλουμε να τυπώσουμε .

Για την δομή της Switchcase εφαρμόζουμε την γραμματική μεσα στην Switchcase\_Stat()

Switchcase {P1}

(case(condition){P2} statements{P3})\*

default statements {P4}

{P1} : outList=emptyList()

{P2}: backpatch(ConditionTrue,nextQuad())

{P3} outJump = makeList(nextQuad())

genQuad('jump','\_','\_','\_')

outList=merge(outList,outJump)

backPatch(ConditionFalse,nextQuad())

{P4}: backPatch(outList,nextQuad())

Η δομή Switchcase ελέγχει τα condition μετά τα case και μόλις ένα από αυτά βρεθεί αληθείς τότε εκτελούνται τα statements του case αυτού και μεταβαίνουμε έξω από την Switchcase , αν κανένα από τα condition δεν είναι αληθές τότε μεταβαίνουμε στην default και εκτελούνται τα statements της. Καλείτε η condition και μας επιστρέφει της δύο λίστες από ετικέτες για της τετράδες που θα πρέπει να ακολουθήσουμε για τις περιπτώσεις true και false .Έπειτα Δημιουργούμε την outlist μια κενή λίστα όπου θα ομαδοποιήσουμε τις ετικέτες από τις τετράδες που θα πρέπει να ακολουθήσουμε για έξοδο σε κάθε case{Ρ1}. Αφού εκτελεστεί η condition κάνουμε backpatch στις τετράδες που επέστρεψε το ConditionTrue και την επόμενη τετράδα που είναι η πρώτη εντολή του statements του συγκεκριμένου case{Ρ2}.Αφού εκτελεστούν οι εντολές του statements τότε δημιουργούμε μια νέα λίστα outjump με την ετικέτα από την τελευταία τετράδα του statements.Δημιουργείτε η επόμενη τετράδα που, αυτή του jump και μετά κάνουμε backpatch τις τετράδες που επέστρεψε η ConditionFalse με την ετικέτα της τετράδας του επόμενου case{Ρ3}.Και έπειτα εκτός επανάληψης και μετά από την εκτέλεση των statements του default κάνουμε backpatch όλες τις τετράδες των jump δηλαδή το outlist με την ετικέτα της πρώτης εντολής εχτός του switchCase.

Για την δομή της επανάληψης Forcase εφαρμόζουμε την γραμματική μεσα στην Forcase\_Stat ()

Forcase {P1}

(case(condition ){P2}statements {P3})

Default statements

{P1} : StartOfForcase=nextQuad()

{P2} : backPatch(ConditionTrue,nextQuad())

{P3} : genQuad('jump','\_','\_',StartOfForcase)

backPatch(ConditionFalse,nextQuad())

Η δομή της επανάληψης Forcase ελέγχει τα condition όλων των case και μόλις μια από αυτές βρεθεί αληθής τότε εκτελούνται τα statements τις και μεταβαίνουμε στην αρχή της Forcase αλλιώς αν καμία από τις case δεν ισχύει τότε μεταβαίνουμε στην Default όπου και εκτελούνται τα statements και μεταβαίνουμε έξω από την forcase. Καλείτε η condition και μας επιστρέφει της δύο λίστες από ετικέτες για της τετράδες που θα πρέπει να ακολουθήσουμε για τις περιπτώσεις true και false .Έπειτα δημιουργούμε την StartOfForcase μια μεταβλητή που περιέχει την πρώτη τετράδα του forcase που θα χρειαστούμε πιο μετά{Ρ1} .Μετά από την εκτέλεση του condition κάνουμε backpatch των τετράδων που επέστρεψε το ConditionTrue και την επόμενη τετράδα που είναι η πρώτη εντολή του statements του συγκεκριμένου case{Ρ2} αφού εκτελεστούν τα statements τότε δημιουργούνται οι τετράδες των jump που μας μεταφέρουν στην αρχή της Forcase και κάνουμε backpatch των τετράδων που επέστρεψε το ConditionFalse και την επόμενη τετράδα που είναι η επόμενη case.

