ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ

MINIMAL++ COMPILER

ΜΕΤΑΦΡΑΣΤΕΣ

2019-2020

ΕΛΕΝΗ ΜΟΥΖΑΚΗ, 3280

ΠΑΝΑΓΙΩΤΗΣ ΠΑΠΑΪΩΑΝΝΟΥ, 3309

ΤΕΛΙΚΗ ΑΝΑΦΟΡΑ

Ιούνιος 2020

1 ΕΙΣΑΓΩΓΗ

Το πρόγραμμα που δημιουργήσαμε αποτελεί έναν μεταφραστή για τη γλώσσα minimal++. Η εργασία υλοποιήθηκε σε 3 φάσεις:

Φάση 1:

- Λεκτική ανάλυση
- Συντακτική ανάλυση

Φάση 2:

- Παραγωγή ενδιάμεσου κώδικα

Φάση 3:

- Πίνακας συμβόλων
- Παραγωγή τελικού κώδικα

2 ΦΑΣΗ 1

2.1 ΛΕΚΤΙΚΗ ΑΝΑΛΥΣΗ

Για την πραγματοποίηση της λεκτικής ανάλυσης δημιουργήθηκε η συνάρτηση lex(). Η συνάρτηση αυτή διαβάζει το πηγαίο πρόγραμμα χαρακτήρα-χαρακτήρα και έχει ως στόχο να επιστρέφει λεκτικές μονάδες είτε, εάν θεωρηθεί αναγκαίο, μηνύματα λάθους.

Η lex() αποτελεί ένα αυτόματο καταστάσεων, το οποίο αναγνωρίζει δεσμευμένες λέξεις, τα σύμβολα της γλώσσας και αναγνωριστικά. Ανάλογα με τον χαρακτήρα που λαμβάνει ως είσοδο, από την αρχική του κατάσταση μεταβαίνει σε μία από τις υπόλοιπες 7 καταστάσεις.

- Κατάσταση 0 – Αρχική κατάσταση: Όσο ο επόμενος χαρακτήρας είναι το «κενό» τον προσπερνάει, ενώ εάν βρει οποιονδήποτε άλλο χαρακτήρα μεταβαίνει στην αντίστοιχη κατάσταση.
- Κατάσταση 1: Σε αυτή την κατάσταση μεταβαίνει μόλις έχει συναντήσει ένα γράμμα. Διαβάζει τον επόμενο χαρακτήρα και με βάση τη γραμματική της minimal++ δημιουργεί μια λέξη.
- Κατάσταση 2: Σε αυτή την κατάσταση μεταβαίνει μόλις έχει συναντήσει έναν αριθμό. Διαβάζει τον επόμενο χαρακτήρα και με βάση τη γραμματική της minimal++ δημιουργεί έναν αριθμό.
- Κατάσταση 3 – 4 – 5: Μεταβαίνει σε μία από αυτές τις καταστάσεις εάν έχουν διαβαστεί οι χαρακτήρες «<», «>», «:» αντίστοιχα, οι οποίο μπορούν να ακολουθηθούν από κάποιον άλλο ειδικό χαρακτήρα.

- Κατάσταση 6: Σε αυτή την κατάσταση μεταβαίνει μόλις έχει συναντήσει τον ειδικό χαρακτήρα «/». Η δουλειά της είναι να αναγνωρίσει αν πρόκειται να δημιουργηθεί κάποιο είδος σχολίου ή εάν αποτελεί μία απλή διαίρεση.
- Κατάσταση -1 – Τελική κατάσταση: Σε αυτή την κατάσταση μεταβαίνει εάν έχει τελειώσει η διαδικασία δημιουργίας μιας λέξης, την οποία μάλιστα επιστρέφει.

2.2 ΣΥΝΤΑΚΤΙΚΗ ΑΝΑΛΥΣΗ

Σε αυτό το σημείο διαπιστώνεται αν το πηγαίο πρόγραμμα ανήκει στη γλώσσα minimal++. Ο συντακτικός αναλυτής κατασκευάστηκε με αναδρομική κατάβαση.

Για κάθε έναν από τους κανόνες γραμματικής της γλώσσας φτιάξαμε και συνάρτηση, ενώ κάθε μία από αυτές καλεί τη συνάρτηση lex() όπου θεωρείται αναγκαίο. Εάν ο λεκτικός αναλυτής δεν φέρει το αποτέλεσμα που ορίζεται από την γραμματική, το πρόγραμμα τερματίζει εμφανίζοντας το ανάλογο μήνυμα λάθους.

Η συντακτική ανάλυση ξεκινάει από τη συνάρτηση program().

