# Μεταφραστές

Όνομα: Παύλος Ορφανίδης ΑΜ: 4134

22 Μαΐου 2025

# Περιεχόμενα

# 1 Εισαγωγή

## 1.1 Δομή

Η greek++ είναι μια εκπαιδευτική γλώσσα προγραμματισμού με περιορισμένες προγραμματιστικές δυνατότητες, αλλά πλούσια προγραμματιστικά στοιχεία. Παρότι απλή, περιέχει όλες σχεδόν τις βασικές δομές που συναντούμε σε άλλες γλώσσες προγραμματισμού:

- Συναρτήσεις και διαδικασίες
- Μετάδοση παραμέτρων με αναφορά και τιμή
- Δομές απόφασης και βρόχων
- Αναδρομικές κλήσεις

## 1.2 Αλφάβητο της Γλώσσας

Το αλφάβητο της greek++ αποτελείται από:

- Γράμματα: μικρά και κεφαλαία της ελληνικής («Α»...«Ω», «α»...«ω») και αγγλικής («A»...«Z», «a»...«z») αλφαβήτου
- Αριθμητικά ψηφία: «0»...«9»
- Σύμβολα αριθμητικών πράξεων: «+», «-», «*», «/»
- Τελεστές συσχέτισης: «<», «>», «=», «<=», «>=», «<>»
- Τελεστής ανάθεσης: «:=»
- Διαχωριστές: «;», «,»
- Σύμβολα ομαδοποίησης: «(», «)», «[», «]»
- Σύμβολα σχολίων: «», «»
- Σύμβολο περάσματος παραμέτρων με αναφορά: «

## 1.3 Δεσμευμένες Λέξεις

Η greek++ περιλαμβάνει τις ακόλουθες δεσμευμένες λέξεις:

Πίνακας 1: Δεσμευμένες λέξεις της γλώσσας greek++

| | | | |
|---|---|---|---|
| **πρόγραμμα** | **δήλωση** | **εάν** | **τότε** |
| **αλλιώς** | **εάν_τέλος** | **επανάλαβε** | **μέχρι** |
| **όσο** | **όσο_τέλος** | **για** | **έως** |
| **με_βήμα** | **για_τέλος** | **διάβασε** | **γράψε** |
| **συνάρτηση** | **διαδικασία** | **διαπροσωπεία** | **είσοδος** |
| **έξοδος** | **αρχή_συνάρτησης** | **τέλος_συνάρτησης** | **αρχή_διαδικασίας** |
| **τέλος_διαδικασίας** | **αρχή_προγράμματος** | **τέλος_προγράμματος** | **ή** |
| **και** | **όχι** | **εκτέλεσε** | |

## 1.4 Κανόνες Εγκυρότητας Λεκτικών Μονάδων

Οι κανόνες εγκυρότητας των λεκτικών μονάδων της greek++ είναι οι εξής:

- Αναγνωριστικά (identifiers): Ακολουθίες γραμμάτων, ψηφίων και κάτω παύλας, αρχίζοντας πάντα από γράμμα. Το μέγιστο μήκος είναι 30 χαρακτήρες.
- Αριθμητικές σταθερές: Προαιρετικό πρόσημο ακολουθούμενο από ψηφία.
- Λευκοί χαρακτήρες και σχόλια: Αγνοούνται και μπορούν να χρησιμοποιηθούν ελεύθερα, αρκεί να μην βρίσκονται μέσα σε δεσμευμένες λέξεις, αναγνωριστικά ή αριθμητικές σταθερές. Τα σχόλια περικλείονται σε άγκιστρα { }.

## 1.5 Τύποι Δεδομένων και Δηλώσεις

Η greek++ υποστηρίζει μόνο ακέραιους αριθμούς. Η δήλωση μεταβλητών γίνεται με την εντολή δήλωση ακολουθούμενη από λίστα αναγνωριστικών χωρισμένων με κόμματα. Παράδειγμα:
δήλωση
i, j
k

## 1.6 Τελεστές και Εκφράσεις

Η προτεραιότητα των τελεστών (από μεγαλύτερη σε μικρότερη) είναι:

1. Μοναδικοί «όχι»
2. Πολλαπλασιαστικοί «*», «/»
3. Μοναδικοί προσθετικοί «+», «-»
4. Δυαδικοί προσθετικοί «+», «-»
5. Σχεσιακοί «=», «<», «>», «<>», «<=», «>=»
6. Λογικοί «και», «ή»

**Εντολές** Η greek++ υποστηρίζει τις ακόλουθες εντολές:

**Εντολή Εκχώρησης** Id := expression

Χρησιμοποιείται για την ανάθεση τιμής μιας μεταβλητής, σταθεράς ή έκφρασης σε μια μεταβλητή.

**Εντολή Απόφασης**

```
εάν condition τότε
    ...
[αλλιώς
    ...]
εάν_τέλος
```

Το τμήμα 'αλλιώς' είναι προαιρετικό.

**Εντολές Επανάληψης**

**Επανάλαβε-μέχρι**

```
επανάλαβε
    ...
μέχρι condition
```

Η εντολή εκτελεί τις εσωτερικές εντολές μέχρι η συνθήκη να γίνει αληθής.

**Όσο**

```
όσο condition επανάλαβε
    ...
όσο_τέλος
```

Η εντολή αποτιμά τη συνθήκη και, εφόσον είναι αληθής, εκτελεί τις εσωτερικές εντολές επαναληπτικά μέχρι η συνθήκη να γίνει ψευδής.

**Για**

```
για id:=expr1 έως expr2 [ με_βήμα expr3 ] επανάλαβε
    ...
για_τέλος
```

Η μεταβλητή id αρχικοποιείται με την τιμή expr1, και σε κάθε επανάληψη αυξάνεται κατά expr3 (ή κατά 1 αν το expr3 παραλείπεται) μέχρι να φτάσει την τιμή expr2. Αν το expr3 είναι θετικό, ο βρόχος τερματίζεται όταν το id γίνει μεγαλύτερο ίσο του expr2. Αν είναι αρνητικό, ο βρόχος τερματίζεται όταν το id γίνει μικρότερο ή ίσο του expr2. Αν είναι μηδέν, έχουμε ατέρμονα βρόχο.