Για την δομή της επανάληψης Incase εφαρμόζουμε την γραμματική μεσα στην Incase\_Stat ()

Incase {P1}

(case(condition){P2}statements{P3})\*{P4}

{P1} : P1\_Quad=nextQuad()

w=newTemp()

genQuad(':=','1','\_',w)

{P2} : backPatch(ConditionTrue,nextQuad())

{P3} : genQuad(':=','0','\_',w)

backPatch(ConditionFalse,nextQuad())

{P4} : genQuad('=',w,'0',P1\_Quad)

Η δομή της επανάληψης Incase ελέγχει τα condition όλων των case με την σειρά .Για κάθε condition που ισχύει εκτελούνται τα statements του και αφού ελεγχθούν όλα τα condition αν έστω και ένα από αυτά τότε μεταβαίνουμε στην αρχή της Incase αλλιώς μεταβαίνουμε έξω από αυτήν. Δημιουργούμε την P1\_Quad μια μεταβλητή με τιμή την ετικέτα της πρώτης τετράδας της Incase που θα χρειαστούμε για να υλοποιήσουμε την επανάληψη. Δημιουργούμε μια καινούργια προσωρινή μεταβλητή και δημιουργείτε μια τετράδα ανάθεσης όπου και εξισώνουμε την προσωρινή μεταβλητή με 1 {Ρ1}. Καλείτε η condition και μας επιστρέφει της δύο λίστες από ετικέτες για της τετράδες που θα πρέπει να ακολουθήσουμε για τις περιπτώσεις true και false και κάνουμε backpatch των τετράδων που επέστρεψε το ConditionTrue με την ετικέτα της επόμενης τετράδας η οποία είναι η επόμενη condition.{Ρ2}.Αν η condition ισχύει τότε δημιουργείτε μια τετράδα ανάθεσης όπου και εξισώνουμε την προσωρινή μεταβλητή με 0 και κάνουμε backpatch των τετράδων που επέστρεψε το ConditionFalse με την ετικέτα της επόμενης τετράδας η οποία είναι η επόμενη condition.{P3} . Μετά το πέρας όλων των condition δημιουργείτε μια καινούργια τετράδα για έλεγχο ισότητας της προσωρινής μεταβλητής με το 0 αν ισχύει τότε μεταβαίνουμε στην αρχή του Incase.

Στο τέλος της παραγωγής του ενδιάμεσου κώδικα όλες οι τετράδες τυπώνονται μέσα σε ένα αρχείο .int για καλύτερη απεικόνιση των τετράδων που παράγονται και εφόσον το πρόγραμμα δεν έχει κάποια συνάρτηση ή διαδικασία τυπώνετε ένα ισοδύναμο πρόγραμμα γραμμένο στην C γλώσσα .

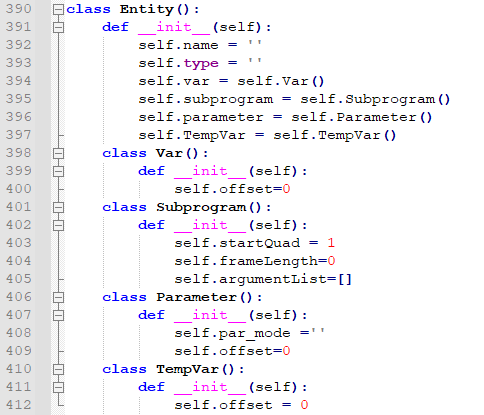
**Παραγωγή Πίνακα Συμβόλων**

Στην φάση της παραγωγής του πίνακα συμβόλων δημιουργούμε ένα πίνακα από σύμβολα scopes entities και arguments. Ένα Scope θα δημιουργείτε για το κυρίως πρόγραμμα και όλα τα υπό προγράμματα και θα περιέχει entities για κάθε μεταβλητή , κάθε προσωρινή μεταβλητή , παράμετρο και υπο πρόγραμμα που ανήκει στο κάθε scope.Τέλος arguments είναι οι τυπικές παράμετροι του κάθε υπο προγράμματος . Το κάθε υπο πρόγραμμα θα δημιουργήσει δικό του scope και έπειτα θα δημιουργήσει entities για κάθε argument που του ανήκουν.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Για την υλοποίηση κάθε scope δημιουργήσαμε ένα class Scope για το οποίο κρατάμε το όνομα του κάθε scope , μια λίστα για τα entities που περιέχει το βάθος φωλιάσματος του συγκεκριμένου scope και αν δεν είναι το scope του κυρίως προγράμματος τότε και το scope πατέρα.

