Σχολή ΗΜΜΥ, Πολυτεχνείο Κρήτης
ΠΛΗ 402 - Θεωρία Υπολογισμού (Εαρινό εξάμηνο 2016-2017)
Διδάσκων : Μ. Γ. Λαγουδάκης
Όνομα φοιτητή : Σκυβαλάκης Κωνσταντίνος
Ημερομηνία παράδοσης : 19 Μαΐου 2017
Εργασία Προγραμματισμού

1 Εισαγωγή

Σκοπός της εργασίας, είναι η κατασκευή ενός μεταγλλωτιστή για τη φανταστική γλώσσα προγραμματισμού **FC**(**F**ictional **C**). Πιο συγκεκριμένα, δημιουργήθηκε ένας transpiler για την απευθείας μετατροπή πηγαίου κώδικα FC σε πηγαίο κώδικα C.

2 Σχεδιασμός λεκτικού αναλυτή

Για τη δημιουργία του λεκτικού αναλυτή χρησιμοποιήθηκε το εργαλείο Flex. Στο αρχείο fc_flex.l μπορούμε να δούμε όλους τους κανόνες που χρησιμοποιούνται κατά την λεκτική ανάλυση του πηγαίου κώδικα FC. Επιπλέον, περιγράφεται που χρησιμεύουν αυτοί οι κανόνες και κατόπιν στο τέλος της ενότητας παρουσιάζονται αποτελέσματα καλής λειτουργίας του λεκτικού αναλυτή.

2.1 Ορισμοί λεκτικού αναλυτή

Αρχικά, στο αρχείο fc_flex.l ορίσαμε τις εκφράσεις που θα δεχόμαστε, ως θετικούς πραγματικούς αριθμούς (REAL), ως θετικούς ακέραιους αριθμούς (NUMBER), ως συμβολοσειρές πεπερασμένου μήκους μιας γραμμής (STRING) και τέλος ώς αναγνωριστικά (IDENTIFIER) όπως λ.χ. το όνομα μιας μεταβλητής. Το γεγονός ότι οι αριθμοί είναι ΄θετικοί΄ ισχύει άτυπα, απλώς επειδή δεν υπάρχει κάποιο πρόσημο μπροστά τους. Στη συνέχεια που θα συνδέσουμε και το κομμάτι της συντατικής ανάλυσης θα μπορέσουμε να ξεχωρίσουμε πότε ένας αριθμός είναι θετικός και πότε αρνητικός μέσω των κανόνων γραμματικής.

2.2 Κανόνες λεκτικού αναλυτή

Σε αυτό το κομμάτι του Flex κώδικα δηλώνουμε τις κωδικές λέξεις που δεχόμαστε στον κώδικα μας όπως είναι το begin,end, κλπ. Ακόμη, δηλώνουμε τα σύμβολα που δεχόμαστε στον κώδικα μας όπως είναι το +,-,*, κλπ. Τέλος, δηλώνουμε ως αναγνωρίσιμα και τα σχόλια μιας γραμμής (π.χ. //this is a comment) καθώς επίσης και τα σχόλια που μπορούν να επεκταθούν σε πολλές γραμμές (π.χ. /*.....*/). Ακολουθούν παρακάτω παραδείγματα καλής λειτουργίας του λεκτικού μας αναλυτή.

2.3 Παραδείγματα λειτουργίας λεκτικού αναλυτή

Για την επίδειξη της καλής λειτουργίας του λεκτικού αναλυτή συντάχτηκαν οι κώδικες που βρίσκονται στα αρχεία correct1.fc και correct2.fc. Επιπλέον, για να δείξουμε ότι ο λεκτικός αναλυτής μας δουλεύει ακόμα και σε μια περίπτωση που υπάρχουν λάθη στον κώδικα συντάχτηκε ο μικρού μήκους κώδικας που βρίσκεται στο αρχείο bad2.fc που χρησιμοποιείται το σύμβολο του αγκίστρου το οποίο και δεν αναγνωρίζεται επιτυχώς. Στην περίπτωση όπου υπάρχουν λάθη στον κώδικα FC ο λεκτικός αναλυτής σταματάει τη φάση της ανάλυσης και ο χρήστης θα πρέπει να διορθώσει τα λάθη του πριν επιχειρήσει να δοκιμάσει ξανά τον λεκτικό έλεγχο του κώδικα του. Να σημειωθεί ότι, οι κώδικες δεν έχουν κάποιο νόημα ως αλληλουχία εντολών εφόσον δεν ήταν αυτός ο σκοπός.

