Καλούδης Σπυρίδων 2447
Χουλιάρας Ιωάννης. 2631
Παρασκευή 31 Μαΐου 2019

Πανεπιστήμιο Ιωαννίνων

Μεταφραστές (ΜΥΥ802)

Διδάσκων: Γεώργιος Μανής

Ακαδημαϊκό έτος 2018-2019

Περιεχόμενα

Εισαγωγή

Η γλώσσα προγραμματισμού Starlet

Η γλώσσα προγραμματισμού Starlet, παρότι είναι μία μικρή γλώσσα, εξυπηρετεί απόλυτα τις διδακτικές ανάγκες του μαθήματος των μεταφραστών. Παρέχει αρκετές ενδιαφέρουσες δυνατότητες τις οποίες έπρεπε να διαχειριστούμε ανάλογα για τη σωστή ανάπτυξη του compiler.

Το αλφάβητο της Starlet

Το βασικό αλφάβητο της Starlet αποτελείται από:

- τα μικρά και κεφαλαία γράμματα της λατινικής αλφαβήτου («A»,…,«Z» και «a»,…,«z»),
- τα αριθμητικά ψηφία («0»,…,«9»),
- τα σύμβολα των αριθμητικών πράξεων («+», «-», «*», «/»),
- τους τελεστές συσχέτισης « < », « > », « = », « <= », « >= », « <> »,
- το σύμβολο ανάθεσης «:=»,
- τους διαχωριστές (« ; », « , », « : »)
- καθώς και τα σύμβολα ομαδοποίησης («(»,«)»,«[»,«]»)
- και διαχωρισμού σχολίων («/*»,«*/»,«//»).

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

Δεσμευμένες λέξεις

Οι δεσμευμένες λέξεις της γλώσσας Starlet είναι οι εξής:

- program, endprogram,
- Declare
- If then else
- While, endwhile, dowhile, enddowhile
- Loop, endloop, exit
- forcase, endforcase, incase, endimcase, when default, enddefault
- function, endfunction, return, in, inout, inandout
- And, or, not
- input, print

Σε κάθε μία από αυτές θα αναφερθούμε εκτενώς στη συνέχεια.

Επιτρεπόμενοι τύποι και δηλώσεις μεταβλητών

Η Starlet υποστηρίζει μόνο ακέραιος αριθμός σαν δεδομένα. Η τιμή τους μπορεί να είναι από -32.767 έως και 32.767. Η δήλωση οποιαδήποτε μεταβλητής γίνεται με την δεσμευμένη λέξη declarations ακολουθούμενη από το όνομα της μεταβλητής. Αν πρόκειται για πολλαπλή δήλωση τα ονόματα χωρίζονται μεταξύ τους με κόμμα. Το τέλος οποιασδήποτε δήλωσης αναγνωρίζεται με το σύμβολο ;

Τελεστές και εκφράσεις

Η προτεραιότητα των τελεστών από τη μεγαλύτερη στη μικρότερη είναι:

1. «not»
2. «*», «/»
3. « + »,« - » (μοναδιαίοι)
4. « + »,« - » (δυαδικοί)
5. « = »,« < »,« > »,« <> »,« <= »,« >= »
6. Λογικό « and »
7. Λογικό « or »

Δομές της Starlet

Οι δομές της Starlet είναι οι εξής:

- Εκχώρηση:

  id := expression

- If:

  if (conditon) then
  statements
  [else
  statements]
  endif

- While:

  while(condition)
  statements
  endwhile

- Dowhile-Enddowhile:

  dowhile
  statements
  enddowhile(condtion)

- Loop:

  loop
  statements
  endloop

- Forcase:

  forcase

  (when (condition): statements)*

  default: statements enddefault

  endforcase

- Incase:

  incase

  (when (condition): statements)*

  endincase

- Return:

  return expression

- Print:

  print expresion

- Input:

  input id

Θα αναπτύξουμε το καθένα ξεχωριστά στη συνέχεια.

Functions

Η Starlet υποστηρίζει την χρήση συναρτήσεων, οι οποίες γράφονται αμέσως μετά το declare (αν αυτό υπάρχει). Η βασική μορφή ενός function είναι η:

```
function id (formal_pars)
        declarations
        subprograms
        statements
endfunction
```

Όπου formal_pars είναι η λίστα τυπικών παραμέτρων

Μετάδοση παραμέτρων

Η Starlet υποστηρίζει τρεις τρόπους μετάδοσης παραμέτρων:

- Με σταθερή τιμή (in)
- Με αναφορά (inout)
- Με αντιγραφή (inandout)

Ανάπτυξη Μεταγλωττιστή

Η ανάπτυξη που θα ακολουθήσουμε θα χωρίζεται με βάση κάθε διαφορετική λεκτική μονάδα, η οποία είναι αποδεκτή από τη γλώσσα Starlet. Η κάθε λεκτική μονάδα θα χωρίζεται σε τέσσερα διαφορετικά στάδια - κεφάλαια, lex (λεκτικός αναλυτής), syntax(συντακτικός αναλυτής), intermediate code(ενδιάμεσος κώδικας), final code(τελικός κώδικας).

Ξεκινάμε τον μεταγλωττιστή μας ανοίγοντας το αρχείο της στάρλετ που θέλουμε να μεταγλωττίσουμε και το αποθηκεύουμε σε μία μεταβλητή. Κρατάμε το μέγεθος της μεταβλητής. Αρχικοποιούμε τις μεταβλητές ***token*** η οποία κρατάει την λέξη την οποία πρέπει να αναγνωρίσουμε, ***numberofchar*** η οποία κρατάει σε ποιο χαρακτήρα έχουμε σταματήσει, ***tokentk*** η οποία κρατάει το είδος του token που αναγνωρίσαμε, ***linecounter*** η οποία κρατάει τη γραμμή στην οποία βρισκόμαστε (για εμφάνιση τυχόν λάθους), ***tokenList*** η οποία είναι μία λίστα που κρατάει όλα tokens και τα αντίστοιχα tokentk, ***counter*** η οποία ενημερώνεται κάθε φορά που έχουμε προσπελάσει μία λεκτική μονάδα, ***programName*** η οποία κρατάει το όνομα του προγράμματος, ***currentScope*** η οποία κρατάει το βάθος φωλιάσματος στο οποίο βρισκόμαστε, ***entities*** η οποία είναι λίστα που περιέχει λίστες οι οποίες αντιστοιχούν στα scopes, ***quads*** η οποία είναι λίστα στην οποία γράφονται οι τετράδες του ενδιάμεσου κώδικα, ***inLoop*** η οποία παίρνει τιμή ανάλογα αν βρισκόμαστε μέσα σε ένα loop ή όχι, ***exitQuad*** η οποία παίρνει τιμή ανάλογα με το αν βρήκαμε exit, ***inFunction*** που στην ουσία μας ενημερώνει αν είμαστε μέσα σε συνάρτηση, ***foundReturn*** η οποία μας δείχνει πόσα return έχουμε βρει πριν κλείσει κάποια συνάρτηση, ***loopCount*** η οποία αυξάνεται αν είμαστε μέσα σε loop, ***exitPointer*** η οποία κρατάει το Quad στο οποίο πρέπει να κάνουμε jump όταν βρούμε exit, ***tempCount*** η οποία μετράει πόσες προσωρινές μεταβλητές έχουμε χρησιμοποιήσει, ***finalCode*** η οποία είναι μία λίστα στην οποία αποθηκεύονται οι εντολές τελικού κώδικα, ***quadCount*** η οποία δείχνει ποια quads έχουν μεταγλωττιστεί σε τελικό κώδικα και τέλος το ***functionlabels*** το οποίο είναι ένα dictionary. Δημιουργούμε τη βοηθητική συνάρτηση error(message), οπου message το μήνυμα λάθος το οποίο θέλουμε να εμφανίσουμε ακολουθούμενο από τη γραμμή στην οποία υπήρχε το λάθος για να βοηθήσουμε τον χρήστη στην καλύτερη αντιμετώπιση του. Αφού εμφανιστεί το μήνυμα ο μεταγλωττιστής τερματίζει.