Σημείωση: Δεν επιτρέπεται η μεταβολή της μεταβλητής ελέγχου id μέσα στο βρόχο.

**Εντολή Εισόδου**

```
διάβασε id
```

Διαβάζει μια τιμή από το πληκτρολόγιο και την αναθέτει στη μεταβλητή id.

**Εντολή Εξόδου**

```
γράψε expression
```

Αποτιμά την έκφραση και εμφανίζει το αποτέλεσμα στην οθόνη.

## 1.7 Υποπρογράμματα

Η greek++ υποστηρίζει συναρτήσεις και διαδικασίες:

### 1.7.1 Συναρτήσεις

```
συνάρτηση id (formal_parameters)
    formal_parameters
    declarations
    αρχή_συνάρτησης
        sequence of statements
    τέλος_συνάρτησης
```

### 1.7.2 Διαδικασίες

```
διαδικασία id (formal_parameters)
    formal_parameters
    declarations
    αρχή_διαδικασίας
        sequence of statements
    τέλος_διαδικασίας
```

Οι τυπικές παράμετροι δηλώνονται μέσα στην παρένθεση, αλλά οι τύποι τους καθώς και ο τρόπος μετάδοσης (τιμή ή αναφορά) δηλώνονται ξεχωριστά.

### 1.7.3 Μετάδοση Παραμέτρων

Η greek++ υποστηρίζει δύο τρόπους μετάδοσης παραμέτρων:

1. **Με σταθερή τιμή:** Αλλαγές στην τιμή της παραμέτρου δεν επιστρέφονται στον καλούντα.

2. **Με αναφορά:** Δηλώνεται με το σύμβολο `%`. Κάθε αλλαγή στην τιμή της παραμέτρου μεταφέρεται στο πρόγραμμα που κάλεσε το υποπρόγραμμα.

## 1.8 Δομή Προγράμματος

Η γενική δομή ενός προγράμματος greek++ είναι η εξής:

```
πρόγραμμα id
    declarations
    subprograms
    αρχή_προγράμματος
        sequence of statements
    τέλος_προγράμματος
```

## 1.9 Παράδειγμα Προγράμματος

Παρακάτω παρουσιάζεται ένα πλήρες παράδειγμα προγράμματος σε γλώσσα greek++:

```
πρόγραμμα τεστ
  δήλωση α,β
  δήλωση γ

  συνάρτηση αύξηση(α,β)
    διαπροσωπεία
    είσοδος α
    έξοδος β
    αρχή_συνάρτησης
      β := α + 1 ;
      αύξηση := α + 1  { δεν μπαίνει ερωτηματικό είναι τέλος block }
    τέλος_συνάρτησης

  διαδικασία τύπωσε_συν_1(χ)
    διαπροσωπεία
    είσοδος χ
    αρχή_διαδικασίας
      γράψε χ+1
    τέλος_διαδικασίας

  αρχή_προγράμματος
    α := 1 ;
    β := 2 + α * α / (2 - α - (2*α));
    γ := αύξηση(α,%β);
    για α:=1 έως 8 με_βήμα 2 επανάλαβε
      εκτέλεσε τύπωσε_συν_1(α)
    για_τέλος;
    β := 1;
    όσο β<10 επανάλαβε
      εάν β<>22 ή [β>=23 και β<=24] τότε
        β := β+1
      εάν_τέλος { όχι ερωτηματικό, είναι τέλος block }
    όσο_τέλος; { θέλει ερωτηματικό χωρίζει εκτελέσιμες εντολές }
    διάβασε β;
```

```
    επανάλαβε
      β := β + 1
    μέχρι β<-100
  τέλος_προγράμματος
```

### 1.10 Γραμματική της Γλώσσας greek++

Η γραμματική της greek++ στηρίζεται στη γραμματική τύπου LL(1), η οποία είναι μια γραμματική χωρίς συμφραζόμενα. Στις γραμματικές αυτού του τύπου, οι λεκτικές μονάδες αναγνωρίζονται από τα αριστερά προς τα δεξιά (L: left to right), επιλέγεται η αριστερότερη δυνατή παραγωγή (L: leftmost derivation) και όταν η γραμματική δεν ξέρει ποιον κανόνα να ακολουθήσει, κοιτάζει τον αμέσως επόμενο χαρακτήρα της λεκτικής μονάδας ((1): one look-ahead symbol).

### 1.11 Βασικοί Κανόνες Γραμματικής

Παρουσιάζονται ενδεικτικά ορισμένοι βασικοί κανόνες της γραμματικής:

```
program: 'πρόγραμμα' ID programblock;
programblock:declarations subprograms
    'αρχή_προγράμματος' sequence 'τέλος_προγράμματος';
declarations: ( 'δήλωση' varlist )*|;
varlist: ID ( ',' ID )*;
subprograms: ( func | proc )*
func:'συνάρτηση' ID '(' formalparlist')' funcblock;
```

## 2 Άνοιγμα και Επεξεργασία Αρχείων greek++

### 2.1 Εισαγωγή στην Επεξεργασία Αρχείων

Τα προγράμματα της γλώσσας greek++ αποθηκεύονται σε αρχεία με επέκταση `.gr`. Για να μεταγλωττιστεί ένα πρόγραμμα greek++, πρέπει πρώτα να ανοίξουμε και να αναλύσουμε το περιεχόμενο του αρχείου πηγαίου κώδικα. Η διαδικασία αυτή αποτελεί το αρχικό βήμα πριν από τη λεκτική και συντακτική ανάλυση.