Listing 1: Flex output of correct1.fc

```
Line 1: Token KEYWORD_STRING: string
Line 1: Token IDENTIFIER: x
Line 1: Token ASSIGNMENT: :=
Line 1: Token STRING: "saas\n\r\t\\\"\'"
Line 1: Token SEMICOLON ;
Line 2: Token KEYWORD_CHAR: char
Line 2: Token IDENTIFIER: x
Line 2: Token ASSIGNMENT: :=
Line 2: Token CHAR: '\''
Line 2: Token SEMICOLON ;
Line 3: Token KEYWORD_INTEGER: integer
Line 3: Token IDENTIFIER: a
Line 3: Token BRACKET_LEFT: [
Line 3: Token CONST_INTEGER: 100
Line 3: Token BRACKET_RIGHT: ]
Line 3: Token BRACKET_LEFT: [
Line 3: Token CONST_INTEGER: 2
Line 3: Token BRACKET_RIGHT: ]
Line 3: Token BRACKET_LEFT: [
Line 3: Token CONST_INTEGER: 2
Line 3: Token BRACKET_RIGHT: ]
Line 3: Token BRACKET_LEFT: [
Line 3: Token CONST_INTEGER: 5
Line 3: Token BRACKET_RIGHT: ]
Line 3: Token BRACKET_LEFT: [
Line 3: Token CONST_INTEGER: 6
Line 3: Token BRACKET_RIGHT: ]
Line 3: Token SEMICOLON ;
Line 4: Token KEYWORD_INTEGER: integer
Line 4: Token IDENTIFIER: b
Line 4: Token BRACKET_LEFT: [
Line 4: Token CONST_INTEGER: 100
Line 4: Token BRACKET_RIGHT: ]
Line 4: Token SEMICOLON ;
Line 6: Token KEYWORD_STATIC: static
Line 6: Token KEYWORD_INTEGER: integer
Line 6: Token IDENTIFIER: ackermann
Line 6: Token PARENTHESIS_LEFT: (
```

```
Line 6: Token KEYWORD_INTEGER: integer
Line 6: Token IDENTIFIER: m
Line 6: Token COMMA ,
Line 6: Token KEYWORD_INTEGER: integer
Line 6: Token IDENTIFIER: n
Line 6: Token PARENTHESIS_RIGHT: )
Line 6: Token SEMICOLON ;
Line 8: Token KEYWORD_VOID: void
Line 8: Token IDENTIFIER: main
Line 8: Token PARENTHESIS_LEFT: (
Line 8: Token PARENTHESIS_RIGHT: )
Line 9: Token KEYWORD_BEGIN: begin
Line 10: Token KEYWORD_INTEGER: integer
Line 10: Token IDENTIFIER: x
Line 10: Token ASSIGNMENT: :=
Line 10: Token CONST_INTEGER: 0
Line 10: Token COMMA ,
Line 10: Token IDENTIFIER: i
Line 10: Token SEMICOLON ;
Line 11: Token KEYWORD_INTEGER: integer
Line 11: Token IDENTIFIER: ptra
Line 11: Token ASSIGNMENT: :=
Line 11: Token CONST_INTEGER: 4
Line 11: Token SEMICOLON ;
Line 12: Token KEYWORD_INTEGER: integer
Line 12: Token IDENTIFIER: b
Line 12: Token BRACKET_LEFT: [
Line 12: Token CONST_INTEGER: 100
Line 12: Token BRACKET_RIGHT: ]
Line 12: Token ASSIGNMENT: :=
Line 12: Token CONST_INTEGER: 4
Line 12: Token COMMA ,
Line 12: Token IDENTIFIER: y
Line 12: Token SEMICOLON ;
Line 13: Token KEYWORD_INTEGER: integer
Line 13: Token IDENTIFIER: foo
Line 13: Token PARENTHESIS_LEFT: (
Line 13: Token PARENTHESIS_RIGHT: )
Line 13: Token SEMICOLON ;
Line 14: Token KEYWORD_INTEGER: integer
Line 14: Token IDENTIFIER: x
Line 14: Token ASSIGNMENT: :=
Line 14: Token CONST_INTEGER: 4
Line 14: Token OP_DIV: /
Line 14: Token CONST_INTEGER: 2
Line 14: Token OP_PLUS: +
Line 14: Token CONST_INTEGER: 5
Line 14: Token SEMICOLON ;
Line 15: Token KEYWORD_INTEGER: integer
Line 15: Token IDENTIFIER: c
```

```
Line 15: Token BRACKET_LEFT: [
Line 15: Token CONST_INTEGER: 50
Line 15: Token BRACKET_RIGHT: ]
Line 15: Token BRACKET_LEFT: [
Line 15: Token CONST_INTEGER: 60
Line 15: Token BRACKET_RIGHT: ]
Line 15: Token ASSIGNMENT: :=
Line 15: Token CONST_INTEGER: 2
Line 15: Token COMMA ,
Line 15: Token IDENTIFIER: y
Line 15: Token SEMICOLON ;
Line 18: Token KEYWORD_FOR: for
Line 18: Token PARENTHESIS_LEFT: (
Line 18: Token IDENTIFIER: i
Line 18: Token ASSIGNMENT: :=
Line 18: Token CONST_INTEGER: 0
Line 18: Token SEMICOLON ;
Line 18: Token IDENTIFIER: i
Line 18: Token OP_LESS: <
Line 18: Token CONST_INTEGER: 100
Line 18: Token SEMICOLON ;
Line 18: Token IDENTIFIER: i
Line 18: Token OP_INCREMENT: ++
Line 18: Token PARENTHESIS_RIGHT: )
Line 18: Token IDENTIFIER: a
Line 18: Token BRACKET_LEFT: [
Line 18: Token IDENTIFIER: i
Line 18: Token BRACKET_RIGHT: ]
Line 18: Token ASSIGNMENT: :=
Line 18: Token IDENTIFIER: i
Line 18: Token SEMICOLON ;
Line 20: Token KEYWORD_FOR: for
Line 20: Token PARENTHESIS_LEFT: (
Line 20: Token IDENTIFIER: i
Line 20: Token ASSIGNMENT: :=
Line 20: Token CONST_INTEGER: 0
Line 20: Token SEMICOLON ;
Line 20: Token IDENTIFIER: i
Line 20: Token OP_LESS: <
Line 20: Token CONST_INTEGER: 100
Line 20: Token SEMICOLON ;
Line 20: Token IDENTIFIER: i
Line 20: Token OP_INCREMENT: ++
Line 20: Token PARENTHESIS_RIGHT: )
