Τελική αναφορά εργασίας Μεταφραστών με θέμα :”Ανάπτυξη Μεταγλωττιστή για τη Γλώσσα GREEK++ με Παραγωγή Ενδιάμεσου Κώδικα, Πίνακα Συμβόλων, αρχείο σε γλώσσα C και Τελική Μετάφραση σε Assembly”

ΤΣΑΚΙΡΗΣ ΔΗΜΗΤΡΙΟΣ Α.Μ. 5371

Περιεχόμενα

Κεφάλαιο 1 - Λεξική και Συντακτική Ανάλυση

Λεξική Ανάλυση

Η λεξική ανάλυση (γνωστή και ως *lexer*) αποτελεί το πρώτο στάδιο μεταγλώττισης και έχει ως στόχο την ανάγνωση της ακολουθίας χαρακτήρων του προγράμματος εισόδου και τη μετατροπή τους σε ακολουθία από tokens, δηλαδή λεξιλογικές μονάδες με σημασιολογική αξία, όπως μεταβλητές, αριθμοί, τελεστές, λέξεις-κλειδιά και σύμβολα ομαδοποίησης.

Σκοπός και ρόλος του lexer

Η βασική λειτουργία του lexer είναι:

- να **διαβάζει τον κώδικα γραμμή-γραμμή**,
- να **αναγνωρίζει** (μέσω κανονικών εκφράσεων ή string matching) το είδος κάθε λέξης (π.χ. αριθμός, αναγνωριστικό, λέξη-κλειδί),

- να **αγνοεί ασήμαντους χαρακτήρες** όπως κενά και σχόλια, και
- να **παράγει μια λίστα tokens** που περνάει στον parser.

Υλοποίηση και Σχεδιαστικές Επιλογές

Η λειτουργία της λεξικής ανάλυσης έχει υλοποιηθεί μέσω της συνάρτησης tokenize(code), η οποία δέχεται ως είσοδο μια λίστα γραμμών προγράμματος και επιστρέφει λίστα tokens.

Κατηγορίες tokens

Η ταξινόμηση των tokens γίνεται με βάση τους παρακάτω τύπους:

Τύπος Token	Περιγραφή
KEYWORD	Λέξεις-κλειδιά όπως δήλωση, γράψε, αρχή_προγράμματος

Τύπος Token	Περιγραφή
IDENTIFIER	Ονόματα μεταβλητών ή συναρτήσεων
NUMBER	Ακέραιοι αριθμοί
OPERATOR	Τελεστές +, -, *, /, :=
COMPARISON_OPERATOR	Συγκριτικοί τελεστές <=, =, <>
GROUPING_SYMBOLS	Παρενθέσεις και άγκιστρα: ( ) [ ]
SEPARATOR	Κόμμα ,
COMMENT	Γραμμές με //, οι οποίες αγνοούνται

Παράδειγμα ελέγχου tokens

Κάθε γραμμή του προγράμματος αναλύεται λέξη-προς-λέξη με τον παρακάτω μηχανισμό:

```
if word in greek_keywords:
    tokens.append(("KEYWORD", word))
```

```
elif word in operators:
    tokens.append(("OPERATOR", word))
elif word.isdigit():
    tokens.append(("NUMBER", word))
```

Κάθε λέξη ταιριάζει με ένα pattern, και η κατηγορία της καθορίζεται άμεσα.

Αναγνωριστικά και μεταβλητές

Για την αναγνώριση αναγνωριστικών (ονομάτων μεταβλητών ή συναρτήσεων), ακολουθείται απλός έλεγχος:

```
elif word.isidentifier():
    tokens.append(("IDENTIFIER", word))
```

Αυτός ο έλεγχος εξασφαλίζει ότι το αναγνωριστικό ξεκινάει με γράμμα και περιέχει μόνο γράμματα, αριθμούς και underscores (_). Η χρήση της isidentifier() από την Python απλοποιεί τον έλεγχο έγκυρων ονομάτων.

Διαχείριση Σχολίων

Τα σχόλια υποστηρίζονται με το σύμβολο //. Η λογική είναι απλή:

```
if "//" in line:
    line = line[:line.index("//")]
```

Με αυτόν τον τρόπο αγνοείται οποιοδήποτε περιεχόμενο μετά το //, χωρίς να επηρεάζει την ανάλυση.