### 2.2 Διαχείριση Γραμμής Εντολών

Ο μεταγλωττιστής της greek++ είναι σχεδιασμένος ως εφαρμογή γραμμής εντολών, επιτρέποντας στους χρήστες να καθορίσουν το αρχείο πηγαίου κώδικα που θα μεταγλωττιστεί και διάφορες επιλογές μέσω παραμέτρων. Χρησιμοποιούμε τη βιβλιοθήκη `argparse` της Python για τη διαχείριση των παραμέτρων της γραμμής εντολών.

```
# Create an argument parser to handle command-line arguments
parser = argparse.ArgumentParser(description='Process a source code file.
# Add a positional argument for the source code file to process
parser.add_argument('file', type=str, help='The source code file to proce
parser.add_argument('-d', '--debug', action='store_true', help='Enable de
# Parse the command-line arguments
args = parser.parse_args()
```

Σε αυτό το απόσπασμα κώδικα:

- Αρχικοποιούμε έναν επεξεργαστή παραμέτρων με μια περιγραφή της λειτουργίας του μεταγλωττιστή.
- Προσθέτουμε μια υποχρεωτική παράμετρο θέσης (`file`) που αντιπροσωπεύει το αρχείο πηγαίου κώδικα greek++ που θα μεταγλωττιστεί.
- Προσθέτουμε μια προαιρετική επιλογή (`-d` ή `--debug`) που επιτρέπει στο χρήστη να ενεργοποιήσει την έξοδο αποσφαλμάτωσης για λεπτομερέστερη πληροφόρηση κατά τη διαδικασία μεταγλώττισης.
- Αναλύουμε τις παραμέτρους της γραμμής εντολών και αποθηκεύουμε τα αποτελέσματα στη μεταβλητή `args`.

## 2.3 Άνοιγμα και Ανάγνωση Αρχείων greek++

Μετά την ανάλυση των παραμέτρων της γραμμής εντολών, προχωρούμε στο άνοιγμα και την ανάγνωση του καθορισμένου αρχείου πηγαίου κώδικα. Για αυτό τον σκοπό, χρησιμοποιούμε τις βασικές λειτουργίες διαχείρισης αρχείων της Python.

```
def compile_file(file, debug):
    file_extension = get_file_extension(file)
    # Perform lexical analysis on the provided source code file
    tokens = perform_lexical_analysis(file, debug)
    # Perform syntax analysis on the generated tokens
    tokens, ast = perform_syntax_analysis(tokens, debug)
    # Generate symbol table from the parsed AST
    symbol_table = get_symbol_table(ast.to_dict(), file.replace(file_exte
    # Generate intermediate code from the parsed AST and symbol table
    quads = get_intermediate_code(ast.to_dict(), file.replace(file_extens
    # Generate RISC-V assembly code from the intermediate code and symbol
    get_riscv_code(quads, file.replace(file_extension, '.asm'), symbol_ta
    return quads
```

Η συνάρτηση `compile_file` δέχεται ως παράμετρο το όνομα του αρχείου που θα ανοιχτεί και το αν πρέπει να τυπώσει bug logs και επιστρέφει το περιεχόμενό του ως συμβολοσειρά

## 2.4 Ολοκληρωμένο Παράδειγμα Χρήσης

Για να χρησιμοποιήσετε τον μεταγλωττιστή greek++ με ένα αρχείο πηγαίου κώδικα, εκτελέστε την ακόλουθη εντολή:

```
python compiler_4134.py my_program.gr
```

Για να ενεργοποιήσετε τη λειτουργία αποσφαλμάτωσης:

```
python compiler_4134.py my_program.gr --debug
```

## 2.5 Σύνδεση με τα Επόμενα Στάδια Μεταγλώττισης

Μετά το επιτυχές άνοιγμα και την ανάγνωση του αρχείου πηγαίου κώδικα, ο μεταγλωττιστής προχωρά στα επόμενα στάδια:

1. **Λεκτική Ανάλυση:** Ανάλυση του πηγαίου κώδικα για την αναγνώριση των λεκτικών μονάδων (tokens).
2. **Συντακτική Ανάλυση:** Ανάλυση της ακολουθίας των λεκτικών μονάδων για τον έλεγχο της συντακτικής ορθότητας.
3. **Παραγωγή Ενδιάμεσου Κώδικα:** Μετατροπή του πηγαίου κώδικα σε μια ενδιάμεση αναπαράσταση.
4. **Παραγωγή Τελικού Κώδικα:** Μετατροπή του ενδιάμεσου κώδικα σε εκτελέσιμο κώδικα μηχανής.

Η επιτυχής ολοκλήρωση κάθε σταδίου εξαρτάται από την επιτυχή ολοκλήρωση του προηγούμενου σταδίου, ξεκινώντας από το άνοιγμα και την ανάγνωση του αρχείου πηγαίου κώδικα.

# 3 Λεκτική Ανάλυση

## 3.1 Εισαγωγή στη Λεκτική Ανάλυση

Η λεκτική ανάλυση αποτελεί το πρώτο στάδιο της μεταγλώττισης της γλώσσας greek++. Σκοπός της είναι η αναγνώριση και παραγωγή των λεκτικών μονάδων (tokens) από το πηγαίο κείμενο του προγράμματος. Ο λεκτικός αναλυτής (lexical analyzer) διαβάζει το αρχείο του προγράμματος χαρακτήρα-χαρακτήρα και παράγει μια ακολουθία λεκτικών μονάδων που αποτελούν την είσοδο για τον συντακτικό αναλυτή.