Line 21: Token KEYWORD_BEGIN: begin
Line 22: Token KEYWORD_IF: if
Line 22: Token PARENTHESIS_LEFT: (
Line 22: Token IDENTIFIER: i
Line 22: Token OP_GEQ: >=
Line 22: Token CONST_INTEGER: 999
```

```
Line 22: Token OP_AND: &&
Line 22: Token IDENTIFIER: i
Line 22: Token OP_LESS: <
Line 22: Token CONST_INTEGER: 200
Line 22: Token OP_AND: &&
Line 22: Token PARENTHESIS_LEFT: (
Line 22: Token IDENTIFIER: i
Line 22: Token OP_NOT_EQUAL: !=
Line 22: Token CONST_INTEGER: 3
Line 22: Token OP_OR: ||
Line 22: Token IDENTIFIER: i
Line 22: Token OP_GREATER: >
Line 22: Token CONST_INTEGER: 5
Line 22: Token PARENTHESIS_RIGHT: )
Line 22: Token PARENTHESIS_RIGHT: )
Line 22: Token IDENTIFIER: a
Line 22: Token BRACKET_LEFT: [
Line 22: Token IDENTIFIER: i
Line 22: Token OP_PLUS: +
Line 22: Token CONST_INTEGER: 1
Line 22: Token BRACKET_RIGHT: ]
Line 22: Token ASSIGNMENT: :=
Line 22: Token CONST_INTEGER: 1
Line 22: Token SEMICOLON ;
Line 22: Token LINE_COMMENT: //   case 1: don't need reload
Line 23: Token KEYWORD_IF: if
Line 23: Token PARENTHESIS_LEFT: (
Line 23: Token IDENTIFIER: b
Line 23: Token BRACKET_LEFT: [
Line 23: Token CONST_INTEGER: 1
Line 23: Token BRACKET_RIGHT: ]
Line 23: Token OP_NOT_EQUAL: !=
Line 23: Token CONST_INTEGER: 3
Line 23: Token PARENTHESIS_RIGHT: )
Line 24: Token KEYWORD_BEGIN: begin
Line 25: Token IDENTIFIER: b
Line 25: Token BRACKET_LEFT: [
Line 25: Token IDENTIFIER: i
Line 25: Token OP_PLUS: +
Line 25: Token PARENTHESIS_LEFT: (
Line 25: Token IDENTIFIER: ptra
Line 25: Token OP_MOD: mod
Line 25: Token IDENTIFIER: i
Line 25: Token PARENTHESIS_RIGHT: )
Line 25: Token OP_DIV: /
Line 25: Token CONST_INTEGER: 2
Line 25: Token BRACKET_RIGHT: ]
Line 25: Token ASSIGNMENT: :=
Line 25: Token IDENTIFIER: a
Line 25: Token BRACKET_LEFT: [
```

```
Line 25: Token CONST_INTEGER: 97
Line 25: Token BRACKET_RIGHT: ]
Line 25: Token SEMICOLON ;
Line 26: Token IDENTIFIER: b
Line 26: Token BRACKET_LEFT: [
Line 26: Token CONST_INTEGER: 1
Line 26: Token BRACKET_RIGHT: ]
Line 26: Token OP_INCREMENT: ++
Line 26: Token SEMICOLON ;
Line 27: Token KEYWORD_CONTINUE: continue
Line 27: Token SEMICOLON ;
Line 28: Token KEYWORD_END: end
Line 29: Token KEYWORD_END: end
Line 31: Token IDENTIFIER: foo
Line 31: Token PARENTHESIS_LEFT: (
Line 31: Token IDENTIFIER: b
Line 31: Token BRACKET_LEFT: [
Line 31: Token CONST_INTEGER: 11
Line 31: Token BRACKET_RIGHT: ]
Line 31: Token COMMA ,
Line 31: Token IDENTIFIER: b
Line 31: Token BRACKET_LEFT: [
Line 31: Token CONST_INTEGER: 23
Line 31: Token BRACKET_RIGHT: ]
Line 31: Token COMMA ,
Line 31: Token IDENTIFIER: b
Line 31: Token BRACKET_LEFT: [
Line 31: Token CONST_INTEGER: 23
Line 31: Token OP_PLUS: +
Line 31: Token CONST_INTEGER: 5
Line 31: Token BRACKET_RIGHT: ]
Line 31: Token BRACKET_LEFT: [
Line 31: Token IDENTIFIER: x
Line 31: Token OP_MOD: mod
Line 31: Token CONST_INTEGER: 2
Line 31: Token BRACKET_RIGHT: ]
Line 31: Token COMMA ,
Line 31: Token CONST_INTEGER: 4
Line 31: Token PARENTHESIS_RIGHT: )
Line 31: Token SEMICOLON ;
Line 32: Token IDENTIFIER: x
Line 32: Token ASSIGNMENT: :=
Line 32: Token IDENTIFIER: foo
Line 32: Token PARENTHESIS_LEFT: (
Line 32: Token CONST_INTEGER: 5
Line 32: Token PARENTHESIS_RIGHT: )
Line 32: Token SEMICOLON ;
Line 34: Token KEYWORD_IF: if
Line 34: Token PARENTHESIS_LEFT: (
Line 34: Token IDENTIFIER: b
```

```
Line 34: Token BRACKET_LEFT: [
Line 34: Token CONST_INTEGER: 1
Line 34: Token BRACKET_RIGHT: ]
Line 34: Token OP_NOT_EQUAL: !=
Line 34: Token CONST_INTEGER: 3
Line 34: Token PARENTHESIS_RIGHT: )
Line 35: Token KEYWORD_BEGIN: begin
Line 36: Token IDENTIFIER: b
Line 36: Token BRACKET_LEFT: [
Line 36: Token CONST_INTEGER: 10
Line 36: Token BRACKET_RIGHT: ]
Line 36: Token ASSIGNMENT: :=
Line 36: Token CONST_INTEGER: 5
Line 36: Token SEMICOLON ;
Line 37: Token KEYWORD_IF: if
Line 37: Token PARENTHESIS_LEFT: (
Line 37: Token IDENTIFIER: b
Line 37: Token BRACKET_LEFT: [
Line 37: Token CONST_INTEGER: 5
Line 37: Token BRACKET_RIGHT: ]
Line 37: Token OP_GREATER: >
Line 37: Token CONST_INTEGER: 3
Line 37: Token PARENTHESIS_RIGHT: )
Line 38: Token KEYWORD_BEGIN: begin
Line 39: Token IDENTIFIER: b
