ΜΕΤΑΦΡΑΣΤΕΣ

REPORT

Γιάτσος Γεώργιος Α.Μ.: 3202

1η Φάση Λεκτική-Συντακτική ανάλυση:

Το πρώτο κομμάτι του μεταφραστή αποτελείται από τον Lexer (def lex). Ο ρόλος του είναι να πάρει το πηγαίο πρόγραμμα ως είσοδο, να το διαβάσει γράμμα-γράμμα και να το διαχωρίσει σε tokens. Επιστρέφει στο συντακτικό αναλυτή έναν ακέραιο που χαρακτηρίζει τη λεκτική μονάδα και τη λεκτική μονάδα (token).

Η μέθοδος lex υλοποιεί τον λεκτικό αναλυτή. Η λογική αυτής της μεθόδου είναι ότι ελέγχει ένα αρχείο και μέσω τον μεθόδων της δημιουργεί τα tokens. Αυτά τα tokens αποθηκεύονται σε ένα global πίνακα tokens. Επίσης περιλαμβάνονται και μερικές global μεταβλητές.

Αρχικά, τοποθετούμε κάποιους global πίνακες και μεταβλητές ώστε το αυτόματο καταστάσεων του λεκτικού αναλυτή να αναγνωρίζει δεσμευμένες λέξεις, σύμβολα της γλώσσας, αναγνωριστικά και σταθερές, λάθη, αλλά και αναγνωριστικά που ξεκινούν από γράμμα και αποτελούνται από γράμματα ή ψηφία, φυσικούς αριθμούς ως αριθμητικές σταθερές, αριθμητικά σύμβολα, σχεσιακούς τελεστές, σύμβολα και σχόλια που περικλείονται σε άγκιστρα “{“ “}” και αγνοούνται. Έπειτα, διαβάζει το αρχείο και τοποθετεί λέξη ή χαρακτήρα σε πίνακα.

Οι μέθοδοι που χρησιμοποιούνται είναι οι:

• word_to_char(array):

Δέχεται έναν πίνακα (array) ως όρισμα και με μία βοηθητική λίστα, διατρέχει αυτόν τον πίνακα που περιέχει ολόκληρες λέξεις ή χαρακτήρες και επιστρέφει ένα νέο πίνακα με ένα χαρακτήρα ανά θέση και χωρίς κενά για τη διευκόλυνση του αυτομάτου καταστάσεων.

• state_auto(array1,start,stop):

Η μέθοδος αυτή είναι το αυτόματο καταστάσεων. Η υλοποίηση γίνεται με σειρά εντολών απόφασης και ανά τριάντα χαρακτήρες state_auto (array,start+30,stop+30).

• main():

Η μέθοδος main απλώς συμβάλλει στη σωστή σειρά λειτουργίας του αρχικού πίνακα της word_to_char και της state_auto.

• connectSyntaxLex():

Επιστρέφει στο συντακτικό αναλυτή ένα ένα τα tokens που διαχωρίζει.

Οι global μεταβλητές είναι:

lextotokens = []

filename = input("Enter filename:")

file = open(filename,"r")

Και το αλφάβητο:

letters=['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','a','b','c','d','e','f','g','h','i','g','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z']

numbers=[0,1,2,3,4,5,6,7,8,9]

math_symbols=['+','-','*','/']

relations=['<','>','=','<=','>=','<>']

delimeters=[';',',',':']

operators=['(',')','{','}','[',']']

comments=['/*','*/','//']

commited_words=['program','declare','function','procedure','in','inout','if','else','while','doublewhile','loop','exit','forcase','incase','when','default','not','and','or','call','return','input','print']

Για τον συντακτικό αναλυτή (syntax) έχω κάποιες μεθόδους που δουλεύουν για να αναγνωρίζουν το συντακτικό της Minimal++. Αρχικά, γίνεται έλεγχος για να διαπιστωθεί αν το πηγαίο πρόγραμμα ανήκει ή όχι στη γλώσσα Minimal++. Βασίζεται σε γραμματική LL(1), αναγνωρίζει