Χειρισμός λαθών

Για κάθε άγνωστο σύμβολο ή λέξη που δεν αναγνωρίζεται, ο lexer επιστρέφει σφάλμα και σταματά την εκτέλεση:

```
else:
    raise Exception(f"Άγνωστο token: {word}")
```

Αυτό καθιστά τη διαδικασία ανθεκτική σε συντακτικά λάθη και προστατεύει από αδιευκρίνιστα inputs.

Τεκμηρίωση σχεδιαστικών αποφάσεων

- Επιλέχθηκε **ρητή αντιστοίχιση λέξεων**, για να είναι πιο ευανάγνωστο και ευέλικτο.
- Όλα τα tokens καταγράφονται ως **(τύπος, τιμή)** και μεταφέρονται αυτούσια στον parser.
- Το λεξικό greek_keywords απομονώνει τις λέξεις-κλειδιά της γλώσσας GREEK++ για σαφή διαχωρισμό από τα αναγνωριστικά.

Συντακτική Ανάλυση

Η συντακτική ανάλυση αποτελεί το δεύτερο στάδιο της μεταγλώττισης, το οποίο λαμβάνει την έξοδο του λεξικού αναλυτή (tokens) και ελέγχει κατά πόσο η ακολουθία αυτών των λέξεων αποτελεί συντακτικά ορθή πρόταση

σύμφωνα με τη γραμματική της γλώσσας **GREEK++**.

Για την υλοποίηση του parser και για κάθε παραγωγή της γραμματικής, υλοποιήθηκε μία αντίστοιχη Python συνάρτηση της μορφής parse_<κανόνας>. Οι συναρτήσεις αυτές διαβάζουν το τρέχον token, το συγκρίνουν με το αναμενόμενο συντακτικό μοτίβο και προχωρούν σε επόμενα tokens με χρήση της advance().

Η αναγνώριση των δομών ελέγχου, δηλώσεων, και εκφράσεων συνδυάζεται με κλήσεις στην ενότητα παραγωγής ενδιάμεσου κώδικα, ώστε να παράγεται ταυτόχρονα ο ενδιάμεσος κώδικας (intermediate code) καθώς γίνεται η ανάλυση.

Δομή Parser και Οργάνωση

- Ο parser οργανώνεται με βάση τις συντακτικές κατηγορίες: πρόγραμμα, δήλωση, ανάθεση, βρόχος, συνθήκη, συνάρτηση.

- Η κεντρική συνάρτηση parse_program() χειρίζεται το βασικό σώμα του προγράμματος.
- Η parse_expression() και parse_condition() χειρίζονται αριθμητικές και λογικές εκφράσεις.
- Η parse_if_stat() υλοποιεί τη δομή ελέγχου εάν ... τότε ... αλλιώς ... εάν_τέλος, παράγοντας εντολές άλματος με κατάλληλες ετικέτες (labels).

Διαχείριση Συνθηκών

Η ανάλυση των συνθηκών υλοποιείται με τέτοιο τρόπο ώστε να μπορεί να υποστηρίξει:

- **Απλές συγκρίσεις** (π.χ. x < 5)
- **Σύνθετες λογικές συνθήκες** (π.χ. [x > 1 και y < 10] ή z = 3) μέσω της παραγωγής κατάλληλων **true/false lists** και τεχνικής **backpatching**.

Επικοινωνία με τα υπόλοιπα μέρη

- Ο parser εισάγει κάθε νέα μεταβλητή ή συνάρτηση στον **Πίνακα Συμβόλων**.
- Οι εντολές που συναντά μεταφράζονται σε τριάδες μέσω του emit(op, arg1, arg2, result)
- Ο parser διαχειρίζεται εσωτερικά counters για labels, προσωρινές μεταβλητές, και επίπεδα εμφωλευμένων μπλοκ (scoping).

Κεφάλαιο 2 - Παραγωγή Ενδιάμεσου Κώδικα

Η παραγωγή ενδιάμεσου κώδικα (intermediate code) αποτελεί το στάδιο όπου οι συντακτικά έγκυρες προτάσεις του προγράμματος μετατρέπονται σε μια ενδιάμεση αναπαράσταση — συνήθως υπό τη μορφή τετράδων (quads) ή τριάδων.