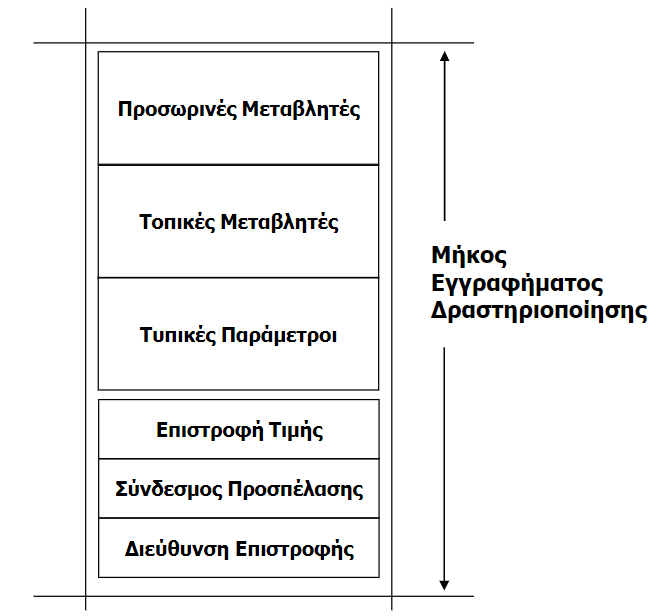
****

Για την υλοποίηση κάθε entity δημιουργήσαμε ένα class Entity το οποίο διαχωρίζεται σε υπο κλάσεις Var , Subprogram ,Parameter ,TempVar. Το κάθε Entity έχει το δικό του όνομα , τύπο VAR για μεταβλητές PARAM για παραμέτρους TEMP για προσωρινές μεταβλητές και SUBPR για υποπρογράμματα , και αρχικοποιούμε τα var,subprogram,parameter,TempVar έτσι ώστε να μπορούν να ενημερωθούν ανάλογα στην συνέχεια . Για την κάθε υπο κλάση κρατάμε πληροφορίες που θα χρησιμεύσουν στη παραγωγή τελικού κώδικα. Για την υπο κλάση Var και TempVar κρατάμε την απόσταση από την αρχή του εγγραφήματος δραστηριοποίησης\*\*, για το Subprogram κρατάμε την αρχική τετράδα του συγκεκριμένου υποπρογράμματος από τον ενδιάμεσο κώδικα , το frameLength δηλαδή το μήκος του εγγραφήματος δραστηριοποίησης\*\* και μια λίστα που θα περιέχει τις τυπικές παραμέτρους του .Τέλος για την Parameter κρατάμε τον τρόπο περάσματος της παραμέτρου , αν είναι με τιμή ή με αναφορά και την απόσταση από την αρχή του εγγραφήματος δραστηριοποίησης\*\*.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Για την υλοποίηση κάθε Argument δημιουργήσαμε ένα class Argument και κρατάμε πληροφορία για το όνομα του και τον τρόπο περάσματος του.



Για ευκολότερη διαχείριση του πίνακα συμβόλων δημιουργήσαμε τις βοηθητικές συναρτήσεις :

**add\_scope()** όπου δημιουργούμε και προσθέτουμε ένα καινούργιο scope σε μια λίστα από scopes , αν η λίστα αυτή είναι κενή πριν την προσθήκη του καινούργιου scope τότε το scope αυτό είναι το κυρίως πρόγραμμα με βάθος φωλιάσματος ίσο με 0 αλλιώς έχει βάθος ίσο με το μέγεθος της λίστας .Τέλος προσθέτουμε το scope στην λίστα και θέτουμε ως τρέχον scope αυτό που μόλις προσθέθηκε.

**close\_scope()** Αφού τελειώσει η μετάφραση του συγκεκριμένου scope θέτουμε ως τρέχον scope το scope πατέρα και διαγράφουμε το συγκεκριμένο scope από την λίστα .

**add\_entities()** Προσθέτουμε ένα entity στην λίστα των entities του τρέχοντος scope

**add\_arguments()** Προσθέτουμε ένα argument στην λίστα του τρέχοντος entity εφόσον αυτό είναι υποπρόγραμμα.

**pass\_parameters()** Αφού κάθε argument ενός υπο προγράμματος θα πρέπει να μετατραπεί σε entity του επόμενου scope όταν καλείτε η συνάρτηση αυτή παίρνουμε από το scope πατέρα του τρέχοντος scope το τελευταίο στοιχείο της λίστας των entity δηλαδή το υποπρόγραμμα και διασχίζουμε την λίστα των arguments δημιουργώντας για το κάθε ένα , ένα καινούργιο entity .

