ΜΕΤΑΦΡΑΣΤΕΣ

Compiler της γλώσσας EEL

Χομόνδοζλης Πασχάλης Βέργας Αιμιλιανός

Περιεχόμενα

Γενικά

Η Early Experimental Language ή EEL εν συντομία, είναι μία LL γλώσσα και ο compiler της είναι LL(1). Διαβάζεται απο αριστερά προς τα δεξιά και δημιουργείται το πιο αριστερό (leftmost) δέντρο παραγωγής βλέποντας μπροστά κατά ένα χαρακτήρα τη φορά. Ο compiler που κατασκευάσαμε αποτελείται τα εξής βασικά κομμάτια τα οποία θα αναπτυχθούν παρακάτω :

1. Λεκτικός Αναλυτής
2. Συντακτικός Αναλυτής
3. Παραγωγή Ενδιάμεσου Κώδικα
4. Παραγωγή Πίνακα Συμβόλων
5. Παραγωγή Τελικού Κώδικα

Όλα αυτά τα τμήματα λειτουργούν μαζί για να παράξουν τον κώδικα assembly. Ο κώδικας assembly είναι για επεξεργαστές με αρχιτεκτονική MIPS.

Γραματτική της γλώσσας EEL

Η γραμματική της γλώσσας EEL είναι η εξής :

```
<program> ::= program id <block> endprogram
<block> ::= <declarations> <subprograms> <statements>
<declarations> ::= ε | declare <varlist> enddeclare
<varlist> ::= ε | id (, id)*
<subprograms> ::= (<procorfunc>) *
<procorfunc> ::= procedure id <procorfuncbody> endprocedure |
                 function id <procorfuncbody> endfunction
<procorfuncbody> ::= <formalpars> <block>
<formalpars> ::= (<formalparlist>)
<formalparlist> ::= <formalparitem> (, <formalparitem>) * | ε
<formalparitem> ::= in id | inout id
<statements> ::= <statement> (; <statement>) *
<statement> ::= ε |
                <assignment-stat> |
                <if-stat> |
                <while-stat> |
                <repeat-stat> |
                <exit-stat> |
                <switch-stat> |
                <forcase-stat> |
                <call-stat> |
                <return-stat>
```

<input-stat> |
<print-stat>
<assignment-stat> ::= id := <expression>
<if-stat> ::= **if** <condition> **then** <statements> <elsepart> **endif**
<elsepart> ::= ε | **else** <statements>
<repeat-stat> ::= **repeat** <statements> **endrepeat**
<exit-stat> ::= **exit**
<while-stat> ::= **while** <condition> <statements> **endwhile**
<switch-stat> ::= **switch** <expression>
(**case** <expression> **:** <statements>) +
endswitch
<forcase-stat> ::= **forcase**
(when <condition> **:** <statements>) +
endforcase
<call-stat> ::= **call** id <actualpars>
<return-stat> ::= **return** <expression>
<print-stat> ::= **print** <expression>
<input-stat> ::= **input** id
<actualpars> ::= **(**<actualparlist>**)**
<actualparlist> ::= <actualparitem> (**,** <actualparitem>) ***** | ε
<actualparitem> ::= **in** <expression> | **inout** id
<return-stat> ::= **return** <expression>
<condition> ::= <boolterm> (**or** <boolterm) ***
<boolterm> ::= <boolfactor> (**and** <boolfactor) ***
<boolfactor> ::= **not [**<condition>**]** | **[**<condition>**]** |
<expression> <relational-oper> <expression> |
true | **false**
<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>

Λεκτικός Αναλυτής

Περιγραφή

Ο λεκτικός αναλυτής, όπως προαναφέρθηκε, διαβάζει έναν χαρακτήρα τη φορά απ' το αρχείο .eel και το τοποθετεί σε έναν buffer. Όταν αναγνωριστεί κάποιο τερματικό σύμβολο της γλώσσας, τότε επιστρέφει τον buffer στη μορφή ενός string.

Οι συναρτήσεις που αποτελούν τον λεκτικό αναλυτή είναι οι εξής :

1. `def getNextChar():`
2. `def getLex():`

Η getNextChar() καλείται μόνο απ' την getLex(), διαβάζει και επιστρέφει τον επόμενο χαρακτήρα στο αρχείο κώδικα EEL. Επίσης, αυξάνει τον αριθμό χαρακτήρων που έχουν διαβαστεί στην τρέχουσα γραμμή.