Ο λεκτικός αναλυτής αποτελείται από μία βοηθητική συνάρτηση ***lexer()*** η οποία ενημερώνει την μεταβλητή counter και καλεί την βασική συνάρτηση ***lex()***. Δουλειά της συνάρτησης lex( ) είναι να αναγνωρίσει το είδος της λεκτικής μονάδας. Πιο συγκεκριμένα διαβάζει χαρακτήρες μέχρι να βρει κάποιο λευκό χαρακτήρα (κενό, tab, newline). Έπειτα το συγκρίνει με τη χρήση πολλαπλών if με τις δεσμευμένες λέξεις της Starlet και αναγνωρίζει αν πρόκειται για δεσμευμένη λέξη ή μεταβλητή ή αριθμό. Αφού το αναγνωρίσει με επιτυχία το προσθέτει στον πίνακα tokenList με το αντίστοιχο αναγνωριστικό του (tokentk). Αν βρει αρχή σχολίου γραμμής (δηλαδή "//") τότε προσπερνάει ολόκληρη τη γραμμή. Αν βρει αρχή μεγάλου σχολίου "/* " τότε προσπερνάει οτιδήποτε βρει μέχρι να βρει το αντίστοιχο κλείσιμο " */ ". Αν υπάρχει εμφωλευμένο σχόλιο, επιστρέφει μήνυμα λάθους. Αν βρει λεκτική μονάδα που δεν αναγνωρίζεται, τότε εμφανίζει μήνυμα λάθους με το σύμβολο που προκάλεσε το error.

Ο συντακτικός αναλυτής αποτελείται από πληθώρα συναρτήσεων οι οποίες ελέγχουν κομμάτι-κομμάτι τη δομή του προγράμματος ώστε να σιγουρευτούν ότι είναι έγκυρο. Αν λείπει κάτι η υπάρχει κάτι περιττό το πρόγραμμα δεν μεταγλωττίζεται και ο χρήστης λαμβάνει σχετικό μήνυμα λάθους το οποίο τον βοηθάει στο να διορθώσει συντακτικά το πρόγραμμα. Το μήνυμα αυτό έρχεται μέσα από τη συνάρτηση ***error()***.

Το κομμάτι του ενδιάμεσου κώδικα συνδέεται άμεσα με το κομμάτι του συντακτικού αναλυτή καθώς οι εντολές υλοποίησής του περιέχονται μέσα σε αυτόν. Για τη σωστή υλοποίηση έχουμε δημιουργήσει τις εξής βοηθητικές συναρτήσεις:

- ***newTemp()*** η οποία χρησιμοποιεί την global μεταβλητή tempCount και κάθε φορά μας επιστρέφει το όνομα μιας καινούργιας προσωρινής μεταβλητής (T_x). Επίσης προσθέτει στο scope όπου δουλεύουμε την T_x σαν μεταβλητή του scope.
- ***genquad(op,x,y,z)*** η οποία γράφει στην λίστα quads μία ετικέτα της μορφής [op,x,y,z].
- ***nextquad()*** η οποία επιστρέφει τον αριθμό της επόμενης τετράδας που θα παραχθεί.
- ***emptylist()*** η οποία επιστρέφει μία κενή λίστα.
- ***makelist(x)*** η οποία επιστρέφει μία λίστα με μοναδικό στοιχείο το x.
- ***mergelist(x,y)*** η οποία συγχωνεύει δύο λίστες x,y στη θέση της x.

- ***backpatch(x,z)*** η οποία συμπληρώνει το τελευταίο μέρος της τετράδας x με z.
- ***printQuads()*** η οποία εμφανίζει όλες στις τετράδες στην οθόνη μας.

Όλες αυτές τις συναρτήσεις τις βάζουμε στα κατάλληλα κομμάτια του syntax ώστε να παραχθούν σωστά οι τετράδες του ενδιάμεσου κώδικα. Πιο αναλυτικά στη συνέχεια.

Το κομμάτι του πίνακα συμβόλων υλοποιείται επίσης μέσα στον syntax και χρησιμοποιεί επίσης τις βοηθητικές συναρτήσεις:

- ***changeScope()*** η οποία χρησιμοποιείται οταν βρίσκουμε καινούργια συνάρτηση αυξάνει το currentScope κατά ένα και γράφει έναν καινούργιο πίνακα στον πίνακα entities ο οποίος αντιστοιχεί στην καινούργια συνάρτηση.
- ***closeScope()*** η οποία κάνει χρήση της checkNames() εμφανίζει τα entities του τωρινού scope υπολογίζει το offset της συνάρτησης της οποίας εχουμε τελειώσει τη μεταγλώττιση με τη βοήθεια της findCurrentOffset(), αφαιρεί τη συνάρτηση από τα entities και μειώνει το currentScope κατά ένα.
- ***addFunc(func)*** η οποία γράφει στο currentScope του entities την δήλωση μιας συνάρτησης με όνομα func ([func,0,"func",[]]), στο τελευταίο μέρος μέσα στον πίνακα θα μπει ο τρόπος περάσματος των μεταβλητών του ορίσματος της συνάρτησης.
- ***addVar(var)*** η οποία προσθέτεις τα entities μία μεταβλητή με το όνομα var και της δίνει το σωστό offset κάνοντας χρήση της findCurrentOffset().
- ***findCurrentOffset()*** η οποία ψάχνει στο τελευταίο scope που έχει ανοίξει βρίσκει την τελευταία μεταβλητή παίρνει το offset της και το επιστρέφει αυξημένο κατά τέσσερα. Αν είναι η πρώτη μεταβλητή του scope παίρνει αυτόματα το offset δώδεκα.
- ***fillFuncVariableType(type)*** η οποία αφού έχει ανοίξει καινούργιο scope πηγαίνει στο προηγούμενο στην τελευταία θέση (εκεί που φυλάσσεται η συνάρτηση) και στον πίνακα στον οποίο κρατούνται οι τύποι των μεταβλητών προσθέτει ένα καινούργιο τύπο αμέσως μετά τον προηγούμενο.
- ***fillFuncVariables(id)*** η οποία είναι απλά μία βοηθητική συνάρτηση που κάνει χρήση της addVar(id) και στην ουσία προσθέτει μία μεταβλητή στο entities.
- ***setFuncOffset(num)*** στην οποία δίνουμε σαν όρισμα το offset που θέλουμε να συμπληρώσουμε και η συνάρτηση πηγαίνει στην τελευταία θέση του προηγούμενου scope (εκεί που φυλάσσεται η συνάρτηση) και συμπληρώνει το σωστό offset.

- ***checkNames( )*** η οποία διατρέχει όλο τον πίνακα συμβόλων και κρατάει κάθε όνομα σε ένα set, αν βρει ίδιο όνομα πάνω από μία φορά τότε εμφανίζει μήνυμα λάθους γιατί δεν μπορούμε να έχουμε το ίδιο όνομα σε δύο διαφορετικές μεταβλητές ή συναρτήσεις.
- ***checkVar(id)*** η οποία δέχεται σαν όρισμα το όνομα μιας μεταβλητής και ψάχνει σε όλα τα entities να δει αν υπάρχει, αν υπάρχει και είναι μεταβλητή επιστρέφει ένα αλλιώς αν υπάρχει και είναι συνάρτηση τυπώνει μην είμαι λάθος Επίσης αν δεν βρεθεί τυπώνει διαφορετικό μήνυμα λάθους ενημερώνοντας το χρήστη ότι η μεταβλητή δεν έχει δηλωθεί.
- ***checkFunc(id)*** η οποία κάνει την αντιστροφή δουλειά από την checkVar(id), δηλαδή ψάχνει να δει αν μία συνάρτηση έχει δηλωθεί.
- ***checkParameterType(id,num,type)*** η οποία δέχεται σαν παραμέτρους το όνομα μίας συνάρτησης, τον αύξων αριθμό που αντιπροσωπεύει ποια παράμετρο της συνάρτησης θέλουμε να ελέγξουμε και τον τύπο της μεταβλητής που ελέγχουμε. Έπειτα κάνοντας αναζήτηση σε όλα τα entities βρίσκει τη δήλωση της συνάρτησης και ελέγχει αν ο τύπος στη σωστή θέση της δήλωσης αντιστοιχεί στην μεταβλητή την οποία περνάμε σαν παράμετρο κατά την κλήση.

Το κομμάτι του τελικού κώδικα παίρνει τα quads του ενδιάμεσου και χρησιμοποιώντας τις παρακάτω συναρτήσεις το μετατρέπει σε κώδικα assembly.

- ***findScopeAndOffset(id)*** η οποία δέχεται σαν όρισμα το όνομα μιας μεταβλητής ψάχνει αντίστροφα τον πίνακα των entities και την αντιστοιχίζει με την πρώτη μεταβλητή που θα βρει με το ίδιο όνομα. Αφού την εντοπίσει επιστρέφει το scope στο οποίο αυτή βρίσκεται καθώς και το offset της, αν από την άλλη δεν την εντοπίσει επιστρέφει None.
- ***gnvlcode(id)*** η οποία δέχεται σαν όρισμα το όνομα μιας μεταβλητής και κάνοντας χρήση της findScopeAndOffset(id) γράφει στη λίστα finalCode με τον τελικό κώδικα τις αντίστοιχες εντολές assembly που χρειάζονται ώστε να αποθηκευτεί η τιμή της μεταβλητής στον καταχωρητή $t0.
- ***loadvr(v,r)*** η οποία δέχεται σαν όρισμα μία μεταβλητή και έναν αριθμό κάνοντας επίσης χρήση της συνάρτησης findScopeAndOffset(v) και με βάση το scope στο οποίο βρίσκεται η μεταβλητή προσθέτει στη λίστα finalCode τις αντίστοιχες εντολές assembly ώστε η τιμή της μεταβλητής v να περαστεί στον καταχωρητή $tr.