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

**compute\_offset()** Εδώ υπολογίζουμε την απόσταση από την αρχή του εγγραφήματος δραστηριοποίησης η οποία ισούται με 12bits +(4\*τον αριθμό των entity που το προηγούνται ) δεν συμπεριλαμβάνονται τα entities που είναι υποπρογράμματα . Γι’ αυτό και διασχίζουμε όλοι την λίστα των entity στο τρέχον scope.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

**print\_Symbols()**  Όπου και τυπώνουμε το τρέχον scope σε ένα αρχείο .txt

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Τα scopes δημιουργούνται δυναμικά κάθε φορά που καλείται η συνάρτηση Blocks αφου τελειώσουν η συναρτήσεις Declarations και Subprograms και αν το συγκεκριμένο block δεν είναι το κυρίως πρόγραμμα τότε υπολογίζουμε την τετράδα στην οποία ξεκινά και την περνάμε στην παράμετρο startQuad του entity . Επίσης πριν το κλείσιμο του block αν το συγκεκριμένο block δεν είναι το κυρίως πρόγραμμα τότε υπολογίζουμε το μήκος του εγγραφήματος δραστηριοποίησης του που είναι όσο και το offset του τελευταίου entity του πατέρα scope συν 4 bits. Στην συνέχεια τυπώνουμε το scope και το διαγράφουμε.

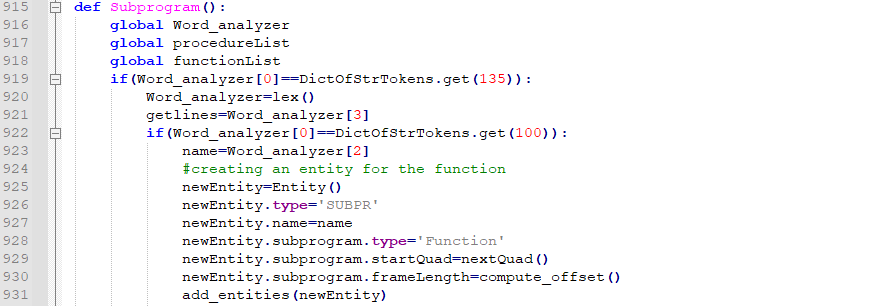
Τα entities δημιουργούνται ανάλογα με τον τύπο τους. Τα entities που θα δημιουργηθούν για της μεταβλητές δημιουργούνται δυναμικά κάθε φορά που καλείται η συνάρτηση VarList

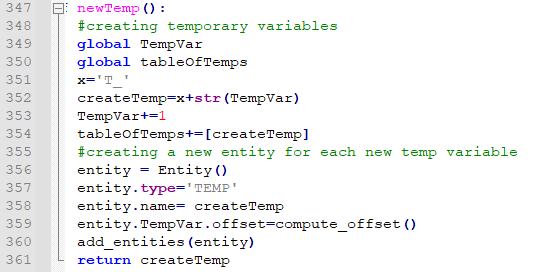
Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Και για τύπο περνάμε την λέξη ‘VAR’ όνομα το όνομα της μεταβλητής , υπολογίζουμε την απόσταση από την αρχή του εγγραφήματος δραστηριοποίησης και τέλος το προσθέτουμε στην λίστα των entities του τρέχοντος scope.

Τα entities που θα δημιουργηθούν για τα υποπρογράμματα δημιουργούνται δυναμικά κάθε φορά που καλείται η συνάρτηση Subprogram οπού και για τον τύπο τους ,τους περνάμε την λέξη 'SUBPR' για όνομα το όνομα του υποπρογράμματος , τύπο υποπρογράμματος περνάμε ανάλογα με το αν το υποπρόγραμμα είναι διαδικασία ή συνάρτηση την λέξη ‘Procedure’ ή ‘Function’ , υπολογίζουμε την πρώτη τετράδα του υποπρογράμματος και τέλος το μήκος του εγγραφήματος δραστηριοποίησης του.



Τα entities που θα δημιουργηθούν για τις προσωρινές μεταβλητές δημιουργούνται δυναμικά κάθε φορά που καλείται η συνάρτηση newTemp όπου και δημιουργούνται η προσωρινές μεταβλητές. Ως τύπο του entity περνάμε τη λέξη 'TEMP' ,ως όνομα το όνομα της προσωρινής μεταβλητής , υπολογίζουμε την απόσταση από την αρχή του εγγραφήματος δραστηριοποίησης και τέλος το προσθέτουμε στην λίστα των entities του τρέχοντος scope.