Στο παρόν έργο, υλοποιείται ένα ενδιάμεσο επίπεδο με τετράδες της μορφής:
<label>: op, arg1, arg2, result

Στόχος Ενδιάμεσου Κώδικα

Ο ρόλος του ενδιάμεσου κώδικα είναι:

- Να αποτυπώσει πιστά τη λογική του προγράμματος χωρίς ακόμα να εξαρτάται από την τελική αρχιτεκτονική (Assembly),
- Να επιτρέψει βελτιστοποιήσεις,
- Να λειτουργήσει ως βάση για τελική παραγωγή RISC-V assembly.

Δομή Υλοποίησης

Η βασική κλάση που διαχειρίζεται τον ενδιάμεσο κώδικα είναι η IntermediateCode, με κύριες μεθόδους:

- emit(op, arg1, arg2, result): Δημιουργεί νέα τετράδα και την αποθηκεύει σε λίστα.

- newtemp(): Δημιουργεί νέες προσωρινές μεταβλητές τύπου t@1, t@2 για ενδιάμεσες εκφράσεις.
- new_label(): Δημιουργεί μοναδικές ετικέτες άλματος (L1, L2, ...).

Αλληλεπίδραση με Parser και Παραγωγή Κώδικα

Η παραγωγή των τετράδων δεν είναι αυτόνομη, αλλά καλείται απευθείας μέσα από τις συναρτήσεις του parser, καθώς αυτός αναγνωρίζει δομές της γλώσσας. Όταν, για παράδειγμα, αναγνωριστεί μία ανάθεση ή αριθμητική έκφραση, καλείται η emit() για να δημιουργηθεί το αντίστοιχο quad. Αυτό εξασφαλίζει συγχρονισμό μεταξύ συντακτικής ανάλυσης και δημιουργίας ενδιάμεσης αναπαράστασης.

Οι ετικέτες (labels) παράγονται κατά την αναγνώριση δομών ελέγχου ροής όπως if,

while και repeat, μέσω της new_label(), και χρησιμοποιούνται για την παραγωγή εντολών άλματος και τη σωστή σύνδεση των τμημάτων του κώδικα. Η δημιουργία τους ενσωματώνεται στον parser ώστε κάθε συντακτική δομή να δημιουργεί και τις αντίστοιχες οδηγίες ελέγχου ροής.

Διαχείριση Άλματος και Συνθηκών

Για τις δομές ελέγχου (if, while, repeat):

- Χρησιμοποιούνται εντολές jump, blt, bgt, beq, bne για τον έλεγχο ροής.
- Οι συνθήκες αναλύονται και μεταφράζονται σε relop τετράδες.
- Εάν πρόκειται για σύνθετες συνθήκες, χρησιμοποιείται τεχνική **backpatching** με true_list και false_list από τις λογικές συνιστώσες.

Παραδείγματα Εντολών

Οι βασικές εντολές που χρησιμοποιούνται στον ενδιάμεσο κώδικα είναι:

- Ανάθεση: :=
- Αριθμητικές πράξεις: +, -, *, /
- Συγκρίσεις: <, <=, =, <>
- Άλματα: jump, call, retv
- Είσοδος/έξοδος: in, out

Κεφάλαιο 3 - Πίνακας Συμβόλων

Σκοπός του Πίνακα Συμβόλων

Ο πίνακας συμβόλων χρησιμοποιείται για:

- Ανίχνευση δηλωμένων και μη δηλωμένων αναγνωριστικών,
- Αποθήκευση τύπων, θέσεων μνήμης (offsets), και scope levels,
- Διαχείριση επιπέδων εμφώλευσης (scoping) σε συναρτήσεις και μπλοκ.

Δομή Υλοποίησης

Η υλοποίηση βασίστηκε σε ένα λεξικό με λίστες ανά επίπεδο εμφώλευσης (scopes). Υπάρχει υποστήριξη για:

- Ανώτατο επίπεδο (Main),
- Επίπεδα για κάθε συνάρτηση (π.χ. επίπεδο 1 για την πρώτη, επίπεδο 2 για συναρτήσεις μέσα σε συνάρτηση κλπ),
- Υποστήριξη διαφορετικών ειδών: variable, parameter, function, temp.