- ***storerv(r,v)*** η οποία δέχεται σαν όρισμα έναν αριθμό και μία μεταβλητή και κάνοντας χρήση της findScopeAndOffset(v) και με βάση το scope στο οποίο βρίσκεται η μεταβλητή θέτει την τιμή της ίση με την τιμή του καταχωρητή $tr.
- ***matchOperator(oper)*** η οποία δέχεται σαν όρισμα ένα σύμβολο και επιστρέφει το αντίστοιχο στη γλώσσα assembly.
- ***matchRelop(relop)*** η οποία δέχεται ένα σύμβολο και επιστρέφει το αντίστοιχο στη γλώσσα assembly.
- ***generateRefParameter(parameter, number)*** η οποία δέχεται σαν όρισμα μία παράμετρο και τον αύξων αριθμό της παραμέτρου, καλεί την συνάρτηση findScopeAndOffset(parameter) και σύμφωνα με το scope γράφει τις αντίστοιχες εντολές που πρέπει να προστεθούν στη λίστα finalCode.
- ***finalCodeGenerator( )*** η οποία διατρέχοντας τα σωστά quads αντιστοιχίζει τις εντολές ενδιάμεσου κώδικα με τις αντίστοιχες εντολές assembly, τις οποίες τοποθετεί στη λίστα finalCode.

<program>

Lex

Αν η λέξη που έχει περάσει στον lex( ) είναι η δεσμευμένη λέξη program τότε το tokentk παίρνει την τιμή programtk και στην λίστα tokenList γίνεται προσθήκη του tokentk και του token.

Syntax

Κάθε πρόγραμμα στη Starlet ξεκινάει με τη συνάρτηση program η οποία είναι κομμάτι του syntax ελέγχει αν το tokentk αντιστοιχεί στο programtk και αν είναι αληθές καλεί τον lex για να πάρει την επόμενη λέξη η οποία πρέπει να είναι όνομα μεταβλητής (id) δηλαδή το αναγνωριστικό της να είναι idtk. Αν το πρόγραμμα δεν ξεκινάει με την λέξη program ή αν η λέξη που ακολουθεί δεν είναι όνομα μεταβλητής τότε εμφανίζεται στο χρήστη σχετικό μήνυμα λάθους. Έπειτα καλεί τη συνάρτηση block(name) στην οποία δίνεται όρισμα το όνομα του προγράμματος. Τέλος η συνάρτηση program είναι υπεύθυνη για να ελέγξει αν το πρόγραμμα τελειώνει σωστά με την δεσμευμένη λέξη endprogram η οποία έχει αναγνωριστικό endprogramtk. Αν υπάρχει κώδικας γραμμένος κάτω από το endprogram τότε εμφανίζει μήνυμα λάθους.

Intermediate code & Symbol table

Το κομμάτι της γραμματικής <program> δεν παράγει αυτό καθαυτό κανένα quad για να χρησιμοποιηθεί στον ενδιάμεσο κώδικα. Ωστόσο ακριβώς πριν τερματίσει το πρόγραμμα καλεί την συνάρτηση closeScope( ) η οποία ελέγχει τα ονόματα των μεταβλητών και εμφανίζει στην οθόνη το scope της main, το οποίο και διαγράφει.

Final code

Ακριβώς πριν κλείσει το scope παράγεται ο τελικός κώδικας που αντιστοιχεί στον κώδικα ο οποίος είναι γραμμένος στην main. Καλείται η συνάρτηση finalCodeGenerator( ) η οποία μεταφράζει τις τετράδες του ενδιάμεσου Κώδικα που έχουν παραχθεί για την main σε τετράδες κώδικα assembly. Έπειτα γράφει αυτές τις τετράδες στο αρχείο στο οποίο γράφεται όλος ο τελικός κώδικας με τη βοήθεια της συνάρτησης printFinalCodeInFile( ).

<block>

Lex

Το κομμάτι του lex όσον αφορά τη συνάρτηση block(name) θα αναπτυχθεί στις παρακάτω συναρτήσεις τις οποίες και καλεί η block(name).

Syntax

Η συνάρτηση block(name) καλεί με τη σειρά τις συναρτήσεις declarations( ), subprograms( ), statements( ).

Intermediate code & Symbol table

Όταν καλείται η συνάρτηση block(name) γράφεται στην λίστα quads μία τετράδα ("begin_block",name,"_","_"), όπου name το όνομα του προγράμματος η της συνάρτησης στην οποία περιέχεται αυτό το block. Αφού κληθούν όλες οι παραπάνω συναρτήσεις γράφεται η τετράδα ("end_block",name,"_","_") που σηματοδοτεί το τέλος του block. Επίσης αν πρόκειται για το τέλος του προγράμματος τότε παράγεται και η τετράδα ("halt","_","_","_"). Η συνάρτηση block δεν πειράζει καθόλου τον πίνακα συμβόλων.

Final code

Δεν παράγεται τελικός κώδικας μέσα από το <block>.

<declarations>

Lex

Αν η λέξη που έχει περάσει στον lex( ) είναι η δεσμευμένη λέξη declare τότε το tokentk παίρνει την τιμή declaretk και στην λίστα tokenList γίνεται προσθήκη του tokentk και του token.

Syntax

Στην σύνταξη αντιστοιχεί η συνάρτηση declarations( ) η οποία όσο δέχεται το αναγνωριστικό declaretk καλεί τον lex( ) ώστε να πάει στην επόμενη λέξη και έπειτα καλεί την συνάρτηση varlist( ). Υπάρχει το while γιατί μπορούμε να έχουμε πολλαπλά declare στην αρχή οποιουδήποτε block.

Intermediate code & Symbol table

Δεν παράγεται ενδιάμεσος κώδικας μέσω της declarations( ) ούτε γράφει κάτι στον πίνακα συμβόλων.

Final code

Δεν παράγεται ούτε τελικός κώδικας μέσα από το <declarations>.

<subprograms>

Lex

Αν η λέξη που έχει περάσει στον lex( ) είναι η δεσμευμένη λέξη function τότε το tokentk παίρνει την τιμή functiontk και στην λίστα tokenList γίνεται προσθήκη του tokentk και του token.

Syntax

Στη συνάρτηση subprograms( ) υπάρχει μία while η οποία τρέχει οσο ο lex μας έχει δώσει σαν αναγνωριστικό το functiontk. Μετά από αυτό το αναγνωριστικό ξανά καλείται ο lex και η συνάρτηση subprogram( ).

Intermediate code & Symbol table

Η συνάρτηση subprograms( ) δεν παράγει ενδιάμεσο κώδικα και δεν γράφει κάτι στον πίνακα συμβόλων. Ωστόσο ενημερώνει την μεταβλητή inFunction την οποία χρησιμοποιούμε για να βρούμε αν γίνεται return σε κάθε συνάρτηση και να βεβαιωθούμε ότι δεν υπάρχει return εξω απο συνάρτηση.

Final code

Δεν παράγεται τελικός κώδικας μέσα από το <subprograms>.

<statements>

Lex

Το <statements> δεν συνδέεται απευθείας με τον lex.

Syntax

Η συνάρτηση statements( ) καλεί την statement η οποία κάνει και τη δουλειά. Επίσης επειδή μπορούμε να έχουμε πολλαπλά statement( ) υπάρχει μία while στην οποία θα μπει αν φορτωμένο από τον lex( ) έρθει το αναγνωριστικό semicolontk όπου και θα κάνει ξανά lex( ) και θα καλέσει ξανά τη statement( ).

Intermediate code & Symbol table

Δεν παράγεται ενδιάμεσος κώδικας μέσω της statements( ) ούτε γράφει κάτι στον πίνακα συμβόλων.

Final code

Δεν παράγεται τελικός κώδικας μέσα από το <statements>.