Η getLex() είναι ουσιαστικά ο λεκτικός αναλυτής. Διαβάζει το αρχείο και αναγνωρίζει προκαθορισμένα tokens (π.χ. 'while', 'endforcase', 'program') που ανήκουν στην γλώσσα EEL, λέξεις που ακολουθούν συγκεκριμένους κανόνες και προορίζονται για ονόματα μεταβλητών, συναρτήσεων, διαδικασιών και αριθμητικές τιμές. Οτιδήποτε αναγνωρίσει, δημιουργεί ένα αντικείμενο Token και το τοποθετεί στην κοινόχρηστη μεταβλητή 'token'. Οι αριθμητικές τιμές θα πρεπεί να είναι φραγμένες μεταξύ -32767 και +32767. Οι λέξεις που δεν είναι προκαθορισμένες αλλά ούτε αριθμητικές τιμές πρέπει να μην ξεκινάνε με αριθμό και το μέγεθός τους να είναι μικρότερο των 30 χαρακτήρων. Αν βρεθεί αριθμός στην αρχή ενός token ακολουθούμενος από κάποιο γράμμα τότε τυπώνεται μήνυμα λάθους. Φυσικά, οποιοσδήποτε λευκός χαρακτήρας βρεθεί απ' τη μία τερματίζει την συνέχιση τοποθέτησης χαρακτήρων στον buffer και απ' την άλλη αγνοείται και δεν επιστρέφεται ούτε μόνος του, ούτε ως μέρος κάποιου token. Λευκοί χαρακτήρες είναι τα tabs, spaces και newlines. Επιπροσθέτως, αγνοεί σχόλια τα οποία ενδεχομένως μπορεί να υπάρχουν. Αν βρεθεί το token '//' τότε αγνοεί όλους τους χαρακτήρες μέχρι το τέλος της γραμμής ενώ αν βρεθεί το token '/*' τότε αγνοεί όλους τους χαρακτήρες, ανεξαρτήτος γραμμής, μέχρι να συναντήσει το token '*/'. Σε περίπτωση που δεν βρεθεί το token '*/' τότε τυπώνεται αντίστοιχο μήνυμα λάθους.

Παράλληλα, σε κάποιες περιπτώσεις ο λεκτικός αναλυτής πρέπει να κοιτάξει ένα χαρακτήρα μπροστά για να ολοκληρώσει ένα token. Για παράδειγμα, το token '<' μπορεί να είναι είτε το token '<>' είτε το token '<=' ή ακόμα και το ίδιο το token '<'. Γι' αυτό τον λόγο όταν συναντάμε τον χαρακτήρα '<' ελέγχουμε τον επόμενο χαρακτήρα. Αν είναι ένας απ' τους χαρακτήρες {'>','='} τότε τον δεχόμαστε και δημιουργούμε το αντίστοιχο Token αντικείμενο. Αν, ωστόσο, δεν ανήκει στο παραπάνω σετ, τότε δημιουργούμε ένα αντικείμενο Token για το token '<' και γυρνάμε τον file pointer ένα χαρακτήρα πίσω ώστε να μην "καταναλωθεί" ένας έξτρα χαρακτήρας.

Δομές

Η δομή που χρησιμοποιείται απ' τον λεκτικό αναλυτή είναι η κλάση Token.

```
class Token:
    # Constructor
    # tokenValue: the token itself
    # tokenType: the type of the token
    # lineNo: the line of the token in the EEL file
    # charNo: the position of the character in the EEL file
    def __init__(self, tokenValue, tokenType, lineNo, charNo):
        self.tokenValue = tokenValue
        self.tokenType = tokenType
        self.lineNo = lineNo
        self.charNo = charNo
```

Ο λεκτικός αναλυτής δημιουργεί αντικείμενα αυτής της κλάσης ώστε να χρησιμοποιηθούν απ' τον συντακτικό αναλυτή και τον πίνακα συμβόλων. Σε αυτά τα αντικείμενα αποθηκεύουμε την τιμή του token, την γραμμή που βρέθηκε, την θέση του πρώτου χαρακτήρα του και τον τύπο του token. Για τους τύπους tokens δημιουργήσαμε το λεξικό tokens που συνδέθηκε με την enumeration κλάση TokenType. Έτσι, επιτυγχάνουμε αναγνωσιμότητα και εύκολη χρήση όταν η γραμμές κώδικα φτάνουν τις χιλιάδες.