Line 39: Token BRACKET_LEFT: [
Line 39: Token CONST_INTEGER: 11
Line 39: Token BRACKET_RIGHT: ]
Line 39: Token ASSIGNMENT: :=
Line 39: Token OP_MINUS: -
Line 39: Token CONST_INTEGER: 6
Line 39: Token SEMICOLON ;
Line 40: Token IDENTIFIER: b
Line 40: Token BRACKET_LEFT: [
Line 40: Token CONST_INTEGER: 12
Line 40: Token BRACKET_RIGHT: ]
Line 40: Token ASSIGNMENT: :=
Line 40: Token OP_MINUS: -
Line 40: Token CONST_INTEGER: 3
Line 40: Token SEMICOLON ;
Line 41: Token KEYWORD_END: end
Line 42: Token KEYWORD_ELSE: else
Line 43: Token KEYWORD_BEGIN: begin
Line 44: Token IDENTIFIER: b
Line 44: Token BRACKET_LEFT: [
Line 44: Token CONST_INTEGER: 12
Line 44: Token BRACKET_RIGHT: ]
Line 44: Token ASSIGNMENT: :=
Line 44: Token OP_MINUS: -
Line 44: Token CONST_INTEGER: 3
```

```
Line 44: Token SEMICOLON ;
Line 45: Token KEYWORD_END: end
Line 46: Token KEYWORD_END: end
Line 47: Token KEYWORD_ELSE: else
Line 47: Token KEYWORD_IF: if
Line 47: Token PARENTHESIS_LEFT: (
Line 47: Token IDENTIFIER: b
Line 47: Token BRACKET_LEFT: [
Line 47: Token CONST_INTEGER: 2
Line 47: Token BRACKET_RIGHT: ]
Line 47: Token OP_NOT_EQUAL: !=
Line 47: Token CONST_INTEGER: 3
Line 47: Token PARENTHESIS_RIGHT: )
Line 48: Token KEYWORD_BEGIN: begin
Line 49: Token IDENTIFIER: b
Line 49: Token BRACKET_LEFT: [
Line 49: Token CONST_INTEGER: 11
Line 49: Token BRACKET_RIGHT: ]
Line 49: Token ASSIGNMENT: :=
Line 49: Token OP_MINUS: -
Line 49: Token CONST_INTEGER: 6
Line 49: Token SEMICOLON ;
Line 50: Token IDENTIFIER: b
Line 50: Token BRACKET_LEFT: [
Line 50: Token CONST_INTEGER: 11
Line 50: Token BRACKET_RIGHT: ]
Line 50: Token ASSIGNMENT: :=
Line 50: Token OP_MINUS: -
Line 50: Token CONST_INTEGER: 6
Line 50: Token SEMICOLON ;
Line 51: Token KEYWORD_END: end
Line 52: Token KEYWORD_ELSE: else
Line 53: Token IDENTIFIER: b
Line 53: Token BRACKET_LEFT: [
Line 53: Token CONST_INTEGER: 11
Line 53: Token BRACKET_RIGHT: ]
Line 53: Token ASSIGNMENT: :=
Line 53: Token CONST_INTEGER: 5
Line 53: Token SEMICOLON ;
Line 55: Token IDENTIFIER: x
Line 55: Token ASSIGNMENT: :=
Line 55: Token IDENTIFIER: ackermann
Line 55: Token PARENTHESIS_LEFT: (
Line 55: Token IDENTIFIER: n
Line 55: Token COMMA ,
Line 55: Token IDENTIFIER: m
Line 55: Token PARENTHESIS_RIGHT: )
Line 55: Token SEMICOLON ;
Line 57: Token KEYWORD_RETURN: return
Line 57: Token IDENTIFIER: x
```

```
Line 57: Token OP_LESS: <
Line 57: Token CONST_INTEGER: 5
Line 57: Token SEMICOLON ;
Line 58: Token KEYWORD_END: end
Line 60: Token KEYWORD_INTEGER: integer
Line 60: Token IDENTIFIER: ackermann
Line 60: Token PARENTHESIS_LEFT: (
Line 60: Token KEYWORD_INTEGER: integer
Line 60: Token IDENTIFIER: m
Line 60: Token COMMA ,
Line 60: Token KEYWORD_INTEGER: integer
Line 60: Token IDENTIFIER: n
Line 60: Token PARENTHESIS_RIGHT: )
Line 61: Token KEYWORD_BEGIN: begin
Line 62: Token KEYWORD_IF: if
Line 62: Token PARENTHESIS_LEFT: (
Line 62: Token OP_NOT: !
Line 62: Token IDENTIFIER: m
Line 62: Token PARENTHESIS_RIGHT: )
Line 62: Token KEYWORD_RETURN: return
Line 62: Token IDENTIFIER: n
Line 62: Token OP_PLUS: +
Line 62: Token CONST_INTEGER: 1
Line 62: Token SEMICOLON ;
Line 63: Token KEYWORD_IF: if
Line 63: Token PARENTHESIS_LEFT: (
Line 63: Token OP_NOT: !
Line 63: Token IDENTIFIER: n
Line 63: Token PARENTHESIS_RIGHT: )
Line 63: Token KEYWORD_RETURN: return
Line 63: Token IDENTIFIER: ackermann
Line 63: Token PARENTHESIS_LEFT: (
Line 63: Token IDENTIFIER: m
Line 63: Token OP_MINUS: -
Line 63: Token CONST_INTEGER: 1
Line 63: Token COMMA ,
Line 63: Token CONST_INTEGER: 1
Line 63: Token PARENTHESIS_RIGHT: )
Line 63: Token SEMICOLON ;
Line 64: Token KEYWORD_RETURN: return
Line 64: Token IDENTIFIER: ackermann
Line 64: Token PARENTHESIS_LEFT: (
Line 64: Token IDENTIFIER: m
Line 64: Token OP_MINUS: -
Line 64: Token CONST_INTEGER: 1
Line 64: Token COMMA ,
Line 64: Token IDENTIFIER: ackermann
Line 64: Token PARENTHESIS_LEFT: (
Line 64: Token IDENTIFIER: m
Line 64: Token COMMA ,
```

```
Line 64: Token IDENTIFIER: n
Line 64: Token OP_MINUS: -
Line 64: Token CONST_INTEGER: 1
Line 64: Token PARENTHESIS_RIGHT: )
Line 64: Token PARENTHESIS_RIGHT: )
Line 64: Token SEMICOLON ;
Line 65: Token KEYWORD_END: end
```