Κάθε λεκτική μονάδα αποτελείται από:

1. Ένα αναγνωριστικό (token identifier)
2. Την λεκτική οντότητα (lexeme)
3. Τον αριθμό γραμμής στην οποία εμφανίζεται

## 3.2 Αυτόματο Πεπερασμένων Καταστάσεων

Η σχεδίαση του λεκτικού αναλυτή της greek++ στηρίζεται σε ένα αυτόματο πεπερασμένων καταστάσεων (finite state automaton). Το αυτόματο αυτό αποτελείται από:

- Μια αρχική κατάσταση (start)
- Ενδιάμεσες καταστάσεις όπως identifier_state, number_state, κ.ά.
- Τελικές καταστάσεις που αντιστοιχούν σε αναγνωρισμένες λεκτικές μονάδες ή σφάλματα

Το αυτόματο ξεκινά από την αρχική κατάσταση και με την είσοδο κάθε χαρακτήρα αλλάζει κατάσταση, σύμφωνα με τους κανόνες μετάβασης, μέχρι να φτάσει σε κάποια τελική κατάσταση. Οι μεταβάσεις καθορίζονται από τον τύπο του χαρακτήρα που διαβάζεται (γράμμα, ψηφίο, σύμβολο, κλπ).

## 3.3 Βασικές Λειτουργίες του Λεκτικού Αναλυτή

### 3.3.1 Αναγνώριση Αναγνωριστικών και Δεσμευμένων Λέξεων

Όταν διαβαστεί γράμμα ή κάτω παύλα, το αυτόματο μεταβαίνει στην κατάσταση identifier_state. Εκεί παραμένει όσο διαβάζονται γράμματα, ψηφία ή κάτω παύλες. Όταν διαβαστεί άλλος χαρακτήρας, ελέγχεται αν η λεκτική μονάδα που σχηματίστηκε είναι:

- Δεσμευμένη λέξη της greek++
- Έγκυρο αναγνωριστικό (μέγιστο μήκος 30 χαρακτήρες, πρώτος χαρακτήρας γράμμα)
- Μη έγκυρο αναγνωριστικό (π.χ. μήκος $> 30$ χαρακτήρες ή ξεκινά με κάτω παύλα)

### 3.3.2 Αναγνώριση Αριθμητικών Σταθερών

Όταν διαβαστεί ψηφίο, το αυτόματο μεταβαίνει στην κατάσταση number_state. Εκεί παραμένει όσο διαβάζονται ψηφία. Όταν διαβαστεί άλλος χαρακτήρας, ελέγχεται αν η αριθμητική σταθερά που σχηματίστηκε είναι έγκυρη (εντός του επιτρεπτού εύρους τιμών: $-2^{32} - 1$ έως $2^{32} - 1$).

### 3.3.3 Αναγνώριση Τελεστών και Συμβόλων

Η greek++ περιλαμβάνει μονοψήφιους τελεστές (π.χ. +, -) και πολυψήφιους τελεστές (π.χ. :=, <=). Για τους πολυψήφιους τελεστές, το αυτόματο χρησιμοποιεί ενδιάμεσες καταστάσεις (π.χ. assign_state, less_state) για να διαχειριστεί τους συνδυασμούς χαρακτήρων.

### 3.3.4 Χειρισμός Σχολίων

Τα σχόλια στη greek++ περικλείονται σε άγκιστρα {}. Όταν αναγνωριστεί το άνοιγμα ενός σχολίου, το αυτόματο αγνοεί όλους τους χαρακτήρες μέχρι να συναντήσει το κλείσιμο του σχολίου.

### 3.3.5 Χειρισμός Λευκών Χαρακτήρων

Οι λευκοί χαρακτήρες (κενά, tabs, αλλαγές γραμμής) αγνοούνται, εκτός αν βρίσκονται μέσα σε συμβολοσειρές ή λεκτικές μονάδες.

## 3.4 Χειρισμός Σφαλμάτων

Ο λεκτικός αναλυτής της greek++ μπορεί να εντοπίσει διάφορους τύπους λεκτικών σφαλμάτων, όπως:

- Μη έγκυρους χαρακτήρες
- Μη έγκυρα αναγνωριστικά (π.χ. αναγνωριστικά που ξεκινούν με κάτω παύλα ή έχουν μήκος $> 30$ χαρακτήρες)
- Μη έγκυρες αριθμητικές σταθερές (εκτός επιτρεπτού εύρους)
- Μη κλεισμένα σχόλια
- Χρήση μη επιτρεπτών συμβόλων

Όταν εντοπιστεί ένα λεκτικό σφάλμα, ο αναλυτής τυπώνει κατάλληλο διαγνωστικό μήνυμα με τον αριθμό γραμμής όπου εντοπίστηκε το σφάλμα και τερματίζει τη διαδικασία μεταγλώττισης.

## 3.5 Υλοποίηση του Λεκτικού Αναλυτή

Ο λεκτικός αναλυτής υλοποιείται ως κλάση `Lexer(file)` με βασική συνάρτηση την `Lexer.tokenize()` η οποία διαβάζει χαρακτήρες από το αρχείο εισόδου και επιστρέφει την αναγνωρισμένη λεκτική μονάδα, τον τύπο της και τη γραμμή όπου εμφανίζεται. Η συνάρτηση αυτή καλείται πριν από τον συντακτικό αναλυτή και επιστρέφει όλο το αρχείο εισόδου με την μορφή tokens. Αυτό γίνεται προκειμένου να απλοποιηθεί η διαδικασία του κώδικα καθώς μπορώ να ελέγξω πιο εύκολα τον κώδικα για σφάλματα.

Η υλοποίηση περιλαμβάνει:

- Πίνακες με τα έγκυρα σύμβολα, δεσμευμένες λέξεις, τελεστές κλπ. της greek++
- Συναρτήσεις βοηθητικές όπως η `collect_identifier()` που επιστρέφει token τα γράμματα σχηματίζουν δεσμευμένη λέξη η οποία πρέπει να θεωρηθεί identifier.
- Διαδικασίες χειρισμού των διαφόρων τύπων λεκτικών μονάδων και των πιθανών σφαλμάτων

# 4 Συντακτική Ανάλυση

## 4.1 Εισαγωγή στη Συντακτική Ανάλυση

Η συντακτική ανάλυση αποτελεί το δεύτερο στάδιο της μεταγλώττισης της γλώσσας greek++. Σκοπός της είναι να ελέγξει αν η ακολουθία των λεκτικών μονάδων που παρήγαγε ο λεκτικός αναλυτής ακολουθεί τους κανόνες της γραμματικής της γλώσσας. Ο συντακτικός αναλυτής (parser) δέχεται ως είσοδο τις λεκτικές μονάδες και παράγει ένα συντακτικό δέντρο ή/και ενδιάμεσο κώδικα.