Και τέλος τα arguments δημιουργούνται δυναμικά στην κλήση της συνάρτησης FormalparItem όπου και δημιουργούνται η τυπικές παράμετροι ενός υποπρογράμματος. Οπού και περνάμε για όνομα του argument το όνομα της τυπικής παραμέτρου και σαν τρόπο ανάλογα με το την δήλωση του , ‘in’ με τιμή και περνάμε την λέξη ‘CV’ ή ‘inout’ με αναφορά και περνάμε την λέξη ‘REF’ .Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

**Παραγωγή τελικού κώδικα**

Ο τελικός κώδικας παράγεται με βάση τις τετράδες του ενδιάμεσου κώδικα και του πίνακα συμβόλων . Ο παραγόμενος κώδικας τυπώνετε σε ένα αρχείο .asm και είναι βασισμένος στην αρχιτεκτονική MIPS. Ο τελικός κώδικας παράγεται στην συνάρτηση final() και για την ορθή παραγωγή του υλοποιήσαμε τις εξής βοηθητικές συναρτήσεις :

**search(n) :** Η συνάρτηση αυτή διατρέχει την λίστα των scope και για κάθε scope την λίστα των entity που εμπεριέχονται για να βρει και επιστρέφει το πρώτο entity με όνομα ‘n’, επιστρέφει επίσης το scope στο οποίο βρίσκετε το entity αυτό σαν ένα ζεύγος .Αν δεν το βρει επιστρέφει το ζεύγος (0,0)

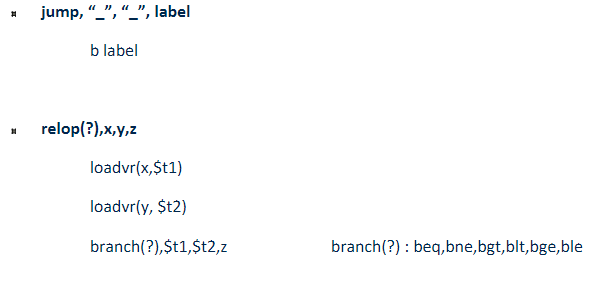
**gnlvcode(name):** Η συνάρτηση gnlvcode μεταφέρει στον $t0 την διεύθυνση μιας μη τοπικής μεταβλητής . Για να βρει πόσα επίπεδα επάνω βρίσκεται η μεταβλητή καλεί την searchγια να βρεί το scope στο οποίο βρίσκεται το entity name . Τυπώνει lw $t0,-4($sp) στο αρχείο του κώδικα για την στοίβα του γονέα και τυπώνει lw $t0,-4($t0) όσες φορές όσες και η διαφορά του βάθους φωλιάσματος του τρέχοντος scope με το scope το οποίο επέστρεψε η search μείον 1 γιατί έχει τυπωθεί για την στοίβα του γονέα. Έτσι παίρνουμε την στοίβα του προγόνου που έχει την μεταβλητή αυτή. Τέλος τυπώνουμε addi $t0,$t0,-offset για να πάρουμε τη διεύθυνση της μη τοπικής μεταβλητής.

**loadvr(v,r):** η συνάρτηση αυτή καλείται για την μεταφορά δεδομένων ‘v’ στον καταχωρητή ‘r’.Καλώντας την συνάρτηση search παίρνουμε το ζεύγος scope και entity που θέλουμε να εξετάσουμε. Διακρίνοντας περιπτώσεις για το είδος του entity παράγουμε τον ανάλογο τελικό κώδικα .Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

**storerv(r,v):** Η συνάρτηση storerv καλείται για την μεταφορά δεδομένων από τον καταχωρητή ‘r’ στην μνήμη ‘v’.Καλείται η search και με βάση το ζεύγος τον scope και entity που επιστρέφονται διακρίνονται οι παρακάτω περιπτώσεις και παράγεται ο κατάλληλος τελικός κώδικας .Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Η συνάρτηση final καλείται μέσα στην συνάρτηση blocks και πριν να γίνει η διαγραφή του τρέχοντος scope .Ο τελικός κώδικας παράγεται με βάση της τετράδες που παράγει το κάθε τρέχον scope. Διατρέχουμε τις τετράδες αυτές και ανάλογα με τις εντολές αυτές παράγουμε τον τελικό κώδικα .

Για τις εντολές αλμάτων :

Όπου relop οι λογικές παραστάσεις .