Listing 2: Flex output of correct2.fc

```
Line 1: Token KEYWORD_INTEGER: integer
Line 1: Token IDENTIFIER: main
Line 1: Token PARENTHESIS_LEFT: (
Line 1: Token PARENTHESIS_RIGHT: )
Line 2: Token KEYWORD_BEGIN: begin
Line 3: Token KEYWORD_INTEGER: integer
Line 3: Token IDENTIFIER: i
Line 3: Token COMMA ,
Line 3: Token IDENTIFIER: j
Line 3: Token COMMA ,
Line 3: Token IDENTIFIER: rows
Line 3: Token SEMICOLON ;
Line 5: Token IDENTIFIER: writeString
Line 5: Token PARENTHESIS_LEFT: (
Line 5: Token STRING: "Enter number of rows: "
Line 5: Token PARENTHESIS_RIGHT: )
Line 5: Token SEMICOLON ;
Line 6: Token IDENTIFIER: readInt
Line 6: Token PARENTHESIS_LEFT: (
Line 6: Token IDENTIFIER: d
Line 6: Token PARENTHESIS_RIGHT: )
Line 6: Token SEMICOLON ;
Line 8: Token KEYWORD_FOR: for
Line 8: Token PARENTHESIS_LEFT: (
Line 8: Token IDENTIFIER: i
Line 8: Token ASSIGNMENT: :=
Line 8: Token CONST_INTEGER: 1
Line 8: Token SEMICOLON ;
Line 8: Token IDENTIFIER: i
Line 8: Token OP_LEQ: <=
Line 8: Token IDENTIFIER: rows
Line 8: Token SEMICOLON ;
Line 8: Token IDENTIFIER: i
Line 8: Token OP_INCREMENT: ++
Line 8: Token PARENTHESIS_RIGHT: )
Line 9: Token KEYWORD_BEGIN: begin
Line 10: Token KEYWORD_FOR: for
```

```
Line 10: Token PARENTHESIS_LEFT: (
Line 10: Token IDENTIFIER: j
Line 10: Token ASSIGNMENT: :=
Line 10: Token CONST_INTEGER: 1
Line 10: Token SEMICOLON ;
Line 10: Token IDENTIFIER: j
Line 10: Token OP_LEQ: <=
Line 10: Token IDENTIFIER: i
Line 10: Token SEMICOLON ;
Line 10: Token IDENTIFIER: j
Line 10: Token OP_INCREMENT: ++
Line 10: Token PARENTHESIS_RIGHT: )
Line 11: Token KEYWORD_BEGIN: begin
Line 12: Token IDENTIFIER: writeString
Line 12: Token PARENTHESIS_LEFT: (
Line 12: Token STRING: "* "
Line 12: Token PARENTHESIS_RIGHT: )
Line 12: Token SEMICOLON ;
Line 13: Token KEYWORD_END: end
Line 14: Token IDENTIFIER: writeString
Line 14: Token PARENTHESIS_LEFT: (
Line 14: Token STRING: "\n"
Line 14: Token PARENTHESIS_RIGHT: )
Line 14: Token SEMICOLON ;
Line 15: Token KEYWORD_END: end
Line 16: Token KEYWORD_RETURN: return
Line 16: Token CONST_INTEGER: 0
Line 16: Token SEMICOLON ;
Line 17: Token KEYWORD_END: end
```