## 4.2 Γραμματική LL(1)

Η συντακτική ανάλυση της greek++ βασίζεται σε γραμματική τύπου LL(1), η οποία είναι μια γραμματική χωρίς συμφραζόμενα. Τα χαρακτηριστικά της είναι:

- **L**: Left-to-right scanning (Ανάγνωση των λεκτικών μονάδων από αριστερά προς τα δεξιά)
- **L**: Leftmost derivation (Αριστερότερη παραγωγή)
- **1**: One-symbol lookahead (Χρήση ενός συμβόλου προεπισκόπησης)

Η γραμματική της greek++ ορίζεται με βάση κανόνες παραγωγής που περιγράφουν τη δομή των επιτρεπτών προγραμμάτων της γλώσσας.

## 4.3 Βασικοί Κανόνες Γραμματικής

Οι βασικοί κανόνες γραμματικής της greek++ περιλαμβάνουν:

```
program: 'πρόγραμμα' ID programblock;
programblock:declarations subprograms
    'αρχή_προγράμματος' sequence 'τέλος_προγράμματος';
declarations: ( 'δήλωση' varlist )*|;
varlist: ID ( ',' ID )*;
subprograms: ( func | proc )*
func:'συνάρτηση' ID '(' formalparlist')' funcblock;
```

## 4.4 Δομή Προγράμματος

Ένα πρόγραμμα greek++ αποτελείται από:

- Την επικεφαλίδα του προγράμματος (`πρόγραμμα` *id*)
- Το τμήμα δηλώσεων μεταβλητών
- Τον ορισμό υποπρογραμμάτων (συναρτήσεις και διαδικασίες)
- Το κύριο σώμα του προγράμματος που περικλείεται μεταξύ `αρχή_προγράμματος` και `τέλος_προγράμματος`

## 4.5 Υποπρογράμματα

Η greek++ υποστηρίζει δύο τύπους υποπρογραμμάτων:

- **Συναρτήσεις** που επιστρέφουν τιμή
- **Διαδικασίες** που δεν επιστρέφουν τιμή

Κάθε υποπρόγραμμα μπορεί να έχει τυπικές παραμέτρους και τοπικές μεταβλητές. Οι τοπικές συναρτήσεις πρέπει να φωλιάζονται εντός των κύριων συναρτήσεων.

## 4.6 Εντολές

Η συντακτική ανάλυση ελέγχει την ορθότητα των εντολών της greek++ που περιλαμβάνουν:

- Εντολές εκχώρησης (`id := expression`)
- Εντολές απόφασης (`εάν-τότε-αλλιώς`)
- Εντολές επανάληψης (`επανάλαβε-μέχρι`, `όσο`, `για`)
- Εντολές εισόδου/εξόδου (`διάβασε`, `γράψε`)
- Κλήσεις υποπρογραμμάτων

## 4.7 Εκφράσεις

Οι εκφράσεις στη greek++ μπορεί να είναι:

- Αριθμητικές (π.χ. `a + b * 2`)
- Λογικές (π.χ. `a < b και c > d`)
- Συνδυασμός των παραπάνω

Η συντακτική ανάλυση ελέγχει αν οι εκφράσεις ακολουθούν τους κανόνες προτεραιότητας των τελεστών και τους κανόνες σύνταξης της γλώσσας.

## 4.8 Αναδρομική Καθοδική Ανάλυση

Η συντακτική ανάλυση της greek++ υλοποιείται με τη μέθοδο της αναδρομικής καθοδικής ανάλυσης (recursive descent parsing). Για κάθε μη τερματικό σύμβολο της γραμματικής, υπάρχει μια συνάρτηση που υλοποιεί τον αντίστοιχο κανόνα παραγωγής. Οι συναρτήσεις αυτές καλούν η μία την άλλη ακολουθώντας τη δομή της γραμματικής.

## 4.9 Χειρισμός Σφαλμάτων

Ο συντακτικός αναλυτής της greek++ μπορεί να εντοπίσει διάφορους τύπους συντακτικών σφαλμάτων, όπως:

- Έλλειψη αναμενόμενων συμβόλων (π.χ. παρένθεση που δεν κλείνει)
- Έλλειψη δεσμευμένων λέξεων (π.χ. έλλειψη της λέξης `τότε` μετά από `εάν`)
- Λανθασμένη σειρά λεκτικών μονάδων
- Μη έγκυρες δομές προγράμματος

Όταν εντοπιστεί ένα συντακτικό σφάλμα, ο αναλυτής τυπώνει κατάλληλο διαγνωστικό μήνυμα με τον αριθμό γραμμής όπου εντοπίστηκε το σφάλμα και τερματίζει τη διαδικασία μεταγλώττισης.

## 4.10 Υλοποίηση του Συντακτικού Αναλυτή

Ο συντακτικός αναλυτής υλοποιείται ως κλάση `Syntax()` οποία καλειται από την συνάρτηση `perform_syntax_analysis()` η οποία παίρνει σαν όρισμα τον πίνακα με τις λεκτικές μονάδες του προγράμματος. Η συνάρτηση αυτή ελέγχει τη συντακτική ορθότητα του προγράμματος και, αν δεν βρεθούν σφάλματα, προχωρά στην παραγωγή ενδιάμεσου κώδικα.