από αριστερά στα δεξιά την αριστερότερη δυνατή παραγωγή και όταν βρίσκεται σε δίλλημα ποιον κανόνα να ακολουθήσει της αρκεί να κοιτάξει το αμέσως επόμενο σύμβολο στην συμβολοσειρά εισόδου. Η ονομασία και η λειτουργία των διάφορων μεθόδων μας δίνονται από την εκφώνηση της άσκησης. Η λογική είναι ότι κάνω προσπέλαση τον πίνακα tokens και, ανάλογα τα tokens με την σειρά που μπήκαν και τις αλλαγές γραμμής, κάνουμε τους ελέγχους που ζητάει το συντακτικό της Minimal++.

Οι global μεταβλητές είναι ένας counter και χρήσιμες μεταβλητές για τις τετράδες.

```
global counter
    E_place =''
    F_place = ''
    T_place = ''
    ID_place = ''
    B_true = []
    B_false = []
    Q_true = []
    Q_false = []
    R_true = []
    R_false = []
    COND_true = []
    COND_false = []
```

Οι μέθοδοι που χρησιμοποιούνται αφορούν αποκλειστικά τη γραμματική της Minamal++.

2η Φάση Παραγωγή Ενδιάμεσου Κώδικα:

Για την υλοποίηση του ενδιάμεσου κώδικα δημιουργώ μία κλάση Quad (ένα σύνολο από τετράδες με έναν τελεστή και τρία τελούμενα). Η λογική είναι ότι οι τετράδες είναι αριθμημένες και κάθε τετράδα έχει μπροστά της έναν αριθμό (quad) που τη χαρακτηρίζει. Μόλις τελειώσει η εκτέλεση μιας τετράδας εκτελείται η τετράδα που έχει τον αμέσως μεγαλύτερο αριθμό, εκτός εάν η τετράδα που μόλις εκτελέστηκε υποδείξει κάτι διαφορετικό. Επίσης, οι τετράδες χρησιμοποιούνται για την παραγωγή εντολών σε assembly, και για την υλοποίηση αρχείων σε c.

Έχω κάποιες global μεταβλητές:

InterCarray = []

nextquad = 0

vartemp = dict()

nextvartemp = 1

• InterCarray: πίνακας από quads.

• nextquad: δείχνει σε ποιο στοιχείο του quad βρισκόμαστε.

• vartemp: λεξικό όπου κρατάει τον αριθμό των quads.

• nextvartemp: μετρητής για να αυξάνουμε τον αριθμό στο λεξικό.

Η κλάση για τον ενδιάμεσο κώδικα η Quads και υλοποιώ 3 private μεθόδους:

• _init_:

Αρχικοποιώ τα quad, p(operator), x(1ο όρισμα), y(2ο όρισμα) και το z(αποτέλεσμα).

• _str_(self):

Μετατρέπω την κλάση Quads στην κατάλληλη μορφή 4άδας.

• makefile(self):

Τη χρησιμοποιώ για να γράψω στο αρχείο .int.

Έχω ξεχωριστές συναρτήσεις που υλοποιούν τις λειτουργίες του ενδιάμεσου κώδικα:

• nextquad():

Επιστρέφει την global μεταβλητή nextquad.

• genquad(p, x, y ,z):

Δημιουργεί τις τετράδες και τις τοποθετεί στη λίστα με τα quads.

• newtemp():

Προσθέτει στο λεξικό έναν καινούργιο αριθμό ανάλογα με την τετράδα, που θα χρησιμοποιηθεί ως ετικέτα σε κάθε τετράδα

• emptyList():

Δημιουργεί μια κενή λίστα ετικετών τετράδων.

• makelis()t:

Παίρνει σαν όρισμα μία ετικέτα και δημιουργεί μια λίστα ετικετών τετράδων που περιέχει μόνο την ετικέτα αυτή.

• merge():

Παίρνει σαν όρισμα δύο λίστες και δημιουργεί μια λίστα ετικετών τετράδων από την συνένωση των δύο λιστών.