<statement>

Lex

Για το κομμάτι statement υπάρχουν πολλές διαφορετικές δεσμευμένες λέξεις οι οποίες παράγουν η καθεμία το δικό της αναγνωριστικό:

- if με αναγνωριστικό iftk.
- while με αναγνωριστικό whiletk
- dowhile με αναγνωριστικό dowhiletk
- loop με αναγνωριστικό looptk
- exit με αναγνωριστικό exittk
- forcase με αναγνωριστικό forcasetk
- incase με αναγνωριστικό incasetk
- return με αναγνωριστικό returntk
- input με αναγνωριστικό inputtk
- print με αναγνωριστικό printtk
- μεταβλητή με αναγνωριστικό idtk (χρησιμοποείται για τις εκχωρήσεις)

Syntax

Ομοίως για τον συντακτικό αναλυτή διακρίνουμε περιπτώσεις για κάθε διαφορετικό αναγνωριστικο:

- iftk καλείται η συνάρτηση lex( ) και αμέσως μετά η συνάρτηση ifstat( ).
- whiletk καλείται η συνάρτηση lex( ) και αμέσως μετά η συνάρτηση whilestat( ).
- dowhiletk καλείται η συνάρτηση lex( ) και αμέσως μετά η συνάρτηση dowhilestat( ).
- looptk καλείται η συνάρτηση lex( ) και αμέσως μετά η συνάρτηση loopstat( ).
- exittk καλείται η συνάρτηση lex( ) και αμέσως μετά η συνάρτηση exitstat( ).
- forcasetk καλείται η συνάρτηση lex( ) και αμέσως μετά η συνάρτηση forcasestat( ).
- incasetk καλείται η συνάρτηση lex( ) και αμέσως μετά η συνάρτηση incasestat( ).
- returntk καλείται η συνάρτηση lex( ) και αμέσως μετά η συνάρτηση returnstat( ).

- inputtk καλείται η συνάρτηση lex( ) και αμέσως μετά η συνάρτηση inputstat( ).
- printtk καλείται η συνάρτηση lex( ) και αμέσως μετά η συνάρτηση printstat( ).
- idtk καλείται η συνάρτηση lex( ) και αμέσως μετά η συνάρτηση assignmentstat( ).

Intermediate code & Symbol table

Δεν παράγεται ενδιάμεσος κώδικας μέσω της statement( ). Βέβαια αν μπει στην συνάρτηση statement με αναγνωριστικό idtk τότε καλεί τη βοηθητική συνάρτηση checkVar( ) η οποία ελέγχει αν η μεταβλητή είναι δηλωμένη κανονικά ως μεταβλητή. Δεν προσθέτει κάτι στον πίνακα συμβόλων.

Final code

Δεν παράγεται τελικός κώδικας μέσα από το <statement>.

<assignmentstat>

Lex

Αν η λέξη που έχει περάσει στον lex( ) είναι ο δεσμευμένος χαρακτήρας := τότε το tokentk παίρνει την τιμή assignmenttk και στην λίστα tokenList γίνεται προσθήκη του tokentk και του token.

Syntax

Ο συντακτικός αναλυτής αντιστοιχίζει το προηγούμενο token το οποίο αποθηκεύει σε μία μεταβλητή id και καλεί την συνάρτηση lex( ) για να προχωρήσει μετά το := και στην συνέχεια καλεί την expression( ). Εάν δεν βρεθεί το := τότε εμφανίζεται μήνυμα λάθους που ενημερώνει τον χρήστη στο να το προσθέσει.

Intermediate code & Symbol table

Το αποτέλεσμα της expression( ) αποθηκεύεται στην μεταβλητή Eplace. Τέλος, παράγεται η τετράδα: (“:=",Eplace,"_",id).

Final code

Δεν παράγεται τελικός κώδικας μέσα από το <assignmentstat>.

<ifstat>

Lex

Αν η λέξη που έχει περάσει στον lex( ) είναι η δεσμευμένη λέξη if τότε το tokentk παίρνει την τιμή iftk και στην λίστα tokenList γίνεται προσθήκη του tokentk και του token.

Syntax

Το επόμενο token θα πρέπει να είναι η αριστερή παρένθεση " (" . Στην συνέχεια καλείται η συνάρτηση condition( ) που πρέπει να ακολουθείται από την δεξιά παρένθεση ")". Η σύνταξη της if απαιτεί και την δεσμευμένη λέξη then για την οποία υπάρχει έλεγχος με το αναγνωριστικό της thentk. Στην συνέχεια καλούνται οι συναρτήσεις statements( ) και elsepart( ). Στο τέλος, για να είναι σωστή και ολοκληρωμένη η ifstat, χρειάζεται η δεσμευμένη λέξη endif την οποία ελέγχουμε βρίσκοντας το endiftk. Αν λείπει οποιαδήποτε από τις απαιτούμενες δεσμευμένες λέξεις, εμφανίζεται το ανάλογο μήνυμα λάθους ενημερώνοντας τον χρήστη ποια από αυτές λείπει.

Intermediate code & Symbol table

Όταν κληθεί η condition( ) το αποτέλεσμά της θα επιστραφεί σε δύο λίστες, την bTrue και την bFalse. Έπειτα μετά τον έλεγχο του then θα γίνει backpatch( ) της τετράδας που αντιστοιχεί στην λίστα bTrue με την επόμενη τετράδα. Κρατάμε επίσης την επόμενη τετράδα (δηλαδή το jump που θα δημιουργήσουμε) σε μία μεταβλητή out και παράγουμε ένα κενό jump. Μετά κάνουμε backpatch( ) την τετράδα της bFalse λίστας με την επόμενη τετράδα και έπειτα συμπληρώνουμε στο κενό jump με χρήση πάλι της backpatch( ) στο τέλος της τετράδας εξόδου την επόμενη τετράδα η οποία είναι και η τετράδα κώδικα αμέσως μετά το endif. Όσο αφορά τον πίνακα συμβόλων, δεν προστίθεται κάτι.

Final code

Δεν παράγεται τελικός κώδικας μέσα από το <ifstat>.

<elsepart>

Lex

Αν η λέξη που έχει περάσει στον lex( ) είναι η δεσμευμένη λέξη else τότε το tokentk παίρνει την τιμή elsetk και στην λίστα tokenList γίνεται προσθήκη του tokentk και του token.

Syntax

Ο συντακτικός αναλυτής εάν αναγνωριστεί η λέξη else, καλεί την συνάρτηση statements( ). Η Συνάρτηση elsepart( ) είναι προαιρετική οπότε δεν υπάρχει κάποιο μήνυμα λάθους.

Intermediate code & Symbol table

Δεν παράγεται ενδιάμεσος κώδικας ούτε προσθέτουμε κάτι στον πίνακα συμβόλων.

Final code

Δεν παράγεται τελικός κώδικας μέσα από το <elsepart>.

<whilestat>

Lex

Αν η λέξη που έχει περάσει στον lex( ) είναι η δεσμευμένη λέξη while τότε το tokentk παίρνει την τιμή whiletk και στην λίστα tokenList γίνεται προσθήκη του tokentk και του token.

Syntax

Το επόμενο token θα πρέπει να είναι η αριστερή παρένθεση " (" . Στην συνέχεια καλείται η συνάρτηση condition( ) που πρέπει να ακολουθείται από την δεξιά παρένθεση ")". Στη συνέχεια καλείται η statements( ) και τέλος θα πρέπει να υπάρχει η δεσμευμένη λέξη endwhile η οποία έχει για αναγνωριστικό το endwhiletk. Αν αυτή λείπει τότε εμφανίζεται σχετικό μήνυμα λάθους.

Intermediate code & Symbol table

Αρχικά αποθηκεύουμε την επόμενη τετράδα (η οποία αντιστοιχεί στην τετράδα συνθήκης) στην μεταβλητή condQuad. Όταν κληθεί η condition( ) το αποτέλεσμά της θα επιστραφεί σε δύο λίστες, την bTrue και την bFalse. Έπειτα όταν κλείσει η παρένθεση της συνθήκης θα γίνει backpatch( ) της τετράδας που αντιστοιχεί στην λίστα bTrue με την επόμενη τετράδα. Παράγουμε ένα jump στην condQuad και μετά κάνουμε backpatch( ) την τετράδα της bFalse λίστας με την επόμενη τετράδα. Στον πίνακα συμβόλων, δεν προστίθεται κάτι.

Final code

Δεν παράγεται τελικός κώδικας μέσα από το <whilestat>.

<dowhilestat>

Lex

Αν η λέξη που έχει περάσει στον lex( ) είναι η δεσμευμένη λέξη dowhile τότε το tokentk παίρνει την τιμή dowhiletk και στην λίστα tokenList γίνεται προσθήκη του tokentk και του token.

Syntax

Αφού αναγνωριστεί η λέξη dowhile καλείται η συνάρτηση statements( ). Αμέσως μετά θα πρέπει να ακολουθεί η δεσμευμένη λέξη enddowhile. Το επόμενο token θα πρέπει να είναι η αριστερή παρένθεση " (" . Στην συνέχεια καλείται η συνάρτηση condition( ) που πρέπει να ακολουθείται από την δεξιά παρένθεση ")". Αν οποιοδήποτε από τα απαραίτητα κομμάτια λείπει τότε εμφανίζεται σχετικό μήνυμα λάθους.