Listing 3: Flex output of bad2.fc

```
Line 1: Token KEYWORD_INTEGER: integer
Line 1: Token IDENTIFIER: main
Line 1: Token PARENTHESIS_LEFT: (
Line 1: Token PARENTHESIS_RIGHT: )
Line 2: Token KEYWORD_BEGIN: begin
Unrecognized token { in line 3.
```

3 Σχεδιασμός συντακτικού αναλυτή

Για τη δημιουργία του συντακτικού αναλυτή χρησιμοποιήθηκε το εργαλείο Bison. Στο αρχείο fc_bison.y μπορούμε να δούμε όλους τους κανόνες γραμματικής που χρησιμοποιούνται κατά την συντακτική ανάλυση του πηγαίου κώδικα FC. Επιπλέον, περιγράφεται που χρησιμεύουν αυτοί οι κανόνες και κατόπιν στο τέλος της ενότητας παρουσιάζονται αποτελέσματα καλής λειτουργίας πλέον του transpiler. Απώτερος σκοπός καθόλη τη διάρκεια του κομματιού της συντακτικής ανάλυσης ήταν η ελαχιστοποίηση των shift/reduce και reduce/reduce conflicts.

3.1 Ορισμοί συντακτικού αναλυτή

Αρχικά, στο αρχείο fc_bison.y ορίζονται τα tokens (τερματικά σύμβολα) που θα αποδέχεται το τμήμα της συντακτικής ανάλυσης από το Flex μερικά εκ των οποίων είναι τα KW_BEGIN, KW_END, κλπ. Στη συνέχεια ακολουθούν οι δηλώσεις προσεταιριστικότητας και προτεραιότητας των πράξεων, οι οποίες περιλαμβάνουν τους αριθμητικούς τελεστές που αναγνωρίζει το Flex. Ακόμη, δημιουργήθηκε μια συλλογή τύπων (union) για τη διευκόλυνση της υλοποίησης της άσκησης όμως περιέχει μόνο έναν τύπο δεδομένων, το string. Τέλος, ακολουθεί η δήλωση των σημασιολογικών τιμών των γραμματικών κανόνων που χρησιμοποιούμε.

3.2 Κανόνες γραμματικής συντακτικού αναλυτή

Στο συγκεκριμένο τμήμα του αρχείου του συντακτικού μας αναλυτή, έγινε προσπάθεια ομαδοποίησης των κανόνων αναλόγως το είδος τους, με απώτερο σκοπό την ευκολότερη απασφαλμάτωση του κώδικα και τη μείωση των conflicts. Πιο συγκεκριμένα, ο κώδικας διαχωρίστηκε σε 4 υποτυπώδη τα οποία παρουσιάζονται και αναλύονται παρακάτω.

