ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Αναφορά Εξαμηνιαίας Εργασίας

στο μάθημα 8ου εξαμήνου (ροής Λ)

Μεταγλωττιστές

Θέμα εργασίας: “*Ανάπτυξη ενός μεταγλωττιστή για τη γλώσσα προγραμματισμού Alan*”

Στοιχεία ομάδας

Ονοματεπώνυμο:	ΜΑΘΙΟΥΛΑΚΗ ΕΛΕΝΗ
Α.Μ.:	03114040
E-mail:	el.mathioulaki@gmail.com

Ονοματεπώνυμο:	ΝΙΑΡΧΟΣ ΣΩΤΗΡΙΟΣ
Α.Μ.:	03114076
E-mail:	sot.niarchos@gmail.com

Εισαγωγή

Αντικείμενο της εργασίας μας είναι η ανάπτυξη ενός μεταγλωττιστή για τη γλώσσα προγραμματισμού Alan, μία απλή προστακτική γλώσσα προγραμματισμού που περιγράφεται αναλυτικά στην εκφώνηση της εργασίας.

Η παρούσα αναφορά χωρίζεται σε αρκετά μέρη, ένα για κάθε τμήμα του μεταγλωττιστή. Συγκεκριμένα, η παρουσίαση θα γίνει ως εξής:

1. Εποπτεία της συνολικής αρχιτεκτονικής του μεταγλωττιστή
2. Λεκτικός αναλυτής (lexer)
3. Συντακτικός αναλυτής (parser) και κατασκευή αφαιρετικού συντακτικού δέντρου (AST)
4. Σημασιολογικός αναλυτής
5. Παραγωγή ενδιάμεσου κώδικα και βελτιστοποίηση
6. Παραγωγή τελικού κώδικα και βελτιστοποίηση
7. alanc – ο τελικός μεταγλωττιστής

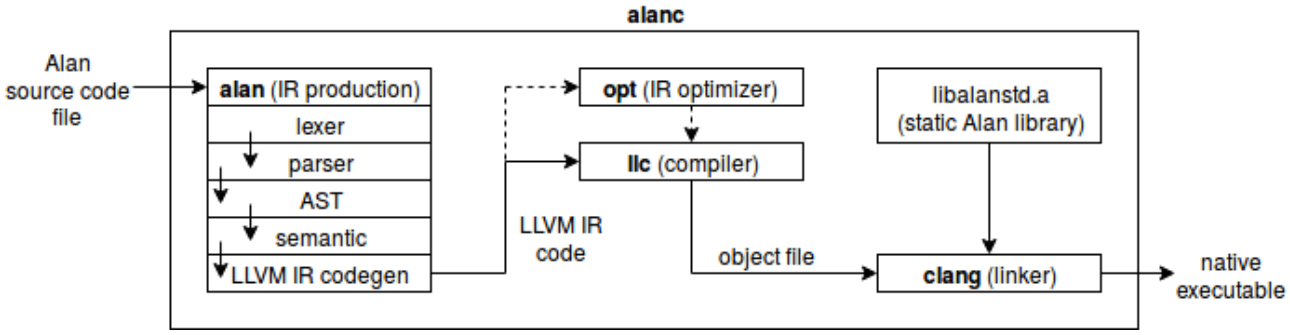
Οι τεχνολογίες που χρησιμοποιήθηκαν σε κάθε τμήμα του μεταγλωττιστή είναι οι εξής:

- Για τον λεκτικό αναλυτή χρησιμοποιήθηκε το εργαλείο flex
- Για τον συντακτικό αναλυτή χρησιμοποιήθηκε το εργαλείο bison
- Ο σημασιολογικός αναλυτής και η κατασκευή του AST γράφτηκαν σε C++
- Η παραγωγή ενδιάμεσου κώδικα έγινε σε C++ με χρήση των αντίστοιχων LLVM bindings (ο μεταγλωττιστής ελέγχθηκε με τις εκδόσεις 3.8 και 6.0 του LLVM)
- Η βελτιστοποίηση του ενδιάμεσου κώδικα, καθώς και η παραγωγή του τελικού κώδικα και η βελτιστοποίησή του έγιναν με χρήση κάποιων command line utilities που προσφέρονται από το LLVM (αναλυτικότερη αναφορά σε αυτά θα γίνει στη συνέχεια)
- Η standard βιβλιοθήκη της Alan γράφτηκε από την αρχή σε C, ακολουθώντας όσο πιο πιστά γινόταν αφενός την περιγραφή των συναρτήσεων στην εκφώνηση της εργασίας, αφετέρου τη συμπεριφορά των αντίστοιχων συναρτήσεων της C. Αυτό επιλέξαμε να το κάνουμε στα πλαίσια της προσπάθειας για μία πιο cross-platform προσέγγιση
- Το τελικό script που αλληλεπιδρά με τον χρήστη και συνενώνει όλα τα παραπάνω γράφτηκε σε Python 3.7. Η Python προτιμήθηκε ως “γλώσσα συνένωσης” (glue language) έναντι της αρχικής μας υλοποίησης σε bash με στόχο το τελικό script να είναι όσο το δυνατόν πιο εύχρηστο, ευανάγνωστο, επεκτάσιμο και cross-platform

1. Εποπτεία της συνολικής αρχιτεκτονικής του μεταγλωττιστή

Αρχικά, θα αναφερθούμε στην συνολική αρχιτεκτονική του μεταγλωττιστή, ώστε να είναι ξεκάθαρα η θέση και ο ρόλος του κάθε τμήματος το οποίο θα αναλύουμε στη συνέχεια της αναφοράς.