Η κάθε εγγραφή έχει:

- name: το αναγνωριστικό,
- offset: η σχετική του θέση στο ενεργό activation record (stack frame),
- kind: variable, parameter, function,
- parMode: cv/ref (αν είναι parameter),
- scope: επίπεδο εμφώλευσης.

Εισαγωγή και Αναζήτηση Συμβόλων

Κάθε φορά που ο parser συναντά νέα μεταβλητή, παράμετρο ή συνάρτηση, καλεί τη insert() ώστε να την προσθέσει στον πίνακα συμβόλων. Ο πίνακας έχει δομή ανά επίπεδο εμφώλευσης (scope_level), και η τρέχουσα τιμή του scope_level ελέγχει πού θα αποθηκευτεί η νέα εγγραφή.

Ο έλεγχος για το "σωστό" scope γίνεται με βάση το context: για παράδειγμα, όταν ξεκινά ένα νέο begin_block, αυξάνεται το scope_level ώστε οι επόμενες εγγραφές να ανήκουν στο εσωτερικό block ή συνάρτηση.

Πώς γίνεται η αναζήτηση (lookup) και fallback ανά scope:

Η lookup() υλοποιεί αναδρομική αναζήτηση: ελέγχει πρώτα το τοπικό scope και, αν δεν βρεθεί η εγγραφή, ανεβαίνει σταδιακά προς τα εξωτερικά scopes. Αυτό υλοποιεί το **lexical scoping** και επιτρέπει την πρόσβαση σε global μεταβλητές ή εξωτερικές παραμέτρους,

διατηρώντας ταυτόχρονα την ιδιωτικότητα εσωτερικών δηλώσεων.

Διαχείριση Offsets

Οι μεταβλητές παίρνουν offset ανά 4 bytes:

- Π.χ. α → 12, β → 16, ι → 20.
- Οι παράμετροι τοποθετούνται πριν την είσοδο στην συνάρτηση (offsets ≥ 0).
- Οι προσωρινές (temp) τοποθετούνται μετά το τελευταίο offset της συνάρτησης.

Αντιμετώπιση Λαθών

Στην υλοποίηση, όταν γίνεται προσπάθεια πρόσβασης σε μη διαθέσιμο επίπεδο (π.χ. scope 2 ενώ το τρέχον είναι 0), εμφανίζεται προειδοποίηση:

Σφάλμα: Δεν υπάρχει επίπεδο 2

Αυτό χρησιμεύει στον εντοπισμό λαθών όπως η χρήση παραμέτρων εκτός συνάρτησης ή λανθασμένο nesting.

Κεφάλαιο 4 - Παραγωγή Τελικού Κώδικα (Assembly)

Το τελευταίο στάδιο του μεταγλωττιστή αφορά τη μετάφραση του ενδιάμεσου κώδικα σε πραγματικό κώδικα Assembly — συγκεκριμένα, για την αρχιτεκτονική **RISC-V**.

Μετατροπή Τετράδων σε Assembly

Ο ενδιάμεσος κώδικας μεταφέρεται σε κώδικα Assembly μέσω της μεθόδου generate_final_code(symtab). Κάθε τετράδα μεταφράζεται γραμμή προς γραμμή, βάσει του operator (+, :=, jump, call, κ.λπ.).

Υποστήριξη των Operators

Η παραγόμενη Assembly υποστηρίζει:

- Αριθμητικές πράξεις: add, sub, mul, div
- Αντιγραφές: li, lw, sw
- Συγκρίσεις με άλματα: beq, bne, blt, bge
- Άμεσο και έμμεσο addressing για παραμέτρους cv, ref
- Άλματα σε συναρτήσεις με jal, jr
- Συστήματα I/O μέσω ecall για out και halt

Εντολές διαχείρισης στοίβας

Κάθε συνάρτηση ξεκινά με:

sw ra, 0(sp)

και τελειώνει με:

lw ra, 0(sp)

jr ra

δηλαδή διασφαλίζεται η επιστροφή στην καλούσα συνάρτηση.

Σωστή Απόδοση των Ετικετών

Οι ετικέτες που προέρχονται από τα quads (π.χ. L13) γίνονται L13: στο τελικό .asm αρχείο. Αυτό επιτρέπει τον καθαρό εντοπισμό των σημείων άλματος.