Intermediate code & Symbol table

Αρχικά αποθηκεύουμε την επόμενη τετράδα (η οποία αντιστοιχεί στην τετράδα των statements) στην μεταβλητή s. Όταν κληθεί η condition( ) το αποτέλεσμά της θα επιστραφεί σε δύο λίστες, την condTrue και την condFalse. Έπειτα θα γίνει backpatch( ) της τετράδας που αντιστοιχεί στην λίστα condTrue με την τετράδα που είναι αποθηκευμένη στο s και κάνουμε backpatch( ) την τετράδα της condFalse με την επόμενη τετράδα. Στον πίνακα συμβόλων, δεν προστίθεται κάτι.

Final code

Δεν παράγεται τελικός κώδικας μέσα από το <dowhilestat>.

<loopstat>

Lex

Αν η λέξη που έχει περάσει στον lex( ) είναι η δεσμευμένη λέξη loop τότε το tokentk παίρνει την τιμή looptk και στην λίστα tokenList γίνεται προσθήκη του tokentk και του token.

Syntax

Αφού αναγνωριστεί η λέξη loop καλείται η συνάρτηση statements( ). Αμέσως μετά θα πρέπει να ακολουθεί η δεσμευμένη λέξη endloop. Αν αυτό λείπει τότε εμφανίζεται σχετικό μήνυμα λάθους.

Intermediate code & Symbol table

Αρχικά ενημερώνουμε τις μεταβλητές inLoop (την οποία κάνουμε ένα) και loopCount (στην οποία προσθέτουμε ένα για κάθε loop (+1 για να δουλεύει για loops μέσα σε loop)). Στη συνέχεια, αποθηκεύουμε την επόμενη τετράδα (η οποία αντιστοιχεί στην τετράδα των statements) στην μεταβλητή sQuad. Δημιουργούμε ένα jump στην μεταβλητή sQuad και αν βρούμε το αναγνωριστικό endlooptk τότε κάνουμε backpatch στην τετράδα που θα έχει σημειωθεί ως exitPointer από την <exitstat> και της βάζουμε στο τέλος την επόμενη τετράδα. Έπειτα αποθηκεύουμε στην μεταβλητή exitQuad, την επόμενη τετράδα και κάνουμε την μεταβλητή inLoop ίση με το μηδέν (διότι βρήκαμε το endloop). Στον πίνακα συμβόλων, δεν προστίθεται κάτι.

Final code

Δεν παράγεται τελικός κώδικας μέσα από το <dowhilestat>.

<exitstat>

Lex

Αν η λέξη που έχει περάσει στον lex( ) είναι η δεσμευμένη λέξη exit τότε το tokentk παίρνει την τιμή exittk και στην λίστα tokenList γίνεται προσθήκη του tokentk και του token.

Syntax

Αφού αναγνωριστεί η λέξη exit καλείται η συνάρτηση exitstat( ).

Intermediate code & Symbol table

Ελέγχουμε αν η μεταβλητή inLoop είναι ίση με ένα και αν ισχύει τότε σημειώνουμε την επόμενη τετράδα και την αποθηκεύουμε στην μεταβλητή exitPointer. Έπειτα γράφουμε ένα κενό jump το οποίο θα συμπληρωθεί στην <loopstat>. Στον πίνακα συμβόλων, δεν προστίθεται κάτι.

Final code

Δεν παράγεται τελικός κώδικας μέσα από το <exitstat>.

<forcasestat>

Lex

Αν η λέξη που έχει περάσει στον lex( ) είναι η δεσμευμένη λέξη forcase τότε το tokentk παίρνει την τιμή forcasetk και στην λίστα tokenList γίνεται προσθήκη του tokentk και του token.

Syntax

Αφού αναγνωριστεί η λέξη forcase καλείται η συνάρτηση forcasestat( ). Αμέσως μετά υπάρχει μία while που ελέγχει αν βρίσκουμε το αναγνωριστικό whentk. Αν αυτό υπάρχει τότε ψάχνει απαραίτητα για το άνοιγμα παρένθεσης " (" και καλείται η condition( ) μετά από την οποία ελέγχει αν υπάρχει ο χαρακτήρας άνω και κάτω τελεία ":" και έπειτα καλεί την statements( ). Στη συνέχεια, αφού δεν βρει άλλο when ελέγχει την επόμενη λέξη, η οποία πρέπει να είναι η λέξη default με αναγνωριστικό το defaulttk και να ακολουθείται από ":" και κλήση της συνάρτησης statements( ). Τέλος, απαραίτητη είναι η δεσμευμένη λέξη enddefault με αναγνωριστικό το endddefaulttk, καθώς και η δεσμευμένη λέξη endforcase με αντίστοιχο αναγνωριστικό το endforcasetk. Αν λείπει οποιδήποτε από τα απαραίτητα κομμάτια, τότε ο χρήστης ενημερώνεται με μήνυμα λάθους για το ποιο από αυτά λείπει.

Intermediate code & Symbol table

Αρχικά, φτιάχνουμε μία προσωρινή μεταβλητή καλώντας την συνάρτηση newTemp( ) και την αποθηκεύουμε στην μεταβλητή t. Μαρκάρουμε την επόμενη τετράδα και την αποθηκεύουμε στην μεταβλητή flagQuad και γράφουμε στην λίστα με τα quads την τετράδα (":=","0","_",t), με την οποία βάζουμε την τιμή μηδέν στην προσωρινή μας μεταβλητή. Όσο βρίσκει when, αποθηκεύουμε το return της condition( ) σε δύο λίστες, τις condTrue και

condFalse και κάνουμε αμέσως backpatch( ) στην τετράδα που αντιστοιχεί στο condTrue την επόμενη τετράδα και γράφουμε στην λίστα με τα quads την τετράδα (“:=“,”1","_",t) πράγμα το οποίο το χρησιμοποιούμε για να δούμε αν το πρόγραμμά μας μπήκε σε κάποιο από τα when. Έπειτα κάνουμε backpatch στην τετράδα που αντιστοιχεί στο condFalse την επόμενη τετράδα. Όταν βρει το default, σημειώνουμε την επόμενη τετράδα στην μεταβλητή exitQuad και γράφουμε στην λίστα με τα quads την τετράδα (“=","1",t,"_") και την τετράδα genquad(“=“,"0",t,flagQuad). Μαρκάρουμε το nextquad στην μεταβλητή lastQuad και κάνουμε backpatch ( ) στην τετράδα που αντιστοιχεί στο exitQuad το lastQuad. Λόγω της χρήσης της newTemp, γράφει την προσωρινή μεταβλητή που έχουμε αποθηκεύσει στο w στο τωρινό scope.

Final code

Δεν παράγεται τελικός κώδικας μέσα από το <forcasestat>.

<incasestat>

Lex

Αν η λέξη που έχει περάσει στον lex( ) είναι η δεσμευμένη λέξη incase τότε το tokentk παίρνει την τιμή incasetk και στην λίστα tokenList γίνεται προσθήκη του tokentk και του token.

Syntax

Αφού αναγνωριστεί η λέξη incase καλείται η συνάρτηση incasestat( ). Αμέσως μετά υπάρχει μία while που ελέγχει αν βρίσκουμε το αναγνωριστικό whentk. Αν αυτό υπάρχει τότε ψάχνει απαραίτητα για το άνοιγμα παρένθεσης " (" και καλείται η condition( ) μετά από την οποία ελέγχει αν υπάρχει ο χαρακτήρας άνω και κάτω τελεία ":" και έπειτα καλεί την statements( ). Στη συνέχεια, αφού δεν βρει άλλο when ελέγχει την επόμενη λέξη, η οποία πρέπει να είναι η δεσμευμένη λέξη endincase με αναγνωριστικό το endincasetk. Αν αυτό λείπει τότε εμφανίζεται μήνυμα λάθους.

Intermediate code & Symbol table

Αρχικά, φτιάχνουμε μία προσωρινή μεταβλητή καλώντας την συνάρτηση newTemp( ) και την αποθηκεύουμε στην μεταβλητή t. Μαρκάρουμε την επόμενη τετράδα και την αποθηκεύουμε στην μεταβλητή flagQuad και γράφουμε στην λίστα με τα quads την τετράδα (":=","0","_",t), με την οποία βάζουμε την τιμή μηδέν στην προσωρινή μας μεταβλητή. Όσο βρίσκει when, αποθηκεύουμε το return της condition( ) σε δύο λίστες, τις condTrue και condFalse και κάνουμε αμέσως backpatch( ) στην τετράδα που αντιστοιχεί στο condTrue την επόμενη τετράδα και γράφουμε στην λίστα με τα quads την τετράδα (":=","1","_",t) πράγμα το οποίο το χρησιμοποιούμε για να δούμε αν το πρόγραμμά μας μπήκε σε κάποιο από τα

when. Έπειτα κάνουμε backpatch στην τετράδα που αντιστοιχεί στο condFalse την επόμενη τετράδα. Τέλος, γράφουμε στην λίστα με τα quads την τετράδα (“=“,”1",t,"_"). Λόγω της χρήσης της newTemp, γράφει την προσωρινή μεταβλητή που έχουμε αποθηκεύσει στο w στο τωρινό scope.