Η υλοποίηση περιλαμβάνει:

- Επιμέρους συναρτήσεις για κάθε μη τερματικό σύμβολο της γραμματικής
- Διαδικασίες χειρισμού των διαφόρων συντακτικών δομών
- Μηχανισμούς ανάκαμψης από σφάλματα (error recovery)
- Ταυτόχρονη παραγωγή ενδιάμεσου κώδικα κατά τη διάρκεια της συντακτικής ανάλυσης

## 4.11 Σύνδεση με Επόμενα Στάδια Μεταγλώττισης

Μετά την επιτυχή ολοκλήρωση της συντακτικής ανάλυσης, ο μεταγλωττιστής της greek++ προχωρά στα επόμενα στάδια:

- Παραγωγή ενδιάμεσου κώδικα
- Δημιουργία πίνακα συμβόλων
- Παραγωγή τελικού κώδικα σε γλώσσα μηχανής (RISC-V Assembly)

# 5 Παραγωγή ενδιάμεσου κώδικα

## 5.1 Λίγα λόγια για τον ενδιάμεσο κώδικα

Αν η συντακτική ανάλυση του greek++ προγράμματος στεφθεί με επιτυχία, δηλαδή το πρόγραμμα δεν περιέχει συντακτικά λάθη, το επόμενο βήμα της μεταγλώττισης είναι η παραγωγή του ενδιάμεσου κώδικα. Ο ενδιάμεσος κώδικας είναι ένα σύνολο τετράδων της μορφής: op, x, y, z, όπου op είναι ο τελεστής της πράξης που εκτελείται και x, y, z τα τελούμενα της πράξης. Οι τετράδες αυτές χαρακτηρίζονται στο αριστερό τους τμήμα από έναν αριθμό, ο οποίος είναι μοναδικός. Όταν τελειώσει η εκτέλεση μίας τετράδας, εκτελείται η τετράδα που έχει τον αμέσως μεγαλύτερο αριθμό, εκτός αν η τετράδα που μόλις εκτελέστηκε υποδείξει διαφορετικό αριθμό. Ένα παράδειγμα τετράδας είναι: 150: +, x, y, z. Μόλις δημιουργηθούν όλες οι τετράδες, αποθηκεύονται σε ένα αρχείο .int, το οποίο δημιουργούμε εμείς.

## 5.2 Περιπτώσεις των τετράδων του ενδιάμεσου κώδικα

Η μορφή που έχει μια τετράδα στον ενδιάμεσο κώδικα εξαρτάται αποκλειστικά από τον τελεστή της πράξης που ενδέχεται να εκτελεστεί. Συγκεκριμένα, διακρίνουμε τις εξής περιπτώσεις:

- **Αριθμητικές πράξεις:** op, x, y, z, όπου:
  - op είναι ένας αριθμητικός τελεστής (+, -, *, /)
  - x, y είναι είτε ονόματα μεταβλητών, είτε αριθμητικές σταθερές
  - z είναι όνομα μεταβλητής

  Συγκεκριμένα, εφαρμόζεται ο τελεστής op στα τελούμενα x και y και το αποτέλεσμα τοποθετείται στο τελούμενο z. Για παράδειγμα, η τετράδα +, a, b, c αντιστοιχεί στην πράξη c = a + b.

- **Ανάθεση τιμής:** =, x, _, z, όπου:
  - x είναι είτε όνομα μεταβλητής, είτε αριθμητική σταθερά
  - z είναι όνομα μεταβλητής

  Συγκεκριμένα, εκχωρείται η τιμή του τελούμενου x στο τελούμενο z. Η τετράδα αυτής της μορφής αντιστοιχεί στην εκχώρηση z = x.

- **Άλμα χωρίς συνθήκη:** jump, _, _, z, όπου z είναι ο αριθμός κάποιας άλλης τετράδας. Συγκεκριμένα, εκτελείται μεταπήδηση χωρίς όρους στον αριθμό της τετράδας που υποδεικνύει το z.

- **Άλμα με συνθήκη:** relop, x, y, z, όπου:

  - relop είναι ένας σχεσιακός τελεστής (==, !=, <, >, <=, >=)
  - x, y είναι είτε ονόματα μεταβλητών, είτε αριθμητικές σταθερές
  - z είναι ο αριθμός κάποιας άλλης τετράδας

  Συγκεκριμένα, αν ισχύει η συνθήκη x relop y, εκτελείται μεταπήδηση στον αριθμό της τετράδας που υποδεικνύει το z.

- **Αρχή ενότητας:** begin_block, name, _, _ Συγκεκριμένα, αυτό σημαίνει ότι βρισκόμαστε στην αρχή του προγράμματος ή του υποπρογράμματος με το όνομα name.

- **Τέλος ενότητας:** end_block, name, _, _ Συγκεκριμένα, αυτό σημαίνει ότι βρισκόμαστε στο τέλος του προγράμματος ή του υποπρογράμματος με το όνομα name.

- **Τερματισμός προγράμματος:** halt, _, _, _ Συγκεκριμένα, η τετράδα αυτής της μορφής είναι η προτελευταία τετράδα του ενδιάμεσου κώδικα και δημιουργείται πριν κλείσει το block της main συνάρτησης.

- **Πέρασμα παραμέτρων:** par, x, m, _, όπου:

  - x είναι η παράμετρος της συνάρτησης που καλείται, η οποία είναι είτε όνομα μεταβλητής, είτε αριθμητική σταθερά
  - m είναι ο τρόπος μετάδοσης της παραμέτρου x

  Συγκεκριμένα, υπάρχουν 2 τρόποι μετάδοσης παραμέτρων: CV, όταν έχουμε μετάδοση με τιμή και RET, όταν έχουμε επιστροφή τιμής συνάρτησης. Στην περίπτωση RET, η μεταβλητή x είναι πάντα προσωρινή μεταβλητή.

- **Κλήση συνάρτησης:** call, name, _, _, όπου name είναι το όνομα της συνάρτησης που καλείται.

- **Επιστροφή τιμής συνάρτησης:** retv, x, _, _, όπου x είναι είτε όνομα μεταβλητής, είτε αριθμητική σταθερά.

