

ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΚΑΙ ΜΕΤΑΓΛΩΤΙΣΤΕΣ

Δημήτριος Νέτσκας (ΑΕΜ: 4341)
Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Τμήμα Πληροφορικής

Εργασία για το μάθημα “Γλώσσες προγραμματισμού και μεταγλωτιστές ”
Διδάσκων καθηγητές: Κατσαρός Παναγιώτης, Παπαρρίζος Ιωάννης

15/9/2025

1. Σκοπός & Περίληψη

Υλοποίηση ενός μεταγλωττιστή για απλή γλώσσα τύπου C. Ο μεταγλωττιστής είναι γραμμένος σε Python με PLY (Lex/Yacc) και παράγει MIXAL κώδικα, εκτελέσιμο σε MIX VM (GNU MDK ή online emulator). Περιλαμβάνει:

- **Λεξική ανάλυση** (tokens, σχόλια `//`, γραμμές σφάλματος, αριθμοί χωρίς leading zeros).
- **Συντακτική ανάλυση** βάσει της δοθείσας γραμματικής, με παραγωγή AST.
- **Πίνακα συμβόλων** ανά μέθοδο + παγκόσμιο scope, και regions μεταβλητών/παραμέτρων.
- **Σημασιολογικούς ελέγχους**: ύπαρξη `main`, `break` εκτός `while`, διπλές δηλώσεις μεταβλητών/συναρτήσεων, χρήση αδήλωτων, έλεγχος κλήσεων (ύπαρξη/πλήθος ορισμάτων).
- **Παραγωγή κώδικα MIXAL**: εκφράσεις, `if/else`, `while`, `break`, `return`, κλήσεις συναρτήσεων με απλή σύμβαση κλήσης.

2. Αρχιτεκτονική Υλοποίησης

Δομή έργου (κύρια αρχεία):

- `lexer.py`: Κανόνες λεξικής ανάλυσης (keywords, IDs, NUM, τελεστές, σχόλια, errors).
- `parser.py`: Κανόνες PLY/yacc βάσει γραμματικής. Παράγει AST με tuples.
- `symbol_table.py`: Δημιουργία πίνακα συμβόλων (global + ανά μέθοδο) από το AST.
- `semantic_check.py`: Έλεγχοι: αδήλωτες, `main`, `break` εκτός `while`, διπλές δηλώσεις, διπλές συναρτήσεις, έλεγχος κλήσεων.
- `mixal_generator.py`: Παραγωγός MIXAL. Labels, σταθερές, αποθήκη μεταβλητών, ροή ελέγχου.
- `main.py`: Εκτελεί όλη τη ροή και γράφει αναφορές στον φάκελο `output/`.

Ροή εκτέλεσης `main.py`:

1. Δημιουργία `output/`.
 2. **Λεξική ανάλυση** και αποθήκευση στο `output/lexical_analysis.txt`.
 3. **Parsing** → AST στο `output/ast_output.txt`.
 4. **Symbol Table** → `output/symbol_table.txt`.
 5. **Semantic Checks** (όλοι οι έλεγχοι, ενιαίο buffer) → `output/semantic_checks.txt`.
 - Αν υπάρχουν μηνύματα σφάλματος, τερματισμός πριν την παραγωγή MIXAL.
 6. **Codegen MIXAL** → `output/main.mixal` (μόνο αν δεν υπάρχουν λάθη πιο πριν).
-

3. Λεξική Ανάλυση (lexer.py)

- **Keywords** → tokens μέσω reserved (if, else, while, int, return, break, true/false).
 - **Literals**: + - * / = () { } ; , [] επιστρέφουν αυτούσια.
 - **Σύνθετοι τελεστές**: == != <= >= < > με regex κανόνες.
 - **NUM**: μόνο δεκαδικοί χωρίς leading zeros (π.χ. απορρίπτεται 05). Μετατρέπεται σε int.
 - **ID**: letter (letter|digit|_) *. Τα true/false μετατρέπονται σε boolean 1/0.
 - **Σχόλια**: // ... \n αγνοούνται. Whitespace αγνοείται.
 - **Αρίθμηση γραμμών και αναφορές σφαλμάτων** (t_error).
-