Συντακτικός Αναλυτής

Περιγραφή

Ο συντακτικός αναλυτής είναι η ραχοκοκαλιά του μεταφραστή μας. Σε αυτόν βασίζονται οι συναρτήσεις του πίνακα συμβόλων και του τελικού κώδικα. Ο λεκτικός αναλυτής καλείται απ' αυτόν, και μόνο απ' αυτόν. Αποτελείται από 39 συναρτήσεις που υλοποιούν την γραμματική της γλώσσας EEL.

Σκοπός του είναι ο έλεγχος ότι ο πηγαίος κώδικας ακολουθεί τους κανόνες της γραμματικής και ενημερώνει τον χρήστη με τον ανάλογο μήνυμα σε περίπτωση που βρεθεί κάποιο λάθος. Καλεί την συνάρτηση getLex() (aka λεκτικός αναλυτής) η οποία "γεμίζει" την κοινόχρηστη μεταβλητή token και οι συναρτήσεις του συντακτικού αναλυτή την χρησιμοποιούν ώστε να εκπληρώνονται τα πολλά 'if statements' που υπάρχουν μέσα σε κάθε μία.

Ενδιάμεσος Κώδικας

Περιγραφή

Ο ενδιάμεσος κώδικας παράγεται κατά τη διάρκεια της συντακτικής ανάλυσης. Δημιουργεί τετράδες-quads με βασικές πράξεις που, αργότερα, θα μεταφραστούν σε κώδικα assembly. Για να επιτευχθεί αυτό ορίσαμε 7 βοηθητικές συναρτήσεις. Αυτές είναι οι εξής :

1. **def** genQuad(name=None, op1="_", op2="_", op3="_"):

   Δημιουργεί ένα αντικείμενο Quad και το τοποθετεί στην κοινόχρηστη λίστα quadList. Επίσης, αυξάνει την κοινόχρηστη μεταβλητή labelID που χρησιμεύει στην αρίθμηση των τετράδων.

2. **def** nextQuad():

   Επιστρέφει την ετικέτα της επόμενης προς παραγωγή τετράδας. (Θα μπορούσε να παραλειφθεί καθώς η μεταβλητή που επιστρέφεται δεν αυξάνεται τοπικά αλλά κοινόχρηστα στην genQuad)

3. **def** newTemp():

   Συνθέτει και επιστρέφει μια νέα προσωρινή μεταβλητή. Αυξάνει τον κοινόχρηστο μετρητή tempVarsID που χρησιμεύει στην σύνθεση του ονόματος της μεταβλητής και τοποθετεί αυτή τη νέα μεταβλητή στο κοινόχρηστο λεξικό tempVars που περιέχει όλες τις προσωρινές μεταβλητές που δημιουργούνται κατά την ανάλυση του πηγαίου κώδικα.

   * Σημείωση : Κατά την συγγραφή αυτού του report πρόσεξα ότι θα μπορούσε να ήταν απλά μια λίστα αντί για λεξικό. Δεν επηρεάζει την λειτουργία του compiler ωστόσο θα ήταν πιο “σωστό”.

4. **def** emptyList():

   Δημιουργεί και επιστρέφει μία κενή λίστα.

5. **def** makeList(label):

   Δημιουργεί και επιστρέφει μια λίστα με μοναδικό στοιχείο την ετικέτα label.

6. **def** mergeList(list1, list2):

   Συνενώνει τις δύο λίστες (list1, list2) και επιστρέφει αυτήν την ένωση.

7. **def** backpatch(list1, newOp3):

   Η backpatch αναζητεί στην κοινόχρηστη λίστα quadList όλες τις τετράδες των οποίων οι ετικέτες βρίσκονται στην λίστα list1. Σε κάθε μία τετράδα που βρίσκει, ενημερώνει το πεδίο op3 με την τιμή newOp3.