- Expressions

  Οι γραμματικοί κανόνες που δημιουργήθηκαν για τα expressions είναι οι εξής :
  data_types,
  postfix_expression,
  argument_list,
  unary_operation,
  operation_expression,
  relational_expression,
  and_or_expression,
  assignment_expression
  και expression.

  Οι παραπάνω κανόνες δημιουργήθηκαν για τον εντοπισμό εκφράσεων όπως η σύγκριση μεταξύ εκφράσεων ή οι αριθμητικές πράξεις μεταξύ εκφράσεων κλπ.

- Statements

  Οι γραμματικοί κανόνες που δημιουργήθηκαν για τα statements είναι οι εξής :
  statement,
  statement_list,
  declaration_list,
  begin_end_statement,
  expression_statement,
  for_expression_statement,
  if_statement,
  jump_statement
  και loop_statement.

  Οι παραπάνω κανόνες δημιουργήθηκαν για τον εντοπισμό εκφράσεων όπως είναι οι βρόγχοι επανάληψης ή οι βρόγχοι begin-end ή ακόμη και για βρόγχους ελέγχου if-else if-else κλπ.

- Declarations

  Οι γραμματικοί κανόνες που δημιουργήθηκαν για τα declarations είναι οι εξής :
  type_specifier,
  parameter_list,
  declaration_specifiers,
  declaration,
  array_declare,
  declarator
  και init_declarator.

  Οι παραπάνω κανόνες δημιουργήθηκαν για τον εντοπισμό εκφράσεων όπως είναι οι τύποι μεταβλητών ή η έκφραση συναρτήσεων και πινάκων ή ακόμη και λίστε παραμέτρων που περνάμε ώς ορίσματα σε συνάρτηση κλπ.

- Program

  Οι γραμματικοί κανόνες που δημιουργήθηκαν για το program είναι οι εξής :
  program,
  translation_unit,
  global_declaration
  και function_declaration.

  Από το τελευταίο αυτό κομμάτι κανόνων γραμματικής ξεκινάει ουσιαστικά η αναγνώριση του κώδικα μας και η εγγραφή του μεταγλωττισμένου αρχείου σε ένα αρχείο εξόδου output.c.

3.3 Επίλογος συντακτικού αναλυτή

Στον επίλογο του αρχείου fc_bison.y δεν έχει δωθεί κάποιος άλλος ορισμός για τη συνάρτηση yyerror(), απλώς καλείται η συνάρτηση yyparse() και αναλόγως την τιμή επιστροφής της αντιλαμβανόμαστε αν υπήρχε σε κάποιο σημείο του κώδικα μας κάποιο συντακτικό λάθος (Result: Rejected!) ή όχι (Result: Accepted!). Ακολουθούν παρακάτω παραδείγματα καλής λειτουργίας του συντακτικού μας αναλυτή.

3.4 Παραδείγματα λειτουργίας συντακτικού αναλυτή

Για την επίδειξη της καλής λειτουργίας του συντακτικού μας αναλυτή θα χρησιμοποιήσουμε και πάλι τους κώδικες που βρίσκονται στο αρχείο correct1.fc και correct2.fc. Επιπλέον, για να δείξουμε ότι ο συντακτικός αναλυτής μας δουλεύει ακόμα και σε μια περίπτωση που υπάρχουν συντακτικά λάθη στον κώδικα θα χρησιμοποιήσουμε και πάλι τον μικρού μήκους κώδικας που βρίσκεται στο αρχείο bad1.fc όπου κάνουμε λάθος σύνταξη στην εντολή *return*. Στην περίπτωση όπου υπάρχουν συντακτικά λάθη στον κώδικα FC ο συντακτικός αναλυτής σταματάει τη φάση της ανάλυσης και ο χρήστης θα πρέπει να διορθώσει τα λάθη του πριν επιχειρήσει να δοκιμάσει ξανά τον συντακτικό έλεγχο του κώδικα του. Το αποτέλεσμα της συντακτικής ανάλυσης στην περίπτωση του σωστού κώδικα FC θα είναι ένας κώδικας σε γλώσσα C ο οποίος θα τυπώνεται στο terminal αλλά θα αποθηκεύεται και στο αρχείο εξόδου output.c. Στην περίπτωση του λανθασμένου FC κώδικα θα σταματτήσει η ανάλυση με κάποιο μήνυμα λάθος προς τον χρήστη. Να τονιστεί ακόμη ότι υπάρχουν πάρα πολλές περιπτώσεις συνταντικών λαθών τις οποίες αναγνωρίζει επιτυχώς ο συντακτικός αναλυτής μας αλλά δεν μπορούμε να τις παρουσιάσουμε εκτενώς όλες. Τα αποτελέσματα (σε κώδικα C) του parsing των correct.fc αρχείων θα τα βρείτε και μέσα στο zip της εργασίας.