- **Είσοδος τιμής (input):** inp, x, _, _, όπου x είναι το όνομα της μεταβλητής, της οποίας η τιμή εισάγεται από το πληκτρολόγιο.

- **Έξοδος αποτελεσμάτων (output):** out, x, _, _, όπου x είναι είτε όνομα μεταβλητής, είτε αριθμητική σταθερά, της οποίας η τιμή τυπώνεται στην οθόνη.

## 5.3 Προγραμματιστική υλοποίηση της δημιουργίας των τετράδων του ενδιάμεσου κώδικα

Για τη δημιουργία των τετράδων του ενδιάμεσου κώδικα χρησιμοποιήθηκαν οι παρακάτω συναρτήσεις:

- **nextquad():** επιστρέφει τον αριθμό της επόμενης τετράδας που πρόκειται να δημιουργηθεί
- **genquad(op, x, y, z):** δημιουργεί και επιστρέφει την επόμενη τετράδα op, x, y, z
- **newtemp():** δημιουργεί και επιστρέφει μια προσωρινή μεταβλητή της μορφής %0, %1, κλπ.
- **emptylist():** δημιουργεί και επιστρέφει μια κενή λίστα ετικετών (αριθμών) τετράδων
- **makelist(quad_number):** δημιουργεί και επιστρέφει μια λίστα ετικετών (αριθμών) τετράδων που περιέχει μόνο την ετικέτα quad_number
- **merge(list1, list2):** δημιουργεί και επιστρέφει μια λίστα ετικετών (αριθμών) τετράδων που προκύπτει από τη συνένωση των λιστών list1 και list2
- **backpatch(uncompleted_quads, quad_number):** επισκέπτεται μία προς μία τις τετράδες που δεν έχουν συμπληρωθεί στο τέλος (το τελευταίο τους στοιχείο ισούται με '_') και τις συμπληρώνει με την ετικέτα (αριθμό) μιας συγκεκριμένης τετράδας

Η συνάρτηση parse() της κλάσης Syntax καλείται στο main τμήμα του .py αρχείου, ενώ οι υπόλοιπες συναρτήσεις καλούνται εντός της συνάρτησης syntax() που υλοποιεί τη συντακτική ανάλυση.

Οι global μεταβλητές και λίστες που χρησιμοποιήθηκαν για τη δημιουργία του ενδιάμεσου κώδικα είναι οι εξής:

- Η λίστα **quads[]** η οποία αποθηκεύει τις τετράδες του ενδιάμεσου κώδικα. Η λίστα έχει κυρίως βοηθητικό ρόλο, γιατί μόλις δημιουργηθεί ο τελικός κώδικας (θα μιλήσουμε για αυτόν στην αντίστοιχη ενότητα) μιας συνάρτησης (είτε της main, είτε κάποιας άλλης) θα γίνεται κενή (reset), ενώ η δεύτερη λίστα θα αποθηκεύει όλες τις τετράδες του ενδιάμεσου κώδικα.
- Η μεταβλητή **next_quad**, η οποία εκφράζει τον αριθμό (ετικέτα) της τετράδας του ενδιάμεσου κώδικα. Η τιμή της αρχικοποιείται στην τιμή 0, δηλαδή ο αριθμός της πρώτης τετράδας είναι ίσος με 0.
- Η μεταβλητή **temp_counter**, η οποία εκφράζει το πλήθος των προσωρινών μεταβλητών που χρησιμοποιούνται στον ενδιάμεσο κώδικα

# 6 Παραγωγή τελικού κώδικα

## 6.1 Λίγα λόγια για τον τελικό κώδικα

Η παραγωγή του τελικού κώδικα είναι το τελευταίο στάδιο της μεταγλώττισης. Ο κώδικας αυτός θα είναι γραμμένος στη συμβολική γλώσσα Assembly του επεξεργαστή αρχιτεκτονικής RISC-V. Για να μπορέσει να παραχθεί σωστά, πρέπει πρώτα να παραχθούν σωστά όλες οι τετράδες του ενδιάμεσου κώδικα, καθώς και ο πίνακας συμβόλων. Συγκεκριμένα, από κάθε εντολή ενδιάμεσου κώδικα προκύπτει μία σειρά εντολών τελικού κώδικα, η οποία για να παραχθεί ανακτά πληροφορίες από τον πίνακα συμβόλων. Κύριες ενέργειες στη φάση αυτή είναι οι μεταβλητές, οι οποίες απεικονίζονται στην μνήμη (στοίβα), το πέρασμα παραμέτρων και η κλήση συναρτήσεων.

## 6.2 Καταχωρητές του RISC-V επεξεργαστή

- Καταχωρητής **zero**: η τιμή του είναι πάντα ίση με 0
- Καταχωρητές προσωρινής αποθήκευσης τιμών: **t0**, …, **t6**
- Καταχωρητές των οποίων οι τιμές διατηρούνται ανάμεσα σε κλήσεις συναρτήσεων: **s0**, …, **s11**
- Καταχωρητές ορισμάτων: **a0**, …, **a7**
- **stack pointer**: **sp**
- **frame pointer**: **fp**
- **return address**: **ra**
- **global pointer**: **gp**

## 6.3 Εντολές RISC-V Assembly

- **Απευθείας εκχώρηση αριθμητικής σταθεράς:**
  `li reg, int`, όπου `int` ένας ακέραιος αριθμός
- **Μεταφορά τιμής:** `mv reg1, reg2`
- **Αριθμητικές πράξεις μεταξύ καταχωρητών:**
  - `add reg, reg1, reg2` (Πρόσθεση)
  - `sub reg, reg1, reg2` (Αφαίρεση)
  - `mul reg, reg1, reg2` (Πολλαπλασιασμός)
  - `div reg, reg1, reg2` (Διαίρεση)
- **Πρόσθεση μεταξύ καταχωρητή και αριθμητικής σταθεράς:**
  `addi target_reg, source_reg, int`
- **Πρόσβαση στη μνήμη:**
  - **Μεταφορά από τη μνήμη (στοίβα) στον καταχωρητή:**
    `lw reg1, offset(reg2)`, όπου `reg1` ο καταχωρητής προορισμού, `reg2` ο καταχωρητής βάσης και `offset` η απόσταση από τον `reg2`
  - **Μεταφορά από τον καταχωρητή στη μνήμη (στοίβα):**
    `sw reg1, offset(reg2)`, όπου `reg1` ο καταχωρητής πηγής, `reg2` ο καταχωρητής βάσης και `offset` η απόσταση από τον `reg2`