Η παραγωγή επιτυγχάνεται καλώντας αυτές τις 7 βοηθητικές συναρτήσεις ενδιάμεσα απ’ τους συντακτικούς κανόνες. Ιδιαίτερο πρόβλημα αντιμετωπίσαμε στην forcaseStat(), repeatStat() και στην exitStat().

Συγκεκριμένα, στην forcaseStat() δημιουργείται μία προσωρινή μεταβλητή που λειτουργεί σαν μεταβλητή-σημαία. Μόλις Όποτε κάποιο when της forcase αποτιμάται σε true, και επομένως εκτελούνται όλα τα statements μετά το when, η μεταβλητή-σημαία παίρνει την τιμή 1. Στο τέλος της forcase, τοποθετήσαμε έναν έλεγχο αυτής της σημαίας. Αν είναι ίση με 0, σημαίνει ότι δεν εκτελέστηκε κανένα when οπότε πρέπει ο έλεγχος να περάσει έξω απ’ τον forcase. Αν είναι ίση με 1, τότε ο έλεγχος επιστρέφεται πίσω, στην αρχή της forcase.

Στη repeatStat() και στην exitStat() το πρόβλημα ήταν στο ότι έπρεπε κάθε exit να αντιστοιχίζεται στο “δικό του” repeat nest και όχι απαραίτητα στο τελευταιό που είδαμε. Το πρόβλημα αυτό λύθηκε διατηρώντας τον αριθμό των φωλιασμένων repeats που συναντήθηκαν μέχρι μια χρονική στιγμή και αποθηκεύοντας όλα τα exit quads σε ένα λεξικό. Όταν βρεθεί το endrepeat, οι τετράδες που έχουν αποθηκευτεί στο λεξικό γίνονται backpatch ώστε να δείχουν έξω απ’ το σωστό repeat.

Μετάφραση σε C

* Η οποιαδήποτε μετάφραση δεν θα είναι επιτυχής σε περίπτωση που το πηγαίο αρχείο περιέχει συναρτήσεις ή διαδικασίες πέραν της main.

Η μετάφραση του ενδιάμεσου κώδικα στον αντίστοιχο κώδικα C ξεκινάει απ’ την συνάρτηση printCequivCode(). Για κάθε τετράδα που έχει παραχθεί, πλην της τετράδας με όνομα ‘begin_block’, καλείται η συνάρτηση quadToC όπου και γίνεται και η ακριβής μετάφραση σε κώδικα C ανάλογο με το όνομα της τετράδας και.

Για την εύρεση όλων των μεταβλητών, προσωρινών και μη, σε αυτή τη φάση δημιουργήσαμε τις συναρτήσεις findVars() και printVars(). Η findVars() βρίσκει όλες τις μεταβλητές και τις τοποθετεί σε μία λίστα την οποία επιστρέφει και η printVars() δέχεται αυτή τη λίστα και τυπώνει τις μεταβλητές στο .c αρχείο.

Δομές

Η δομή που χρησιμοποιήθηκε για την παραγωγή του ενδιάμεσου κώδικα είναι η κλάση Quad.

```
class Quad:
    # Constructor
    # label: quad's label
    # name: quad's name
    # op1,op2,op3: quad's components
    def __init__(self, label, name, op1, op2, op3):
        self.label = label
        self.name = name
```

```
8.      self.op1 = op1
9.      self.op2 = op2
10.     self.op3 = op3
11.
12.  def __str__(self):
13.     return str(self.label) + ": " + "(" + str(self.name) + "," + str(self.op1) + \
14.           "," + str(self.op2) + "," + str(self.op3) + ")"
```

Δημιουργούνται αντικείμενα αυτής της κλάσης απ' την genQuad() και τοποθετούνται στην κοινόχρηστη λίστα quadList. Σε αυτά τα αντικείμενα τοποθετούμε την ετικέτα της τετράδας που αντιπροσωπεύουν, το όνομα/πράξη που πρόκειται να εκτελεστεί απ' αυτήν την τετράδα και τους τρεις όρους που απαιτούνται γι' αυτή την πράξη.