Listing 4: Flex output of correct2.fc

```
#include <stdio.h>
#include "fclib.h"

int main() {
 int i,j,rows;
 writeString("Enter number of rows: ");
 readInt(d);
 for(i = 1; i <= rows; i++)
{
 for(j = 1; j <= i; j++)
{
 writeString("* ");
}

 writeString("\n");
}

 return 0;

}
```

Listing 5: Bison output of bad1.fc

```
Line 1: Token KEYWORD_INTEGER: integer
Line 1: Token IDENTIFIER: main
Line 1: Token PARENTHESIS_LEFT: (
Line 1: Token PARENTHESIS_RIGHT: )
Line 2: Token KEYWORD_BEGIN: begin
Line 3: Token KEYWORD_INTEGER: integer
Line 3: Token IDENTIFIER: x
Line 3: Token ASSIGNMENT: :=
Line 3: Token CONST_INTEGER: 0
Line 3: Token COMMA ,
Line 3: Token IDENTIFIER: i
Line 3: Token SEMICOLON ;
Line 4: Token KEYWORD_INTEGER: integer
Line 4: Token IDENTIFIER: ptra
Line 4: Token ASSIGNMENT: :=
Line 4: Token CONST_INTEGER: 4
Line 4: Token SEMICOLON ;
Line 5: Token KEYWORD_FOR: for
Line 5: Token PARENTHESIS_LEFT: (
Line 5: Token IDENTIFIER: i
Line 5: Token ASSIGNMENT: :=
Line 5: Token CONST_INTEGER: 0
Line 5: Token SEMICOLON ;
Line 5: Token IDENTIFIER: i
Line 5: Token OP_LESS: <
Line 5: Token CONST_INTEGER: 100
Line 5: Token SEMICOLON ;
Line 5: Token IDENTIFIER: i
Line 5: Token OP_INCREMENT: ++
Line 5: Token PARENTHESIS_RIGHT: )
Line 6: Token KEYWORD_BEGIN: begin
Line 7: Token IDENTIFIER: a
Line 7: Token BRACKET_LEFT: [
Line 7: Token IDENTIFIER: i
Line 7: Token BRACKET_RIGHT: ]
Line 7: Token ASSIGNMENT: :=
Line 7: Token IDENTIFIER: i
Line 7: Token SEMICOLON ;
Line 8: Token KEYWORD_END: end
Line 10: Token KEYWORD_RETURN: return
Line 10: Token KEYWORD_INTEGER: integer
Syntax error in line 10.
Result: Rejected!
```

Listing 6: Flex output of correct1.fc

```
#include <stdio.h>
#include "fclib.h"

char* x="saas\n\r\t\\\"\'";
 char x='\'';
 int a[100][2][2][5][6];
 int b[100];
 static int ackermann(int m,int n);
 void main() {
 int x=0,i;
 int ptra=4;
 int b[100]=4,y;
 int foo();
 int x=4 / 2 + 5;
 int c[50][60]=2,y;
 for(i = 0; i < 100; i++) a[i] = i;
 for(i = 0; i < 100; i++)
{
 if( i >= 999 && i < 200 && (i != 3 || i > 5) ) a[i + 1] = 1;
 if( b[1] != 3 )
{
 b[i + (ptra % i) / 2] = a[97];
 b[1]++;
 continue;
}

}

 foo(b[11],b[23],b[23 + 5][x % 2],4);
 x = foo(5);
 if( b[1] != 3 )

{
 b[10] = 5;
 if( b[5] > 3 )

{
 b[11] = -6;
 b[12] = -3;
}

 else
{
 b[12] = -3;
}


}
```

```
 else if( b[2] != 3 )

{
 b[11] = -6;
 b[11] = -6;
}

 else b[11] = 5;


 x = ackermann(n,m);
 return x < 5;

}
 int ackermann(int m,int n)
{
 if( !m ) return n + 1;
 if( !n ) return ackermann(m - 1,1);
 return ackermann(m - 1,ackermann(m,n - 1));
}
```

4 Επίλογος

Μέσω της συγκεκριμένης εργασίας, κατανοήθηκε καλύτερα η χρήση των εργαλείων Flex και Bison καθώς επίσης και η μεταξύ τους συνεργασία για την δημιουργία του transpiler μας. Είναι πολύ πιθανό να υπάρχουν περιπτώσεις που ο συγκεκριμένος transpiler να μην μπορέσει να βγάλει κάποιο αποτέλεσμα λόγω κάποιου λάθους ή κάποιου κανόνα που δεν έχει συμπεριληφθεί. Επομένως, πιθανόν να υπάρχουν περιθώρια βελτίωσης.

Βιβλιογραφία

- Flex 2.6.4 Manual
- Bison 3.04 Manual
- ANSI C grammar (Lex)
- ANSI C grammar (Yacc)
- Θεωρία Μαθήματος, Μ. Γ. Λαγουδάκης
- Σημειώσεις Flex και Bison, Γ. Ανέστης