Final code

Δεν παράγεται τελικός κώδικας μέσα από το <incasestat>.

<returnstat>

Lex

Αν η λέξη που έχει περάσει στον lex( ) είναι η δεσμευμένη λέξη incase τότε το tokentk παίρνει την τιμή returntk και στην λίστα tokenList γίνεται προσθήκη του tokentk και του token.

Syntax

Αφού αναγνωριστεί η λέξη return καλείται η συνάρτηση returnstat( ), στην οποία και καλείται το expression( ).

Intermediate code & Symbol table

Στην αρχή της returnstat( ) ελέγχουμε αν το inFunction είναι θετικό. Αν είναι θετικό, αυτό σημαίνει ότι βρισκόμαστε μέσα σε τουλάχιστον ένα function και δεν έχει βρεθεί ήδη return για αυτό. Κάνουμε τη μεταβλητή foundReturn ίση με ένα και αποθηκεύουμε οτιδήποτε επιστρέφει η συνάρτηση expression( ) στην μεταβλητή returnValue. Γράφουμε στα quads την τετράδα ("ret", returnValue,"_","_") και επιστρέφουμε το returnValue. Αν το inFunction δεν ήταν θετικό, αυτό σημαίνει ότι δεν είμαστε μέσα σε συνάρτηση, οπότε δεν θα έπρεπε να υπάρχει return σε αυτό το σημείο και εμφανίζεται μήνυμα λάθους ώστε να ενημερωθεί ο χρήστης. Δεν προστίθεται κάτι στον πίνακα συμβόλων.

Final code

Δεν παράγεται τελικός κώδικας μέσα από το <returnstat>.

<printstat>

Lex

Αν η λέξη που έχει περάσει στον lex( ) είναι η δεσμευμένη λέξη print τότε το tokentk παίρνει την τιμή printtk και στην λίστα tokenList γίνεται προσθήκη του tokentk και του token.

Syntax

Αφού αναγνωριστεί η λέξη print καλείται η συνάρτηση printstat( ), στην οποία και καλείται το expression( ).

Intermediate code & Symbol table

Η τιμή επιστροφής του expression( ) αποθηκεύεται στην μεταβλητή printValue και γίνετια προσθήκη της τετράδας (“OUT",printValue,"_","_") στα quads, καθώς και επιστροφή του returnValue. Στον πίνακα συμβλολων δεν γίνεται κάποια προσθήκη.

Final code

Δεν παράγεται τελικός κώδικας μέσα από το <printstat>.

<inputstat>

Lex

Αν η λέξη που έχει περάσει στον lex( ) είναι η δεσμευμένη λέξη input τότε το tokentk παίρνει την τιμή inputtk και στην λίστα tokenList γίνεται προσθήκη του tokentk και του token.

Syntax

Αφού αναγνωριστεί η λέξη input καλείται η συνάρτηση inputstat( ), ελέγχουμε αν το αναγνωριστικό της επόμενης λεκτικής μονάδας είναι το idtk, δηλαδή αν ακολουθεί μεταβλητή μετά το input. Αν είναι οτιδήποτε άλλο εμφανίζεται μήνυμα λάθους.

Intermediate code & Symbol table

Κρατάμε την μεταβλητή στην μεταβλητή id και προσθέτουμε την τετράδα (“INP",id,"_","_").

Final code

Δεν παράγεται τελικός κώδικας μέσα από το <inputstat>.

<varlist>

Lex

Η συνάρτηση varlist( ) δεν έχει άμεση σχέση με τον lex( ).

Syntax

Αν η λεκτική μονάδα που έχουμε στο token είναι μεταβλητή, δηλαδή έχει αναγνωριστικό το idtk, τότε κάνουμε lex( ) και όσο βρίσκουμε το αναγνωριστικό commatk ξανακάνουμε lex( ) για να προσπελάσουμε και τις υπόλοιπες μεταβλητές. Στο τέλος πρέπει να υπάρχει οπωσδήποτε ο χαρακτήρας ";" που αντιστοιχεί στο αναγνωριστικό semicolontk. Αν δεν υπάρχει, τότε εμφανίζεται μήνυμα λάθους.

Intermediate code & Symbol table

Δεν δημιουργείται ενδιάμεσος κώδικας σε αυτό το σημείο, ωστόσο για κάθε μεταβλητή που θα προσπελάσουμε, καλούμε την βοηθητική συνάρτηση addVar( ) στην οποία δίνουμε σαν παράμετρο το όνομα της μεταβλητής (token) και η addVar προσθέτει στο currentScope την μεταβλητή και της δίνει το σωστό offset.

Final code

Δεν παράγεται τελικός κώδικας μέσα από το <varlist>.

<subprogram>

Lex

Η συνάρτηση subprogram( ) δεν έχει άμεση σχέση με τον lex( ).

Syntax

Με το που μπει στο subprogram( ) αναμένεται το token να είναι μία μεταβλητή η οποία αντιστοιχεί στο όνομα του προγράμματος ή της συνάρτησης στην οποία βρισκόμαστε και το tokentk να είναι idtk. Αν δεν ισχύει αυτό εμφανίζεται error. Αν ισχύει τότε αποθηκεύουμε το όνομα στην μεταβλητή subprogramName και κάνουμε lex( ) και καλούμε την funcbody με όρισμα το subprogramName. Έπειτα ψάχνουμε για το ανγνωριστικό endfunctiontk και αν δεν βρεθεί τότε εμφανίζουμε μήνυμα λάθους.

Intermediate code & Symbol table

Δεν δημιουργείται ενδιάμεσος κώδικας σε αυτό το σημείο, ωστόσο αφού βρούμε το endfunction, θέτουμε τη μεταβλητή inFunction σε -1 και ελέγχουμε αν η μεταβλητή foundReturn είναι ίση με ένα. Αν ισχύει, τότε την κάνουμε μηδέν, αλλιώς εμφανίζουμε error διότι υπάρχει συνάρτηση η οποία δεν περιέχει return.

Final code

Δεν παράγεται τελικός κώδικας μέσα από το <subprogram>.

<funcbody>

Lex

Η συνάρτηση funcbody(name) δεν έχει άμεση σχέση με τον lex( ).

Syntax

Στο funcbody(name) καλείται η συνάρτηση formalpars( ) και έπειτα η συνάρτηση block(name) με όρισμα το όνομα που έχει δοθεί σαν όρισμα στο funcbody(name).

Intermediate code & Symbol table

Με το που μπει στο funcbody(name), καλείται η βοηθητική συνάρτηση addFunc(name) η οποία παίρνει σαν όρισμα το name και προσθέτει στο τέλος του scope, την συνάρτηση με όνομα name χωρίς να γνωρίζει ακόμα το framelength της. Έπειτα καλείται η changeScope( ) η οποία αλλάζει το currentScope. Αφού τελειώσει με την κλήση της block(name), καλεί την συνάτηση closeScope( ) η οποία διαγράφει το scope που άνοιξε για το συγκεκριμένο function και επιστρέφει το frameLength της το οποίο αποθηκεύουμε στην μεταβλητή off. Τέλος, δίνουμε το off σαν όρισμα στην συνάρτηση setFuncOffset(off) η οποία ανανεώνει το offset του function το οποίο προηγουμένως δεν γνωρίζαμε.

Final code

Ακριβώς πριν καλέσουμε την closeScope( ), καλούμε την συνάρτηση finalCodeGenerator( ) ώστε να μεταφράσει όλες τις τετράδες του function που μόλις τελείωσε σε τελικό κώδικα assembly και μετά θέτουμε το quadCount ίσο με τον αριθμό των τετράδων που έχουμε μέχρι στιμγής αποθηκευμένα στη λίστα quads.

<formalpars>

Lex

Η συνάρτηση formalpars( ) δεν έχει άμεση σχέση με τον lex( ).

Syntax

Στο formalpars( ) αναμένεται να έχουμε σαν token το άνοιγμα παρένθεσης " (". Αφού βρεθεί αυτό, καλέιται ο lex( ) και η formalparlist( ). Έπειτα αναμένεται να βρεθεί το κλείσιμο παρένθεσης ") ". Αν λέιπει κάποια από τις παρενθέσεις, τότε εμφανίζεται σχετικό μήνυμα λάθους.

Intermediate code & Symbol table

Δεν παράγεται ενδιάμεσος κώδικας στην formalpars( ) ούτε προστίθεται κάτι στον πίνακα συμβόλων.

Final code

Δεν παράγεται τελικός κώδικας στην formalpars( ).

<formalparlist>

Lex

Η συνάρτηση formalparlist( ) δεν έχει άμεση σχέση με τον lex( ).