Πίνακας Συμβόλων

Περιγραφή

Ο πίνακας συμβόλων είναι το τελευταίο στάδιο πριν την παραγωγή του τελικού κώδικα. Σε αυτόν αναγνωρίζονται οι εμβέλειες των μεταβλητών και των συναρτήσεων, ελέγχεται ο σωστός ορισμός αυτών καθώς και των παραμέτρων. Χρησιμοποιούμε 7 συναρτήσεις για την δημιουργία του πίνακα συμβόλων.

1. `def createScope():`

   Καλείται κατά την αναγνώριση μιας καινούργιας συνάρτησης ή διαδικασίας, δηλαδή στην procorfunc(). Δημιουργεί ένα καινούργιο αντικείμενο κλάσης Scope και το προσθέτει στην κοινόχρηστη λίστα scopesList.

2. `def addFunction(id, funcorproc):`

   id: Το όνομα της συνάρτησης που πρόκειται να δημιουργηθεί

   funcorproc: 0 για διαδικασία, 1 για συνάρτηση

   Προσθέτει μία νέα συνάρτηση ή διαδικασία ως "παιδί" της γονικής εμβέλειας. Δημιουργεί ένα νέο αντικείμενο της κλάσης Function και το προσθέτει στο γονικό Scope. Αρκεί, βέβαια, να μην έχει οριστεί ξανά αυτή η συνάρτηση. Καλείται απ' την procorfunc().

3. `def addVariable(id):`

   id: Το όνομα της μεταβλητής που πρόκειται να δημιουργηθεί

   Εντελώς αντίστοιχα με την addFunction(), δημιουργούμε ένα νέο αντικείμενο κλάσης Variable και το προσθέτουμε ως "παιδί" στην γονική συνάρτηση/εμβέλεια. Ελέγχεται και τυπώνεται

μήνυμα λάθους σε περίπτωση που η μεταβλητή έχει δηλωθεί δύο φορές ή η μεταβλητή έχει δηλωθεί ως παράμετρος στην ίδια εμβέλεια. Καλείται κατά την διάρκεια ορισμού των μεταβλητών στην varlist().

4. **def** addParameter(id, parMode):

   id: Το όνομα της παραμέτρου που πρόκειται να δημιουργηθεί

   parMode: 'cv' για 'in', 'cr' για 'inout'

   Όταν συναντάμε μια καινούργια τυπική παράμετρο στην formalparitem() καλούμε την addParameter(). Δημιουργείται ένα καινούργιο αντικείμενο κλάσης Parameter και το τοποθετεί ως new Entity στο κάτω απ' την γονική συνάρτηση. Παράλληλα, αυξάνεται το frame length της γονικής συνάρτησης κατα 4. Αν η παράμετρος έχει ήδη δηλωθεί στο ίδιο βάθιος φωλιάσματος τότε βγαίνει αντίστοιχο μήνυμα λάθους.

5. **def** addArgument(funcId, parMode):

   id: Το όνομα της συνάρτησης στην οποία πρόκειται να προστεθεί το νέο όρισμα

   parMode: 'cv' για 'in', 'cr' για 'inout'

   Δημιουργείται ένα νέο αντικείμενο κλάσης Argument τύπου parMode ('cv' για παράμετρο με τιμή, 'cr' για παράμετρο με αναφαρά), βρίσκει συνάρτηση με όνομα funcId και το τοποθετεί ως όρισμα της συνάρτησης αυτής. Αν η συνάρτηση δεν έχει οριστεί τότε τυπώνεται ανάλογο μήνυμα.

6. **def** setFunctionStartQuad(id):

   id: Το όνομα της συνάρτησης της οποίας θα ενημερωθεί το πεδίο startQuad

   Ενημερώνει το πεδίο startQuad του αντικειμένου κλάσης Function με όνομα 'id' με το label του πρώτου Quad γι' αυτή την συνάρτηση. Σε περίπτωση που δεν υπάρχει συνάρτηση με όνομα 'id' τότε ο χρήστης ενημερώνεται με αντίστοιχο μήνυμα. Καλείται στην αρχή της block(), ακριβώς πριν δημιουργηθεί η τετράδα με όνομα 'begin_block'.