4. Συντακτική Ανάλυση & AST (parser.py)

- Υλοποίηση όλων των παραγωγών της εκφώνησης.
 - Το AST παράγεται με tuples. Ενδεικτικά:
 - Πρόγραμμα: ('program', [method_nodes...])
 - Μέθοδος: ('method', return_type, name, params_list, body_node)
 - Σώμα: ('body', decls_list, stmts_list)
 - Δήλωση: ('decl', type, [(name, init_expr_or_None), ...])
 - Εκχώρηση: ('assign', id, expr) + wrapper ('assign_stmt', ...)
 - if/else: ('if_stmt', cond_expr, then_stmt_or_block, else_stmt_or_empty)
 - while: ('while_stmt', cond_expr, body_stmt_or_block)
 - break: ('break_stmt',)
 - return: ('return_stmt', expr)
 - Εκφράσεις: πρόσθεση/αφαίρεση ('add', op, L, R), γιν/διαίρ ('mulop', op, L, R), συγκρίσεις ('relop', op, L, R), κλήσεις ('call', fname, [args...]), boolean ('bool', 0|1).
-

5. Σημασιολογικοί Έλεγχοι (semantic_check.py)

Υλοποιούνται οι απαιτούμενοι έλεγχοι:

- **main**: ύπαρξη main στο global.
- **break εκτός while**: DFS σε stmts με μετρητή «μέσα σε while».
- **Αδήλωτες μεταβλητές**: σε εκχωρήσεις/εκφράσεις εντός μεθόδου.
- **Διπλές δηλώσεις**: ανά μέθοδο (set-based έλεγχος).
- **Διπλές συναρτήσεις**: στο global.
- **Κλήσεις**: ύπαρξη συνάρτησης & συμβατότητα πλήθους ορισμάτων.

- Διαίρεση με το μηδέν

6. Παραγωγή Κώδικα MIXAL (mixal_generator.py)

6.1 Βασικές αρχές

- Γεννήτρια γραμμών (label, opcode, operand, comment) με αυστηρό μήκος label ≤ 10 και αποφυγή σύγκρουσης/δεσμευμένων.
- Πίνακες: const_pool (σταθερές), var_addr (labels για τοπικές/παραμέτρους), func_label, ret_label.
- Επικεφαλίδα: ORIG 2000 + άλμα στην main. Τερματισμός: EXIT: HLT.
- Τμήμα δεδομένων: ORIG 3000 και όλες οι CON (μεταβλητές/σταθερές/temps).
- Σύμβαση κλήσης:
 1. Τα ορίσματα αξιολογούνται και αποθηκεύονται στις πραγματικές θέσεις των παραμέτρων της καλούμενης (ανά όνομα).
 2. Κλήση με JMP FUNC_LABEL.
 3. Η διεύθυνση επιστροφής σώζεται/φορτώνεται μέσω STJ RET(0:2) και RET_LABEL: JMP 0 (self-modifying return).
 4. Επιστρεφόμενη τιμή στο A.

6.2 Εκφράσεις & Τελεστές

- +/− με ADD/SUB, * με MUL (χειρισμός πλάτους μέσω A:X), / με DIV (στήσιμο A:X όπου χρειάζεται).
- Σχέσεις: CMPA + άλματα για συνθήκες ή υλοποίηση boolean (0/1) σε A.

6.3 Ροή ελέγχου

- if/else: ετικέτες ELSE*, ENDIF* και condition jump-false.
- while: ετικέτες WH* (top) και WEND* (exit). break με άλμα στο τρέχον WEND*.
- return:
 - Στη main: άλμα σε EXIT.
 - Αλλιώς: άλμα στο RET_LABEL της τρέχουσας μεθόδου.

7. Δοκιμαστικά Προγράμματα & Αποτελέσματα

Ο φάκελος main.py περιλαμβάνει ένα ενδεικτικό πρόγραμμα.

Αρχεία εξόδου (παράγονται αυτόματα):



- output/lexical_analysis.txt (λίστα tokens)
- output/ast_output.txt (AST)
- output/symbol_table.txt (πίνακας συμβόλων)

- output/regions.txt (regions ανά μέθοδο)
- output/semantic_checks.txt (κενό όταν δεν υπάρχουν σφάλματα)
- output/main.mixal (κώδικας MIXAL)

Σημείωση: Για τα λανθασμένα παραδείγματα της εκφώνησης, ο parser/semantic αναμένεται να εμφανίσει μηνύματα λάθους και να μην παράγει MIXAL.