Για την εκχώρηση τιμών :

Εικόνα που περιέχει κείμενο

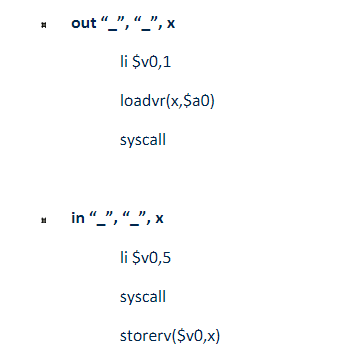
Περιγραφή που δημιουργήθηκε αυτόματα

Για τις εντολές αριθμητικών πράξεων :

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Για τις εντολές Εισόδου-Εξόδου :



Για επιστροφή τιμής συνάρτησης :

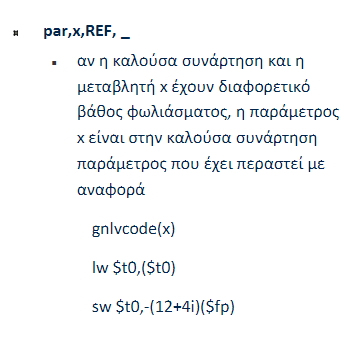
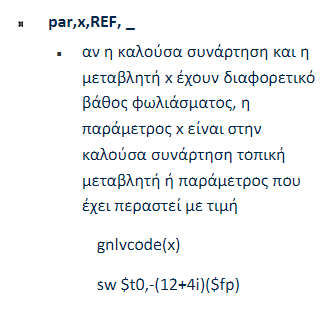
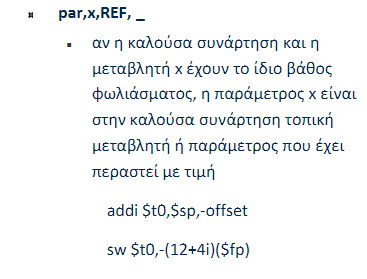
Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Για τις παραμέτρους συνάρτησης πριν γίνει το πέρασμα της πρώτης παραμέτρου τοποθετούμε τον $fp να δείχνει στην στοίβα της συνάρτησης που θα δημιουργηθεί με την εντολή addi$fp,$sp,framelength , για να ξέρουμε πότε θα τυπωθεί η εντολή αυτή δημιουργήσαμε έναν μετρητή με αρχική τιμή 0 και τον αυξάνουμε κατά ένα κάθε φορά που περνούμε μια παράμετρο.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματαΕικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματαΕικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Για την κλήση μιας συνάρτησης αν η καλούσα και η κληθείσα συνάρτηση έχουν το ίδιο βάθος φωλιάσματος τότε τυπώνονται οι εντολές :

Lw $t0,-4($sp) sw $t0,-4($fp)

Αλλιώς αν έχουν διαφορετικό βάθος φωλιάσματος :

Sw $sp,-4($fp)

Στην συνέχεια μεταφέρουμε τον δείκτη στοίβας στην κληθείσα ,καλούμε την συνάρτηση και επιστρέφοντας πίσω σε αυτήν παίρνουμε πίσω τον δείκτη στοίβας στην καλούσα.

addi$sp,$sp,framelength jal f addi$sp,$sp,-framelength

Με κάθε αρχή συνάρτησης αποθηκεύουμε την διεύθυνση επιστροφής της η οποία βρίσκετε στην $ra τυπώνοντας την εντολή

Sw $ra,($sp)

Και στο τέλος κάθε συνάρτησης παίρνουμε την διεύθυνση επιστροφής της συνάρτησης και την καταχωρούμε στην $ra όπου μέσα του οποίου επιστρέφουμε στην καλούσα συνάρτηση .

Lw $ra,($sp)

Jr $ra

Αφού το κυρίως πρόγραμμα δεν μεταφράζεται πρώτο όταν βρούμε την αρχή του κυρίως προγράμματος γράφουμε στην αρχή του αρχείου ένα άλμα που μας μεταβαίνει στην ετικέτα του κυρίως προγράμματος.

J Lmain

Στην συνέχεια μέσα στην ετικέτα του κυρίως προγράμματος πρέπει να κατεβάσουμε τον $sp κατά το framelength του κυρίως προγραμματος και να αποθηκεύσουμε στον $s0 το εγγράφημα δραστηριοποίησης του κυρίως προγράμματος για ευκολότερη πρόσβαση στις global μεταβλητές.

Addi $sp,$sp,framelength move $s0,$sp