7. **def** setFunctionFrameLength(id, framelength):

   id: Το όνομα της συνάρτησης της οποίας θα ενημερωθεί το πεδίο frameLength

   framelength: Η τελική τιμή του εγγρασήματος δραστηριοποίησης

   Ενημερώνει το πεδίο frameLength του αντικειμένου κλάσης Function με όνομα 'id' με το 'framelength'. Σε περίπτωση που το 'id' είναι το όνομα του κιρίως προγράμματος τότε ενημερώνεται η κοινόχρηστη μεταβλητή 'mainframe'. Καλείται στο τέλος block() αμέσως μετά την παραγωγή της τετράδας με όνομα 'end_block'.

Δομές

Οι δομές που χρησιμοποιούνται για την παραγωγή του πίνακα συμβόλων είναι οι εξής:

- **class** Scope:

```
1.    # Constructor
2.    # nestingLevel: scope's nesting level
3.    # entities: list of scope's entities
4.    # previousScope: pointer to the immediate above Scope object
5.    # tempFrameLength: holds the frame length value of it's enclosed Scop
      e
6.    def __init__(self, nestingLevel, previousScope):
7.       self.nestingLevel = nestingLevel
8.       self.entities = list()
9.       self.previousScope = previousScope
10.      self.tempFrameLength = 12 # Default value
11.
12.   def __str__(self):  ...
13.
14.   def addEntity(self, newEntity):  ...
15.
16.   def setTempFrameLength(self):  ...
17.
18.   def getTempFrameLength(self):  ...
```

Κάθε αντικείμενο κλάσης Scope αντιπροσωπεύει και ένα βάθος φωλιάσματος. Σε αυτά αποθηκεύουμε το βάθος φωλιάσματος (nestingLevel), τις οντότητες που ανήκουν σε αυτό το βάθος φωλιάσματος (entities), το γονικό Scope αντικείμενο (previousScope) και το frame length του Scope (tempFrameLength) το οποίο αρχικοποιούμε στο 12 γιατί σε κάθε εγγράφημα δραστηριοποίησης αρχικά αποθηκεύουμε τη διεύθυνση επιστροφής, τον σύνδεσμο προσπέλασης και την επιστροφή τιμής.

Η setTempFrameLength αυξάνει το frame length του τρέχοντος Scope κατά 4. Καλείται στην newTemp() κατά τη δημιουργία μιας νέας προσωρινής μεταβλητής, στην addVariable() κατά την προσθήκη μιας νέας μεταβλητής ως Entity κάτω απ’ το τρέχον Scope και στην addParameter() κατά την προσθήκη μιας νέας παραμέτρου ως Entity κάτω απ’ το τρέχον Scope.

- **class** Entity:

```
1.    # Constructor
2.    # id: entity's identifier returned from lex
3.    # type: entity's type
4.    def __init__(self, id, typeOf):
5.       self.id = id
6.       self.typeOf = typeOf
```

```
7.
8.    def __str__(self):  ...
9.
10.   def toString(self):  ...
```

- **class** Variable(Entity): ...
- **class** Function(Entity): ...
- **class** Parameter(Entity): ...
- **class** TempVariable(Entity): ...

Οι κλάσεις Variable, Function, Parameter και TempVariable κληρονομούν την κλάση Entity και αντιπροσωπεύουν τις οντότητες που προστίθονται σε κάθε Scope αντικέιμενο. Συγκεκριμένα, αντιπροσωπεύουν μεταβλητές, συναρτήσεις/διαδικασίες, παραμέτρους και προσωρινές μεταβλητές αντίστοιχα.

- **class** Argument():

```
1.    # Constructor
2.    # parMode: cv = call by value || cr = call by reference
3.    def __init__(self, parMode):
4.        self.parMode = parMode
5.
6.    def __str__(self):
7.        return "ParMode = " + self.parMode
```

Τέλος, έχουμε την κλάση Argument που αντιπροσωπεύει τα ορίσματα μιας συνάρτησης, γι’ αυτό και αμέσως μετά την δημιουργία ενός αντικειμένου Argument προστίθεται στο πεδίο args ενός αντικειμένου κλάσης Function.