7.1 Λάθος πρόγραμμα 1 από εκφώνηση:

```
int method1(int a)
{
    int b;
    b = a+10;
    return b;
}
int main()
{
    return method1(5+);
}
```

Αποτέλεσμα:  Lexical analysis generated at output/lexical_analysis Syntax error at token ')' (line 5).  Syntax errors — aborting.

7.2 Σωστό πρόγραμμα 3 από εκφώνηση:

```
int method1(int a)
{
    int b;
    b = a+10;
    return b;
}
int method2(int c, int d)
{
    int e;
    e = method1(c);
    e = e + d;
    return e;
}
```

```

}
int main()
{
    return method2(5,6);
}

```

Αποτέλεσμα: ✓ Lexical analysis generated at output/lexical_analysis ✓ Abstract Syntax Tree generated at output/ast_output ✓ MIXAL generated at output/main.mixal

Παραγόμενος κώδικας MIXAL:

	ORIG	2000
START	NOP	
	JMP	MAIN
METHOD1	NOP	
	STJ	RETMETHOD1(0:2)
	LDA	METHOD1A
	ADD	K0001
	STA	METHOD1B
	LDA	METHOD1B
	JMP	RETMETHOD1
RETMETHOD1	JMP	0
METHOD2	NOP	
	STJ	RETMETHOD2(0:2)
	LDA	METHOD2C
	STA	METHOD1A
	JMP	METHOD1
	STA	METHOD2E
	LDA	METHOD2E
	ADD	METHOD2D
	STA	METHOD2E
	LDA	METHOD2E
	JMP	RETMETHOD2
RETMETHOD2	JMP	0
MAIN	NOP	
	STJ	RETMETHOD2(0:2)
	LDA	K0002
	STA	METHOD2C
	LDA	K0003
	STA	METHOD2D
	JMP	METHOD2
	JMP	EXIT
RETMETHOD2	JMP	0
EXIT	NOP	
	HLT	

	ORIG	3000
METHOD1A	CON	0
METHOD1B	CON	0
K0001	CON	10
METHOD2C	CON	0
METHOD2D	CON	0
METHOD2E	CON	0
K0002	CON	5
K0003	CON	6
	END	START

Στον emulator:

```
MIX> load test
Program loaded. Start address: 2000
MIX> run
Running ...
... done
Elapsed time: 59 /Total program time: 59 (Total uptime: 59)
MIX> pall
rA: + 00 00 00 00 21 (0000000021)
rX: + 00 00 00 00 00 (0000000000)
rJ: + 31 46 (2030)
rI1: + 00 00 (0000)      rI2: + 00 00 (0000)
rI3: + 00 00 (0000)      rI4: + 00 00 (0000)
rI5: + 00 00 (0000)      rI6: + 00 00 (0000)
Overflow: F
Cmp: E
```

8. Οδηγίες Build & Run

Απαιτήσεις: Python 3.10+, ply.

(προαιρετικά δημιουργία venv)

- python -m venv venv
 - venv\Scripts\activate
 - pip install ply
 - python main.py
-

9. Εκτέλεση MIXAL

Συναρμολόγηση/Εκτέλεση σε GNU MDK :

1. Φόρτωση `output/main.mixal`.
2. Πέρασμα του `output/main.mixal` στον emulator.
3. Άνοιγμα αρχείου με nano π.χ. `test.mixal` για την επικόλλησή του αποτελέσματος στον emulator
4. `Mixasm test.mixal` μεταφράζει (assemble) και παράγει αρχείο `test.mix` το οποίο φορτώνεται μετά στο `mixnm`.
5. `Mixnm` τρέχει (emulate) το πρόγραμμα MIX
6. Μέσα στο prompt:
 1. `load test` → φορτώνει το πρόγραμμα.
 2. `run` → το τρέχει μέχρι `HLT`.
 3. `pull` → δείχνει όλους τους καταχωρητές (για να δεις το αποτέλεσμα).