Syntax

Αν το token που έχουμε έχει αναγνωριστικό τα "intk" ή "inouttk" ή "inandouttk" τότε κάνουμε lex( ) και αν υπάρχει όνομα μεταβλητής τότε ξανακάνουμε lex, αλλιώς εμφανίζουμε error αν λείπουν είτε τα πρώτα αναγνωριστικά είτε το idtk.

Intermediate code & Symbol table

Δεν παράγεται ενδιάμεσος κώδικας στην formalparlist( ), αλλά στην αρχή της καλείται η βοηθητική συνάρτηση fillFuncVariableType(type) στην οποία περνάμε σαν όρισμα το token που μπορεί να είναι in, inout, inandout και προσθέτει στο entities, στο τελευταίο στοιχείου του προηγούμενου scope, τον τύπο της μεταβλητής (token). Την ίδια δουλειά κάνει μετά τον lex( ) η fillFuncVariables(id) η οποία συμπληρώνει και το όνομα της μεταβλητής αμέσως μετά τον τύπο της.

Final code

Δεν παράγεται τελικός κώδικας στην formalparlist( ).

<actualpars>

Lex

Η συνάρτηση actualpars(id) δεν έχει άμεση σχέση με τον lex( ).

Syntax

Αν το token που έχουμε έχει αναγνωριστικό το άνοιγμα παρένθεσης " (" καλείται ο lex( ) και η actualparlist(id). Αν δεν υπάρχει το άνοιγμα παρένθεσης, τότε εμφανίζεται μήνυμα λάθους. Αντίστοιχα, ελέγχουμε μετά αν υπάρχει το κλείσιμο παρένθεσης ") " και αν δεν υπάρχει τότε εμφανίζεται επίσης error.

Intermediate code & Symbol table

Δεν παράγεται ενδιάμεσος κώδικας στην actualpars(id), ούτε προστίθεται τίποτα στον πίνακα συμβόλων.

Final code

Δεν παράγεται τελικός κώδικας στην actualpars(id).

<actualparlist>

Lex

Η συνάρτηση actualparlist(id) δεν έχει άμεση σχέση με τον lex( ).

Syntax

Αν το token που έχουμε έχει αναγνωριστικό τα "intk" ή "inouttk" ή "inandouttk" τότε καλούμε την actualparitem(id, parameter) και όσο βρίσκουμε σαν αναγνωριστικό το commatk κάνουμε lex( ) και καλούμε πάλι την actualparitem(id, parameter).

Intermediate code & Symbol table

Στην αρχή ορίζουμε μία μεταβλητή parameter ίση με το μηδέν, η οποία μετράει πόσα ορίσματα έχει η συνάρτηση και κάθε φορά που βρίσκουμε in ή inout ή inandout την αυξάνουμε κατά ένα. Δεν προστίθεται κάτι στον πίνακα συμβόλων.

Final code

Δεν παράγεται τελικός κώδικας στην actualparlist(id).

<actualparitem>

Lex

Η συνάρτηση actualparitem(id, parameter) δεν έχει άμεση σχέση με τον lex( ).

Syntax

Αν το token που έχουμε έχει αναγνωριστικό το “intk”, τότε καλούμε τον lex( ) και την expression( ) την οποία αποθηκεύουμε στην μεταβλητή expr. Αν το token που έχουμε έχει αναγνωριστικό τα “inouttk” ή “inandouttk” τότε καλούμε τον lex( ) και αν ακολουθεί idtk αποθηκεύουμε το όνομα (token) στην μεταβλητή id και κάνουμε lex( ), αλλιώς εμφανίζουμε error.

Intermediate code & Symbol table

Στην αρχή ορίζουμε μία μεταβλητή parameterType η οποία παίρνει τιμή “in” ή “inout” ή “inandout”. Έπειτα καλούμε την συνάρτηση checkParameterType(id, parameter, parameterType) η οποία ελέγχει αν ο τύπος μεταβλητής που προσπαθούμε να χρησιμοποιήσουμε, συμπίπτει με τον τύπο μεταβλητής που έχει οριστεί στην δήλωση της συνάρτησης. Σε κάθε ένα από τα διαφορετικά είδη μεταβλητών, προστίθεται στη λίστα με τα quads η αντίστοιχη εντολή που ξεκινάει με “par”. Δεν προστίθεται κάτι στον πίνακα συμβόλων.

Final code

Δεν παράγεται τελικός κώδικας στην actualparitem(id, parameter).

<condition>

Lex

Η συνάρτηση condition( ) δεν έχει άμεση σχέση με τον lex( ).

Syntax

Στην αρχή της condition( ) καλούμε το boolterm το οποίο γυρίζει δύο λίστες τις οποίες αποθηκεύουμε στα q1True, q1False. Όσο βρίσκει σαν αναγνωριστικό το ortk, κάνει lex( ) και ξανακαλέι την boolterm( ) των οποίων το αποτέλεσμα αποθηκεύει στις μεταβλητές q2True, q2False.

Intermediate code & Symbol table

Η condition( ) θέτει το bTrue ίσο με q1True και το bFalse ίσο με q1False και όσο βρίσκει σαν αναγνωριστικό το ortk, κάνει lex( ) και έπειτα backpatch την τετράδα που αντιστοιχεί στη λίστα bFalse, το nextquad( ). Τέλος, επιστρέφει τα bTrue και bFalse. Δεν προσθέτουμε κάτι στον πίνακα συμβόλων.

Final code

Δεν παράγεται τελικός κώδικας στην condition( ).

<boolterm>

Lex

Η συνάρτηση boolterm( ) δεν έχει άμεση σχέση με τον lex( ).

Syntax

Στην αρχή της boolterm( ) καλούμε το boolfactor το οποίο γυρίζει δύο λίστες τις οποίες αποθηκεύουμε στα r1True, r1False. Όσο βρίσκει σαν αναγνωριστικό το andtk, κάνει lex( ) και ξανακαλέι την boolterm( ) των οποίων το αποτέλεσμα αποθηκεύει στις μεταβλητές r2True, r2False.

Intermediate code & Symbol table

Η condition( ) θέτει το qTrue ίσο με r1True και το qFalse ίσο με r1False και όσο βρίσκει σαν αναγνωριστικό το ortk, κάνει lex( ) και έπειτα backpatch την τετράδα που αντιστοιχεί στη λίστα qTrue, το nextquad( ). Στη συνέχεια θέτει το qFalse ως την ένωση των qFalse και r2False με την βοήθεια της mergelist( ) και θέτει το qTrue ίσο με r2True. Τέλος, επιστρέφει τα qTrue, qFalse. Δεν προστίθεται κάτι στον πίνακα συμβόλων.

Final code

Δεν παράγεται τελικός κώδικας στην boolterm( ).

<boolfactor>

Lex

Η συνάρτηση boolfactor( ) δεν έχει άμεση σχέση με τον lex( ).

Syntax

Στην boolfactor( ) ελέγχουμε αν το αναγνωριστικό είναι nottk ή αν ανοίγουν αγκύλες " [". Αν βρούμε not, τότε καλούμε τον lex( ) και περιμένουμε να βρουμε άνοιγμα αγκύλης και ξανακαλούμε τον lex( ) και επιβεβαιώνουμε ότι υπάρχει το αντίστοιχο κλέισιμο αγκύλης "] ". Αν βρούμε σκέτο άνοιγμα αγκύλης τότε καλούμε τον lex( ) και επιβεβαιώνουμε ότι υπάρχει το αντίστοιχο κλείσιμο. Αν δεν βρούμε κάποιο από τα παραπάνω καλούμε με τη σειρά την expression( ), την relationoper, ξανά την expression και αποθηκεύουμε τις τιμές σε ξεχωριστές μεταβλητές (expr1, relop, expr2 αντίστοιχα).

Intermediate code & Symbol table

Αν βρούμε not, τότε καλούμε τον lex( ) και περιμένουμε να βρουμε άνοιγμα αγκύλης και ξανακαλούμε τον lex( ). Αποθηκεύουμε την επιστροφή της condition( ) στις λίστες q1True, q1False και τις επιστρέφουμε. Αν βρούμε σκέτο άνοιγμα αγκύλης τότε πάλι καλούμε τον lex( ) και αποθηκεύουμε την επιστροφή της condition( ) στις λίστες q1True, q1False και τις επιστρέφουμε. Αν δεν βρούμε κάποιο από τα παραπάνω θέτουμε την rTrue σαν νέα λίστα που κρατάει την επόμενη τετράδα με τη βοήθεια της makelist, προσθέτουμε στα quads την τετράδα (relop,expr1,expr2,"_") και θέτουμε την rFalse σαν νέα λίστα που κρατάει την επόμενη τετράδα και προσθέτουμε στα quads ένα κενό jump. Επιστρέφουμε τα rTrue και rFalse. Δεν προστίθεται κάτι στον πίνακα συμβόλων.

Final code

Δεν παράγεται τελικός κώδικας στην boolfactor( ).