2.3 ΚΑΘΟΛΙΚΕΣ ΜΕΤΑΒΛΗΤΕΣ

Για την ορθή λειτουργία τον παραπάνω αναλύσεων χρησιμοποιήθηκαν οι global μεταβλητές:

- line: κρατάει τον αριθμό της γραμμής που διαβάζει στον πηγαίο κώδικα και χρησιμοποιείται σε μηνύματα λάθους.
- token: για τη συνεργασία λεκτικού και συντακτικού αναλυτή. Κρατάει τη λέξη που επιστρέφεται κάθε φορά από τον λεκτικό αναλυτή.

3 ΦΑΣΗ 2

3.1 ΠΑΡΑΓΩΓΗ ΕΔΙΑΜΕΣΟΥ ΚΩΔΙΚΑ

Αυτή η φάση μας έφερε ένα βήμα πιο κοντά στην δημιουργία κώδικα σε γλώσσα μηχανής και επομένως την μετάφραση προγραμμάτων σε γλώσσα minimal++.

Με την παραγωγή του ενδιάμεσου κώδικα δημιουργείται ένα σύνολο από τετράδες, τα quads, οι οποίες αντιπροσωπεύουν τον πηγαίο κώδικα. Κάθε εντολή του πηγαίου κώδικα αντιστοιχεί σε μία σειρά από τετράδες, τις οποίες δημιουργήσαμε με τη βοήθεια των συναρτήσεων nextquad(), genquad(), newTemp(), makeList(), mergeList(), backpatch().

Με τη χρήση των παραπάνω συναρτήσεων και με τη βοήθεια των διαφανειών του μαθήματος, δημιουργήσαμε αντιστοιχίες μεταξύ της γραμματικής της minimal++ και των quads. Όσο αφορά την forecase, η οποία μας ζητήθηκε να αναλύσουμε μόνοι μας την μεταφράσαμε σε τετράδες με αυτόν τον τρόπο:

S -> forecase {P1}

(when: (condition): {P2} statements[1] {P3}) *

default: statements[2]

{P1}: Bquad = nextquad()

{P2}: backpatch(cond. true, nextquad())

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

backpatch(cond. false, nextquad())

Σε κάθε δημιουργία ενός quad, αυτό προστίθεται σε μία λίστα που περιέχει όλες τις τετράδες που έχουν δημιουργηθεί μέχρι εκείνη τη στιγμή.

3.2 ΔΗΜΙΟΥΡΓΙΑ .INT ΚΑΙ .C ΑΡΧΕΙΟΥ

Για τη δημιουργία του προγράμματος σε ενδιάμεση γλώσσα, έχει κατασκευαστεί η συνάρτηση createIntFile(). Η συνάρτηση αυτή διατρέχει την λίστα των quads και τις γράφει σε ένα αρχείο με κατάληξη .int.

Για την δημιουργία του ισοδύναμου προγράμματος με την ενδιάμεση γλώσσα σε γλώσσα C, έχει κατασκευαστεί η συνάρτηση createCFile(). Η συνάρτηση αυτή διατρέχει τη λίστα των quads και αντιστοιχεί κάθε quad σε εντολή κώδικα C.

3.3 ΚΑΘΟΛΙΚΕΣ ΜΕΤΑΒΛΗΤΕΣ

Για την ορθή λειτουργία της παραγωγής του ενδιάμεσου κώδικα χρησιμοποιήθηκαν οι global μεταβλητές:

- num_quad: κρατάει την ετικέτα της τετράδας
- quad_list: λίστα στην οποία αποθηκεύονται όλα τα quads μετά την δημιουργία τους (γεμίζει κατά τη διάρκεια του συντακτικού αναλυτή και αδειάζει στην παραγωγή του τελικού κώδικα)
- temp_num: κρατάει τον αριθμό των προσωρινών μεταβλητών που χρησιμοποιήθηκαν στα quads.
- program_name: κρατάει το όνομα του κύριου προγράμματος για την σωστή χρήση της τετράδας για τον τερματισμό τους προγράμματος (halt).
- pars, depth: χρησιμεύουν για τη σωστή δημιουργία τετράδων σε περιπτώσεις εμφωλευμένων κλήσεων συναρτήσεων
  π.χ. max (in max (in x, in y), in max (in z, in k)).

4 ΦΑΣΗ 3

4.1 ΠΙΝΑΚΑΣ ΣΥΜΒΟΛΩΝ

Ο πίνακας συμβόλων συνδέεται άμεσα με την παραγωγή του τελικού κώδικα καθώς το εγγράφημα δραστηριοποίησης περιέχει χρήσιμες πληροφορίες για την παραγωγή του τελικού κώδικα.