Τελικός Κώδικας

Περιγραφή

Η παραγωγή του τελικού κώδικα αφορούσε την μετατροπή του ενδιάμεσου κώδικα σε κώδικα assembly για τον επεξεργαστή MIPS. Η μετάφραση καλείται απ’ την block() κάθε φορά που τελείωνει η ανάλυση ενός βάθος φωλιάσματος EEL κώδικα. Οι παρακάτω συναρτήσεις αποτελούν την παραγωγή τελικού κώδικα του Compiler.

- **def** gnlvcode(var):

Η gnlvcode εντοπίζει μία μη τοπική μεταβλητή μέσω του σύνδεσμου προσπέλασης και μεταφέρει στον καταχωρητή $t0 την διεύθυνσή της. Παράγει: lw $t0, -4($sp)

N-1 φορές: lw $t0, -4($t0)
Τέλος: add $t0, $t0, -offset

- **def** loadvr(v, r):

Η loadvr μεταφέρει τα δεδομένα απ' το 'v' στον καταχωρητή '$tr'. Συγκεκριμένα έχουμε τις εξής περιπτώσεις :

1. Αν η 'v' είναι απλά μια αριθμητική τιμή τότε έχουμε: 1. li $tr, v
2. Αν η 'v' είναι κοινόχρηστη μεταβλητή τότε έχουμε: 1. lw $tr, -offset($s0)
3. Αν η 'v' είναι τοπική μεταβλητή, τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος ίσο με το τρέχον ή προσωρινή μεταβλητή τότε η τιμή υπάρχει στην τρέχουσα στοίβα: 1. lw $tr, -offset($sp)
4. Αν η 'v' είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος ίσο με το τρέχον, τότε: 1. lw $t0, -offset($sp)

   2. lw $tr, ($t0)
5. Αν η 'v' είναι τοπική μεταβλητή στον πρόγονο ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος μικρότερο απ' το τρέχον, τότε: 1. gnlvcode(v)

   2. lw $tr, ($t0)
6. Αν η 'v' είναι τυπική παράμετρος που περνάει με αναφορά στον πρόγονο και έχει βάθος φωλιάσματος μικρότερο απ' το τρέχον, τότε: 1. gnlvcode(v)

   2. lw $t0, ($t0)

   3. lw $tr, ($t0)

- **def** storerv(r, v):

Η storerv μεταφέρει δεδομένα απ' τον καταχωρητή '$tr' στη μνήμη (μεταβλητή 'v'). Διακρίνουμε τις εξής περιπτώσεις :

1. Αν η 'v' είναι μια κοινόχρηστη μεταβλητή, τότε: 1. sw $tr, -offset($s0)
2. Αν η 'v' είναι τοπική μεταβλητή, τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος ίσο με το τρέχον ή προσωρινή μεταβλητή, τότε: 1. sw $tr, -offset($sp)
3. Αν η 'v' είναι τυπική παράμετρος που περνάει με αναφορά και βάθος φωλιάσματος ίσο με το τρέχον, τότε: 1. lw $t0, -offset($sp)

   2. sw $tr, ($t0)

4. Αν η ‘v’ είναι τοπική μεταβλητή στον πρόγονο ή τυπική παράμετρος που περνάει με τιμή και βάθος φωλιάσματος μικρότερο απ’ το τρέχον, τότε: 1. gnlvcode(v)
   2. sw $tr, ($t0)
5. Αν η ‘v’ είναι τυπική παράμετρος που περνάει με αναφορά στον πρόγονο και έχει βάθος φωλιάσματος μικρότερο απ’ το τρέχον, τότε: 1. gnlvcode(v)
   2. lw $t0, ($t0)
   3. lw $tr, ($t0)

- `def quadToAsm(quad, name):`

Τέλος, έχουμε την quadToAsm, η οποία καλείται στο τέλος της block(). Αναλύει τις τετράδες που φτάνουν και ανάλογα με το τύπο πράξης τους, δηλαδή το πεδίο name του αντικειμένου Quad, τυπώνονται οι ανάλογες εντολές σε assembly στο αρχείο .asm.