<expression>

Lex

Η συνάρτηση expression( ) δεν έχει άμεση σχέση με τον lex( ).

Syntax

Στην αρχή της expression( ) καλούμε την optionalsign( ) και το term( ) του οποίου το αποτέλεσμα αποθηκεύει στην μεταβλητή t1. Όσο βρίσκει σαν αναγνωριστικό το plustk ή το minustk, κάνει lex( ) και καλεί ξανά το term( ) του οποίου το αποτέλεσμα αποθηκεύει στην μεταβλητή t2.

Intermediate code & Symbol table

Όσο είμαστε μέσα στην while, κρατάμε τον τελεστή στην μεταβλητή oper, θέτει το w σαν μία νέα προσωρινή μεταβλητή και προσθέτει στα quads την τετράδα (oper,t1,t2,w), έπειτα θέτει το t1 ίσο με w. Τέλος, επιστρέφει το t1 που θα είναι και το συνολικό αποτέλεσμα των πράξεων. Λόγω της χρήσης της newTemp, γράφει την προσωρινή μεταβλητή που έχουμε αποθηκεύσει στο w στο τωρινό scope.

Final code

Δεν παράγεται τελικός κώδικας στην expression( ).

<term>

Lex

Η συνάρτηση term( ) δεν έχει άμεση σχέση με τον lex( ).

Syntax

Στην αρχή της term( ) καλούμε το factor( ) του οποίου το αποτέλεσμα αποθηκεύουμε στην μεταβλητή f1. Όσο βρίσκει σαν αναγνωριστικό το multtk ή το divtk, κάνει lex( ) και καλεί ξανά το factor( ) του οποίου το αποτέλεσμα αποθηκεύει στην μεταβλητή f2.

Intermediate code & Symbol table

Όσο είμαστε μέσα στην while, κρατάμε τον τελεστή στην μεταβλητή oper, θέτει το w σαν μία νέα προσωρινή μεταβλητή και προσθέτει στα quads την τετράδα (oper,f1,f2,w), έπειτα θέτει το f1 ίσο με w. Τέλος, επιστρέφει το f1 που θα είναι και το συνολικό αποτέλεσμα των πράξεων. Λόγω της χρήσης της newTemp, γράφει την προσωρινή μεταβλητή που έχουμε αποθηκεύσει στο w στο τωρινό scope.

Final code

Δεν παράγεται τελικός κώδικας στην term( ).

<factor>

Lex

Η συνάρτηση factor( ) δεν έχει άμεση σχέση με τον lex( ).

Syntax

Στην αρχή της factor( ) περιμένουμε να βρούμε άνοιγμα παρένθεσης " (" ή μεταβλητή (με αναγνωριστικό idtk) ή σταθερά (με αναγνωριστικό constanttk). Αν δεν βρούμε κάποιο από αυτά, εκτυπώνεται μήνυμα λάθους. Αν βρούμε άνοιγμα παρένθεσης, καλούμε τον lex( ), την expression( ), την οποία και αποθηκεύουμε στην μεταβλητή expr και όταν βεβαιωθούμε ότι υπάρχει το κλείσιμο παρένθεσης ") ", τότε επιστρέφουμε το expr. Αν το αναγνωριστικό ήταν το idtk, τότε κρατάμε το όνομα (token) στην μεταβλητή id και καλούμε τον lex( ) και έπειτα την idtail(id) και επιστρέφουμε το id. Τέλος, αν το αναγνωριστικό ήταν το constanttk, τότε κρατάμε την σταθερά στην μεταβλητή constant και, αφού κάνουμε lex( ), την επιστρέφουμε.

Intermediate code & Symbol table

Δεν παράγεται ενδιάμεσος κώδικας και δεν προσθέτουμε κάτι στον πίνακα συμβόλων.

Final code

Δεν παράγεται τελικός κώδικας στην <factor>.

<idtail>

Lex

Η συνάρτηση idtail(id) δεν έχει άμεση σχέση με τον lex( ).

Syntax

Στην αρχή της idtail(id) ελέγχουμε αν το αναγνωριστικό είναι το opbrackettk, δηλαδή το άνοιγμα παρένθεσης " (". Έπειτα καλούμε την actualpars(id)

Intermediate code & Symbol table

Αν το αναγνωριστικό στην αρχή είναι το opbrackettk καλούμε τη συνάρτηση checkFunc(id) η οποία ελέγχει αν το id είναι όντως όνομα συνάρτησης και έπειτα προσθέτει στα quads την τετράδα ("call",id,"_","_"). Δημιουργούμε μία προσωρινή μεταβλητή την οποία αποθηκεύουμε στο w και με τη χρήση της addVar(w) την προσθέτουμε και στον πίνακα συμβόλων. Τέλος, κάνουμε genquad("par", w, "RET", "_") ώστε να προσθέσουμε και αυτή την τετράδα. Λόγω της χρήσης της newTemp, γράφει την προσωρινή μεταβλητή που έχουμε αποθηκεύσει στο w στο τωρινό scope. Αν το αναγνωριστικό δεν ήταν άνοιγμα παρένθεσης, τότε καλεί την checkVar(id) για να ελέγξει ότι η μεταβλητή που μόλις έφτασε στο idtail είναι όντως μεταβλητή.

Final code

Δεν παράγεται τελικός κώδικας στην <idtail>.

<relationoper>

Lex

Η συνάρτηση relationoper( ) δεν έχει άμεση σχέση με τον lex( ).

Syntax

Όταν καλέιται η relationoper( ) τότε περιμένουμε να βρούμε ένα σύμβολο σύγκρισης, οπότε με μία πολλαπλή if, αντιστοιχίζουμε το αναγνωριστικό του με ένα από τα equaltk, lessorequaltk, moreorequaltk, moretk, lesstk, differenttk και αποθηκεύει το token στην μεταβλητή oper, το οποίο και επιστρέφει. Αν δεν αντιστοίχισε το tokenid σωστά, τοτε εμφανίζει μήνυμα ότι περίμενε να βρει operator.

Intermediate code & Symbol table

Δεν παράγεται ενδιάμεσος κώδικας και δεν προστίθεται οτιδήποτε στον πίνακα συμβόλων.

Final code

Δεν παράγεται τελικός κώδικας στην <relationoper>.

<optionalsign>

Lex

Η συνάρτηση optionalsign( ) δεν έχει άμεση σχέση με τον lex( ).

Syntax

Όταν καλέιται η optionalsign( ) τότε αν το αναγνωριστικό του token είναι το plustk ή το minustk καλείται ο lex( ).

Intermediate code & Symbol table

Δεν παράγεται ενδιάμεσος κώδικας και δεν προστίθεται οτιδήποτε στον πίνακα συμβόλων.

Final code

Δεν παράγεται τελικός κώδικας στην <optionalsign>.

Χρήση Του Compiler

Για να χρησιμοποιήσει κάποιος τον compiler μας, τότε αρκεί να ανοίξει ένα τερματικό και να γράψει “met.py filename”, όπου filename.stl το όνομα του αρχείου Starlet που θέλει να μεταγλωττίσει. Ο compiler μας ξεκινάει ανοίγοντας το αρχείο αυτό, αποθηκεύοντάς το στην μεταβλητή c και καλεί τη συνάρτηση program( ). Έπειτα για να εξάγει τον ενδιάμεσο κώδικα σε αρχείο C, τότε καλεί την printCFile( ), η οποία παίρνει τις τετράδες τις μετατρέπει σε κώδικα C και τις εξάγει στο αρχείο filename.c . Έπειτα για να εξάγει τις τετράδες σε αρχείο καλεί την printQuadsInFile( ), η οποία γράφει όλα τα quads στο αρχείο filename.int . Τέλος, για την εξαγωγή του τελικού κώδικα έχουμε αναφερθεί παραπάνω, στα αντίστοιχα κομμάτια του <program> και <funcbody> όπου καλείται η finalCodeGenerator( ) και στο τέλος-τέλος η printFinalCodeInFile( ) η οποία εξάγει όλο τον τελικό κώδικα στο αρχείο filename.asm

Επίλογος

Το project των μεταφραστών μας βοήθησε να ανακαλύψουμε τον τρόπο μεταγλώττισης οποιασδήποτε γλώσσας προγραμματισμού και να αποκτήσουμε ικανότητες στο να γράφουμε έναν ολοκληρωμένο και αποδοτικό σε ικανοποιητικό βαθμό μεταγλωττιστή.

Κλείνοντας, θέλουμε να ευχαριστήσουμε τον κύριο Μανή για την καθοδήγησή του καθ όλη τη διάρκεια του εξαμήνου, διότι χωρίς τα παραδείγματα που έδειξε στις διαλέξεις καθώς και χωρίς τις απαντήσεις που έδωσε σε όλα τα ερωτήματα που μας δημιουργήθηκαν, δεν θα μπορούσαμε να ολοκληρώσουμε το project.