• backpatch():

Παίρνει σαν όρισμα έναν πίνακα από τετράδες στους οποίους το αποτέλεσμα (z: το τελευταίο τελούμενο) δεν είναι συμπληρωμένο και μια ετικέτα η οποία θα συμπληρώσει τις τετράδες αυτές με την ετικέτα z.

• writetofile_int():

Παίρνει τον πίνακα InterCarray και γράφει στο αρχείο .int μία μία τις τετράδες με τις ετικέτες τους.

• vardeclare():

Παίρνει σαν όρισμα μία τετράδα και βρίσκει ποιες μεταβλητές πρέπει να δηλωθούν.

•changedeclare():

Παίρνει σαν όρισμα μία μεταβλητή και τη μετατρέπει στην αντίστοιχη μεταβλητή της γλώσσας C.

• convert_c(quad):

Μετατρέπει τις τετράδες σε κώδικα C.

• make_c():

Δημιουργεί το αρχείο C.

<u>3η Φάση Πίνακας Συμβόλων:</u>

Δημιουργώ έναν global πίνακα scope_array [] ο οποίος κρατάει όλα τα scopes του κώδικα. Τα scopes δημιουργούνται όταν μπαίνουμε σε μια συνάρτηση και κρατάνε πληροφορίες όπως τις παραμέτρους (in,inout), τις μεταβλητές των παραμέτρων (π.χ. x,y), το όνομα της συνάρτησης και λοιπά.

Δημιουργώ μια κλάση Scope με τα εξής χαρακτηριστικά:

--nestlvl =0 που είναι το βάθος φωλιάσματος

--enclosescope =None που δειχνει στο scope που βρίσκεται από πάνω του

• addEntity():

Ένας πίνακας που περιέχει όλα τα entities και προσθέτει κάθε καινούργιο εκεί.

• getOffset():

Δείχνει την απόσταση από την κορυφή της στοίβας και το αυξάνει κατά 4 για να είναι έτοιμο να δείξει τη νέα μεταβλητή.

Έπειτα δημιουργώ τη κλάση Argument με παραμέτρους:

--partype: που δείχνει τον τρόπο περάσματος του argument

--next_arg =None που δείχνει στο επόμενο argument

• set_next():

Που ενημερώνει το next_arg.

Ακόμα δημιουργώ μια κλάση Entity που έχει παραμέτρους όνομα (name) και τύπο (type).

Μία κλάση Variable που κάνει extend το Entity και έχει offset ανάλογο με αυτό που υπάρχει στο Scope.

Μια κλάση Function που κάνει extend το Entity και έχει ένα startquad που αφορά τον ενδιάμεσο κώδικα και αρχικοποιεί:

• _init_(): την πρώτη τετράδα της function τετράδας

• add_arguments(): ένα πίνακα arguments με τα arguments

• setframelength(): ένα framelength με το framelength της συνάρτησης.

Επιπλέον έχω μια κλάση TempVariable που κάνει extend το Entity και έχει το offset του εκάστοτε scope.

Τέλος έχω και μια κλάση Parameter που κάνει extend το Entity, έχει ένα partype το οποίο θα παίρνει είτε "in", "inout” και το offset του εκάστοτε scope.

Στη συνέχεια δημιούργησα μεθόδους για την υλοποίηση του πίνακα συμβόλων:

• addscope(): Δημιουργεί νέο Scope μόλις μπούμε σε συνάρτηση

• addparentity():Προσθέτει ένα entity Parameter

• addfunctentity(): Προσθέτει ένα function entity

• updatequad_functentity(): Ενημερώνει το αρχικό quad label της entity Function

• updateframelength_functentity: Ενημερώνει το framelength της συνάρτησης.

• addvarentity(): Προσθέτει ένα entity Variable

• addfunctarg(): Προσθέτει argument σε μια υπάρχουσα συνάρτηση

• search_entity(): Ελέγχει να βρει entity με το συγκεκριμένο όνομα και τύπο στο εκάστοτε scope και τα enclosing scope.

• search_entity_by_name(): Ελέγχει entity με το συγκεκριμένο όνομα στο εκάστοτε scope και τα enclosing scope.