Για κάθε ενέργεια στον πίνακα συμβόλων δημιουργήσαμε τις αντίστοιχες συναρτήσεις: addNewScope(), addNewVarEntity(), addNewParEntity(), addNewTempEntity(), addNewFuncEntity(), addNewArgument(), searchFuncEntity(), searchEntity(), deleteScope(). Κάθε μία από αυτές τις συναρτήσεις ενσωματώθηκε στον κώδικα του συντακτικού αναλυτή και σε αυτόν του ενδιάμεσου κώδικα.

Ο πίνακας συμβόλων μας αποτελείται από scopes. Κάθε scope αποτελεί μία λίστα από λίστες, τα entities, ενώ η πρώτη θέση κάθε λίστας scope είναι περιέχει το nested level του scope μαζί με το offset του αυξημένο κατά τέσσερα.

Μορφή των entities:

- Μεταβλητή: [String var_name, String type, int offset]
- Παράμετρος: [String par_name, String type, String par_mode, int offset]
- Προσωρινή μεταβλητή: [String temp_name, String type, int offset]
- Συνάρτηση: [String func_name, String type, int startQuad, String[ ] arguments, int framelength]

Όσον αφορά τις αναζητήσεις για entities, η searchFuncEntity() επικεντρώνεται στην αναζήτηση entities function ή procedure μέσα στον πίνακα συμβόλων (ξεκινώντας από το τελευταίο scope), ενώ η searchEntity() αγνοεί τα function και procedure. Και οι δύο συναρτήσεις επιστρέφουν το entity που βρέθηκε και το nested level του.

Ακόμη έχει υλοποιηθεί μία συνάρτηση printScopes() η οποία τυπώνει στο τερματικό τον πίνακα συμβόλων, αλλάζοντας μία καθολική μεταβλητή από false σε true.

4.2 ΠΑΡΑΓΩΓΗ ΤΕΛΙΚΟΥ ΚΩΔΙΚΑ

Για την παραγωγή του τελικού κώδικα, από κάθε εντολή του ενδιάμεσου κώδικα παραγάγαμε τις αντίστοιχες εντολές του τελικού. Για να συμβεί αυτό χρειάστηκε η δημιουργία των συναρτήσεων fillFinalCodeList(), generateFinalCode() και fillAsmFile().

Η fillFinalCodeList() έχει ως ρόλο να διατρέχει τη λίστα των quads, η οποία περιέχει όλες τις τετράδες που έχουν δημιουργηθεί έως εκείνη τη στιγμή, και να καλεί την generateFinalCode() για κάθε μία από αυτές.

Η generateFinalCode() με τη βοήθεια των gnlvcode(), loadvr() και storerv(), μετατρέπει κάθε τετράδα στην αντίστοιχη εντολή assembly του επεξεργαστή MIPS, λαμβάνοντας υπόψιν κάθε πιθανό σενάριο.

Η fillAsmFile() δέχεται τις εντολές σε assembly που δημιουργήθηκαν από τις fillFinalCodeList () και generateFinalCode() και τις αποτυπώνει σε ένα .asm αρχείο.

4.3 ΚΑΘΟΛΙΚΕΣ ΜΕΤΑΒΛΗΤΕΣ

Για την ορθή υλοποίηση του πίνακα συμβολών και παραγωγή του τελικού κώδικα χρησιμοποιήθηκαν οι global μεταβλητές:

- scopes: λίστα που αποτελεί τον πίνακα συμβόλων
- nested_level: κρατάει το συνολικό επίπεδο φωλιάσματος του πίνακα συμβόλων
- func_flag: χρησιμοποιείται για την αναγνώριση procedure και function
- print_scopes: false για να μην τυπώνεται ο πίνακας συμβόλων στο τερματικό και true για να τυπώνεται
- final_quad_list: λίστα με όλες τις τετράδες
- final_code_list: λίστα από λίστες που περιέχει όλες τις εντολές assembly
- par_index: ο αύξων αριθμός της παραμέτρου για την εντολή par, x, CV, _
- helper: κρατάει το index του final_code_list, διότι η fillAsmFile() λειτουργεί ανά block και χρειάζεται να γνωρίζει από ποια θέση του final_code_list να συνεχίσει.
- returned: λίστα που κρατάει το όνομα κάποιου function και έναν αριθμό για το αν έχει χρησιμοποιηθεί το return (0 ή 1)

5 ΕΚΤΕΛΕΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ

Το πρόγραμμα εκτελείται στο τερματικό με τον εξής τρόπο:

python final.py myProgram.min

Στην τελευταία παράδοση του κώδικα για τον μεταφραστή minimal++, συμπεριλήφθηκαν 9 αρχεία test, τα οποία μεταφράστηκαν με επιτυχία και «έτρεξαν» σωστά στον προσομοιωτή MARS.