Διαχείριση μεταβλητών με offsets — γιατί -12(sp) για το α

Κάθε μεταβλητή τοποθετείται σε συγκεκριμένο offset στο **stack frame** της συνάρτησης ή του main. Το -12(sp) αντιστοιχεί στη μεταβλητή με offset 12, διότι στην αρχιτεκτονική που χρησιμοποιείται (RISC-V) η κορυφή του stack βρίσκεται σε αυξανόμενες διευθύνσεις προς τα **κάτω**, και τα τοπικά δεδομένα τοποθετούνται σε **αρνητικά offsets** από το sp.

Τα offsets είναι **πολλαπλάσια των 4 bytes** και αποδίδονται σειριακά κατά την εισαγωγή στο symbol table. Αυτή η επιλογή επιτρέπει

ευθεία μετάφραση σε lw και sw εντολές **χωρίς επιπλέον mapping**.

Πώς αποθηκεύονται και φορτώνονται οι τιμές

Η φόρτωση και αποθήκευση μεταβλητών γίνεται μέσω των βοηθητικών συναρτήσεων loadvr() και storerv().

- Η loadvr() φορτώνει την τιμή μιας μεταβλητής σε register, ανάλογα με τον τύπο της (local, global, παράμετρος, προσωρινή).
- Η storerv() εκτελεί την αντίστροφη διαδικασία — παίρνει τιμή από register και την αποθηκεύει στη σωστή θέση στο stack.

Και οι δύο συναρτήσεις ενσωματώνουν τη λογική για **call-by-value** και **call-by-reference**, καθώς και για προσπέλαση **global** μεταβλητών.

Πώς δουλεύει η generate_final_code() — λογική υλοποίησης

Η generate_final_code() διατρέχει την ακολουθία των τετράδων του ενδιάμεσου κώδικα και για κάθε μία παράγει την αντίστοιχη εντολή σε RISC-V assembly. Χρησιμοποιεί ένα mapping από operator σε opcode και διαχειρίζεται:

- labels,
- εντολές ελέγχου ροής,
- αριθμητικές πράξεις,
- είσοδο/έξοδο (ecall),
- καθώς και par, call, retv.

Διατηρεί δείκτη par_index για να κατατάξει σωστά τους παραμέτρους στα αντίστοιχα offsets κατά τη par φάση. Η οργάνωση είναι γραμμική και γίνεται **ενιαία διέλευση** του ενδιάμεσου κώδικα.

Επεξήγηση υποβοηθητικών συναρτήσεων loadvr, storerv, gnlvcode

- Η loadvr(v, r, output_lines) χρησιμοποιείται για να φορτώσει την τιμή του v στον καταχωρητή r. Λαμβάνει υπόψη εάν το v είναι μεταβλητή, temp ή παράμετρος και αν ανήκει στο ίδιο scope ή σε εξωτερικό. Υλοποιεί την απαραίτητη προσπέλαση (άμεση ή έμμεση).
- Η storerv(r, v, output_lines) είναι η αντίστροφη: αποθηκεύει την τιμή του register r πίσω στο v, χρησιμοποιώντας τη σωστή μέθοδο πρόσβασης (direct/indirect).
- Η gnlvcode(v, output_lines) χρησιμοποιείται όταν το v ανήκει σε εξωτερικό scope. Υπολογίζει τη διεύθυνση της μεταβλητής μέσω του **dynamic link** και γράφει στο t0 την τελική

διεύθυνση, η οποία χρησιμοποιείται από loadvr ή storerv για έμμεση προσπέλαση.

Κεφάλαιο 5 - Δυσκολίες

- Στο lexer και parser, δεν αντιμετώπισα ιδιαίτερες δυσκολίες, καθώς οι δομές ήταν σχετικά απλές και υλοποιήθηκαν με βάση τη γραμματική.
- Οι δυσκολίες εμφανίστηκαν έντονα στην παραγωγή ενδιάμεσου κώδικα, ιδιαίτερα με το αρχείο test4.gr, το οποίο περιλάμβανε εμφωλευμένες συναρτήσεις και πολύπλοκες συνθήκες.
- Στο τελικό στάδιο της παραγωγής Assembly, η υλοποίηση λειτούργησε χωρίς πρόβλημα στα βασικά προγράμματα. Όμως, δυσκολίες παρουσιάστηκαν με την υποστήριξη πολλαπλών συναρτήσεων και των αλμάτων (jumps).