  Αν το `offset` είναι 0, τότε μπορεί να παραλειφθεί.
- **Εντολές διακλάδωσης που κάνουν άλματα (branches):**
  `branch t1, t2, label`, όπου το `branch` είναι ένα από τα παρακάτω:
  `beq` (==), `bne` (!=), `blt` (<), `ble` (<=), `bgt` (>), `bge` (>=)
- **Κλήσεις συναρτήσεων:**

- `j label` (άλμα στο label)
- `jal label` (κλήση συνάρτησης)
- `jr ra` (άλμα στη διεύθυνση του καταχωρητή ra που περιέχει την διεύθυνση επιστροφής συνάρτησης)

- **Είσοδος δεδομένων:**

```
li a7, 5
ecall
```

- **Έξοδος δεδομένων:**

```
li a0, 44
li a7, 1
ecall
```

- **Τερματισμός προγράμματος (halt):**

```
li a0, 0
li a7, 93
ecall
```

## 6.4 Προγραμματιστική υλοποίηση της δημιουργίας του τελικού κώδικα

Για τη δημιουργία του τελικού κώδικα χρησιμοποιήθηκε η κλάση `RISCVCodeGenerator` που υλοποιεί όλες τις απαραίτητες λειτουργίες για τη μετατροπή των τετράδων του ενδιάμεσου κώδικα σε κώδικα RISC-V Assembly. Οι σημαντικότερες μέθοδοι αυτής της κλάσης είναι:

- **emit(instruction):** Προσθέτει μια εντολή Assembly στον παραγόμενο κώδικα
- **emit_label(label):** Προσθέτει μια ετικέτα (label) στον παραγόμενο κώδικα
- **get_assembly_label(quad_label):** Μετατρέπει την ετικέτα μιας τετράδας σε ετικέτα Assembly
- **get_var_offset(var):** Υπολογίζει τη θέση μιας μεταβλητής στη μνήμη
- **allocate_register(var):** Αναθέτει έναν καταχωρητή σε μια μεταβλητή ή προσωρινή μεταβλητή
- **gnlvcode(var):** Παράγει κώδικα για την εύρεση της διεύθυνσης μιας μη τοπικής μεταβλητής
- **loadvr(v, r):** Φορτώνει την τιμή μιας μεταβλητής/σταθεράς v στον καταχωρητή r
- **storerv(r, v):** Αποθηκεύει την τιμή του καταχωρητή r στη μεταβλητή v

- **generate_code_from_quads(quads):** Παράγει τον τελικό κώδικα Assembly από τις τετράδες του ενδιάμεσου κώδικα
- **generate_data_section():** Δημιουργεί το τμήμα δεδομένων (data section) του Assembly κώδικα
- **get_complete_code():** Επιστρέφει τον πλήρη παραγόμενο κώδικα Assembly

Η κλάση `RISCVCodeGenerator` επεξεργάζεται κάθε τετράδα του ενδιάμεσου κώδικα και παράγει τις αντίστοιχες εντολές RISC-V Assembly. Η μετατροπή γίνεται ανάλογα με τον τελεστή της τετράδας. Για παράδειγμα:

- Για αριθμητικές πράξεις (+, -, *, /), παράγονται εντολές `add`, `sub`, `mul`, `div` αντίστοιχα
- Για εντολές ανάθεσης (:=), παράγονται εντολές φόρτωσης και αποθήκευσης (`lw`, `sw`)
- Για εντολές άλματος (jump), παράγονται εντολές `j` και `beq`/`bne`/κτλ.
- Για κλήσεις συναρτήσεων (call), παράγονται εντολές `jal`
- Για τα μπλοκ συναρτήσεων (begin_block, end_block), παράγεται ο κατάλληλος κώδικας για τη δημιουργία και καταστροφή του εγγραφήματος δραστηριοποίησης

Το αποτέλεσμα είναι ένα αρχείο .asm που περιέχει τον κώδικα RISC-V Assembly που υλοποιεί το αρχικό πρόγραμμα greek++.

### 6.4.1 Παράδειγμα δημιουργίας τελικού κώδικα

Για παράδειγμα, η τετράδα του ενδιάμεσου κώδικα `+, a, b, c` μετατρέπεται στις ακόλουθες εντολές RISC-V Assembly:

```
lw t0, -offset_a(sp)    # Φόρτωση του a στον καταχωρητή t0
lw t1, -offset_b(sp)    # Φόρτωση του b στον καταχωρητή t1
add t2, t0, t1          # Πρόσθεση των τιμών
sw t2, -offset_c(sp)    # Αποθήκευση του αποτελέσματος στη μεταβλητή c
```

Όπου `offset_a`, `offset_b`, και `offset_c` είναι οι θέσεις των μεταβλητών στο εγγράφημα δραστηριοποίησης, όπως αυτές υπολογίζονται από τον πίνακα συμβόλων.

Η διαδικασία παραγωγής του τελικού κώδικα ακολουθεί τα πρότυπα που περιγράφονται στις διαφάνειες του μαθήματος, διασφαλίζοντας ότι ο τελικός κώδικας υλοποιεί σωστά τη σημασιολογία του αρχικού προγράμματος greek++.

Να σημειωθεί ότι παρόλο που στην αναφορά γράφω πώς θα γίνει ο κώδικας σε RISC-V δεν το έχω υλοποιήσει πλήρως.