• unique_entity(): Ελέγχει να βρει αν υπάρχει κι άλλο entity με το ίδιο όνομα και τύπο.

• varispar(): Ελέγχει να βρει αν υπάρχει το όνομα ήδη δηλωμένο σαν parameter.

• printscope(): Τυπώνει το εκάστοτε scope και το προηγούμενό του.

Τοποθέτησα και κάποιες global μεταβλητές:

• scope_count = 0 μετράει το level που βρίσκεται το scope.

• scope_array = [] ο πίνακας με τα scopes.

• mainframelength=-1 το μήκος εγγραφήματος δραστηριοποίησης.

3η Φάση (συνέχεια) Τελικός κώδικα:

Σε αυτή τη φάση πρέπει να γραφτούν σε ένα αρχείο (assembly.asm) κατάλληλες εντολές ανάλογα με τις τετράδες που έχουν δημιουργηθεί από την δεύτερη φάση και με βάση τα scopes του προγράμματος. Για την υλοποίηση του χρησιμοποίησηα τρεις βοηθητικές μεθόδους

• gnvlcode (variable, register) όπου:

Α)μεταφέρει στον $t0 την διεύθυνση μιας μη τοπικής μεταβλητής

Β)από τον πίνακα συμβόλων βρίσκει πόσα επίπεδα επάνω βρίσκεται η μη τοπική μεταβλητή και μέσα από τον σύνδεσμο προσπέλασης την εντοπίζει

• loadvr (variable, register) για:

Α) τη μεταφορά δεδομένων στον καταχωρητή r

Β) τη μεταφορά που μπορεί να γίνει από τη μνήμη (στοίβα)

Γ) να εκχωρηθεί στο r μία σταθερά και διακρίνω τις απαραίτητες περιπτώσεις

• storerv (register, variable) :

Για τη μεταφορά δεδομένων από τον καταχωρητή r στη μνήμη (μεταβλητή v)

και μία • transform_to_assembly(quad,element):

Για τη σωστή μετατροπή των τετράδων ενός element σε εντολές assembly.

Χρησιμοποιώ και κάποιες global μεταβλητές:

• branches = ('beq', 'bne', 'blt', 'ble', 'bgt', 'bge')

• math_operations = ('add', 'sub', 'mul', 'div')

• Υποτίθεται υπάρχει ένα assembly_file: αρχείο εγγραφής εντολών της Assembly, το οποίο δεν υλοποίησα.

• quads = []: απαραίτητος πίνακας που αποτελείται από τετράδες όταν ο operator είναι par.

• counterformips = 0 ένας μετρητής του αριθμού των συναρτήσεων του προγράμματος

Στην εκκίνηση του κώδικα εμφανίζεται μόνο το “Enter filename:” . Εκεί τοποθετώ ένα αρχείο .txt που δημιούργησα εγώ ως εξής:

```
program giorgos
{
	declare T_1, p;
	function giatsos(in p, in x, inout g)
	{
		if(p>x)
			then print(ok)
		else
			return(g+1)
	}
	procedure d(in w, inout q)
	{
		while(w>=q)
			call kostas(w,q)
	}
}
```

Παρ΄όλα αυτά δεν εμφανιζόταν κάποιο αποτέλεσμα που με κάνει να σκέφτομαι ότι έσφαλα σε κάποιες εκτυπώσεις (print).

Τέλος πιθανά λάθη βρίσκονται:

Α) Στον lex(): και κυρίως στο state_auto

Β) Στο syntax(): μάλλον δεν δέχεται σωστά κάποια tokens

Γ) Στον ενδιάμεσο κώδικα και στον πίνακα συμβόλων ίσως ήθελε προσθήκη των μεθόδων τους και στους αναλυτές μου αλλά και στις μετατροπές σε.int και .c υπάρχουν σφάλματα, ενώ

Δ) Στον τελικό κώδικα που τελικά δεν μπόρεσα να κάνω κάποιο assembly_file και επομένως ίσως οι μετατροπές των τετράδων σε εντολές assembly έχουν λάθη.

Σαν σύνταξη ο κώδικας δεν εμφανίζει errors.