Για τον έλεγχο των παραμέτρων ότι είναι σωστός ο τύπος περάσματός τους και το πλήθος τους είναι σύμφωνο με το πλήθος των ορισμάτων της συνάρτησης που πρόκειται να περαστούν χρησιμοποιούμε:

1. Τη κοινοχρηστή μεταβλητή parametersEncountered που μετράει πόσες παραμέτρους έχουμε έχουμε συναντήσει εξαιρώντας τις τετράδες ‘par’ με πεδίο op2 = ‘ret’.
2. Το κοινόχρηστο λεξικό parametersDict όπου περνάμε τις τετράδες ‘par’ εξαιρώντας αυτές με πεδίο op2 = ‘ret’.

Κάθε φορά που έρχεται τετράδα με πράξη ‘call’ καλείται η συνάρτηση parameterCheck για να διαπιστωθεί ότι ο αριθμός παραμέτρων και ο τύπος της κάθε μίας συμφωνεί με τον αριθμό και τον τύπο των ορισμάτων της κληθείσας συνάρτησης.

Επεξήγηση global μεταβλητών

1. **token:** Η μεταβλητή την οποία “γεμίζει” ο λεκτικός αναλυτής όταν το ζητήσει ο συντακτικός αναλυτής.
2. **file_size:** Κρατάει το μέγεθος του αρχείου .eel.
3. **totalLineNo:** Ο αριθμός της γραμμής που διαβάζει ο λεκτικός αναλυτής.
4. **totalCharNo:** Ο αριθμός του χαρακτήρα που διαβάζει ο λεκτικός αναλυτής.
5. **quadList:** Η λίστα με όλες τις τετράδες που παράγονται για τον ενδιάμεσο κώδικα.
6. **labelID:** Ο μετρητής των ετικετών για τις τετράδες.
7. **tempVars:** Λεξικό που κρατάει όλες τις προσωρινές μεταβλητές.
8. **tempVarsID:** Ο μετρητής που χρησιμοποιείται για τη σύνθεση ονομάτων προσωρινών μεταβλητών

9. **main_name:** Το όνομα του κυρίως προγράμματος στο πηγαίο αρχείο.
10. **haltLabel:** Η ετικέτα της τετράδας με (halt,_,_,_) που σημαίνει το τέλος του προγράμματος EEL.
11. **repeatNests:** Ο αριθμός των φωλιασμένων repeat statements.
12. **exitStats:** Λεξικό που κρατάει ως κλειδιά τις τετράδες (jump,_,_,_), που προκύπτουν από exit statements, και ως τιμές τον αριθμό του φωλιασμένου repeat statement.
13. **exitQuads:** Οι τετράδες που πρόκειται να γίνουν backpatched ώστε να δείχνουν έξω απ' το σωστό repeat statement.
14. **scopesList:** Κρατάει όλα τα αντικείμενα Scopes που παράγονται.
15. **returnDict:** Λεξικό που κρατάει ζεύγη {βάθος_φωλιασματος : True or False} και χρησιμοποιείται στον έλεγχο για return statements έξω από μία συνάρτηση ή την παράλειψη αυτού.
16. **mainframe:** Το μέγεθος του εγγρασήματος δραστηριοποίησης του κυρίως προγράμματος.
17. **symtablefile:** Το όνομα του αρχείου για τον πίνακα συμβόλων.
18. **asmfile:** Το όνομα του αρχείου για τον τελικό κώδικα.
19. **parametersEncounteted:** Ο αριθμός των παραμέτρων που είδαμε κατά την κλήση μιας συνάρτησης ή διαδικασίας.
20. **parametersDict:** Κρατάει τις τετράδες των παραμέτρων που είδαμε κατά την κλήση μιας συνάρτησης ή διαδικασίας.
21. **promptDict:** Κρατάει τα ονόματα των μεταβλητών που είτε θα τυπωθούν είτε θα ζητήσουν είσοδο απ' τον χρήστη. Αγνοεί προσωρινές μεταβλητές.

Interface

Για το interface χρησιμοποιήθηκε η argparse της Python. Δημιουργήθηκε η δυνατότητα για μετονομασία όλων των αρχείων που πρόκειται να δημιουργηθούν. Διατίθεται βοήθεια με το όρισμα -h ή –help και πληροφορίες σχετικά με την έκδοση και τη χρήση του Compiler με το όρισμα -i ή –info.