Μία αφαιρετική οπτική της αρχιτεκτονικής του μεταγλωττιστή που υλοποιήσαμε φαίνεται στο διάγραμμα που ακολουθεί. Πριν προχωρήσουμε στην επιμέρους ανάλυση των τμημάτων του μεταγλωττιστή, θα αναφερθούμε συνοπτικά στον τρόπο που αυτά αλληλεπιδρούν και ποιές είναι οι αρμοδιότητες του κάθε τμήματος, όπως αυτές προκύπτουν από τον σχεδιασμό μας.



Η καρδιά του μεταγλωττιστή μας είναι το πρόγραμμα **alan**. Αυτό είναι υπεύθυνο για την ανάγνωση του πηγαίου κώδικα Alan, την επεξεργασία του και, εν τέλει, την παραγωγή του αντίστοιχου ενδιάμεσου κώδικα, σε LLVM IR. Για να το επιτύχει αυτό, αποτελείται μεταξύ άλλων από τον λεκτικό, τον συντακτικό και τον σημασιολογικό αναλυτή. Ο λεκτικός αναλυτής αναγνωρίζει και προωθεί στο pipeline του alan τα lexemes, τα οποία στη συνέχεια ο συντακτικός αναλυτής χρησιμοποιεί για να κατασκευάσει το abstract syntax tree (AST) που αντιπροσωπεύει το αρχικό πρόγραμμα. Αυτό επιτυγχάνεται, προφανώς, στην περίπτωση που δεν εντοπιστούν ούτε λεκτικά ούτε συντακτικά λάθη. Στη συνέχεια, ο σημασιολογικός αναλυτής ελέγχει ολόκληρο το AST για σημασιολογικά λάθη. Εάν τέτοια δεν εντοπιστούν, ένα ακόμα τελευταίο πέρασμα του AST λαμβάνει χώρα, κατά το οποίο παράγεται ενδιάμεσος κώδικας στη γλώσσα του LLVM (LLVM IR), με χρήση των κατάλληλων C++ bindings.

Σε αυτό το σημείο, έχει χαθεί οτιδήποτε σχετίζεται με την Alan: έχουμε αυτή τη στιγμή στα χέρια μας το ισοδύναμο του αρχικού προγράμματος σε LLVM IR, το οποίο σημαίνει πως τώρα μπορούμε να το αντιμετωπίσουμε ως τέτοιο, απολύτως ανεξάρτητα από την αρχική (πηγαία) γλώσσα. Συνεπώς, η πορεία από εδώ και πέρα καθορίζεται σχεδόν αποκλειστικά από το LLVM.

Πιο συγκεκριμένα, εάν ο χρήστης έχει ζητήσει να γίνουν βελτιστοποιήσεις, ο ενδιάμεσος κώδικας περνάει από το **opt**, ένα command line utility του LLVM το οποίο δέχεται ως είσοδο LLVM IR κώδικα και παράγει (πιθανώς) βελτιστοποιημένο LLVM IR κώδικα. Δεν θελήσαμε να κάνουμε υποχρεωτικό το στάδιο της βελτιστοποίησης του ενδιάμεσου κώδικα, ώστε να μπορούμε να πειραματιστούμε περισσότερο με τις διαφορές που προκύπτουν (με ή χωρίς βελτιστοποίηση) στη συνέχεια του pipeline, αλλά και για να μπορεί ο χρήστης να δει τον ενδιάμεσο κώδικα ακριβώς όπως τον παράγουμε εμείς μέσω των C++ bindings, χωρίς καμία αλλαγή (με χρήση της σημαίας -i, όπως περιγράφεται στην εκφώνηση της εργασίας).

Στη συνέχεια, το **llc**, ο compiler του LLVM, αναλαμβάνει να μετατρέψει τον ενδιάμεσο κώδικα σε object file, να το μεταγλωττίσει, δηλαδή, στη native assembly.

Τέλος, ο **clang** αναλαμβάνει να συνδέσει (link) το object file που έχει παραχθεί με τη standard βιβλιοθήκη της Alan (libalanstd.a), παράγοντας έτσι το τελικό native εκτελέσιμο πρόγραμμα.

2. Λεκτικός αναλυτής (lexer)

Αρχικά, θα

3. Συντακτικός αναλυτής (parser) και κατασκευή abstract syntax tree (AST)

Αρχικά, θα

4. Σημασιολογικός αναλυτής

Αρχικά, θα

5. Παραγωγή ενδιάμεσου κώδικα και βελτιστοποίηση

Αρχικά, θα

6. Παραγωγή τελικού κώδικα και βελτιστοποίησησ

Αρχικά, θα

7. alanc – ο τελικός μεταγλωττιστής

Αρχικά, θα