ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
UNIVERSITY OF WEST ATTICA

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΜΕΡΟΣ Β

ΔΗΜΙΟΥΡΓΙΑ ΣΥΝΤΑΚΤΙΚΟΥ ΑΝΑΛΥΤΗ ΜΕ ΤΗ ΓΕΝΝΗΤΡΙΑ BISON

ΣΤΟΙΧΕΙΑ ΟΜΑΔΑΣ / ΕΡΓΑΣΙΑΣ

ΑΡΙΘΜΟΣ ΟΜΑΔΑΣ: 2
ΣΤΟΙΧΕΙΑ ΦΟΙΤΗΤΗ 1: Αθανασίου Βασίλειος Ευάγγελος (ΠΑΔΑ-19390005)
ΣΤΟΙΧΕΙΑ ΦΟΙΤΗΤΗ 2: Θεοχάρης Γεώργιος (ΠΑΔΑ-19390283)
ΣΤΟΙΧΕΙΑ ΦΟΙΤΗΤΗ 3: Τάτσης Παντελής (ΠΑΔΑ-20390226)
ΣΤΟΙΧΕΙΑ ΦΟΙΤΗΤΗ 4: Ηλιού Ιωάννης (ΠΑΔΑ-19390066)
ΣΤΟΙΧΕΙΑ ΦΟΙΤΗΤΗ 5: Δομινάρης Βασίλειος (ΠΑΔΑ-21390055)

ΤΜΗΜΑ ΕΡΓΑΣΤΗΡΙΟΥ: Β1 Τετάρτη 12:00-14:00
ΥΠΕΥΘΥΝΟΣ ΕΡΓΑΣΤΗΡΙΟΥ: Ιορδανάκης Μιχάλης
ΠΡΟΘΕΣΜΙΑ ΥΠΟΒΟΛΗΣ: 19/06/2024
ΗΜΕΡΟΜΗΝΙΑ ΟΛΟΚΛΗΡΩΣΗΣ: 19/06/2024

ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

ΠΕΡΙΕΧΟΜΕΝΑ

Εισαγωγή

Αναφορά

Το μέρος "Β Δημιουργία Συντακτικού Αναλυτή με τη γεννήτρια BISON" αποτελεί το τελευταίο στάδιο κατασκευής του μεταγλωττιστή της γλώσσας Uni-C. Η αναφορά εστιάζει στην τεκμηρίωση της γραμματικής για λεκτική, συντακτική και σημασιολογική ανάλυση, καθώς και στον σχεδιασμό. Ο λεκτικός αναλυτής που κατασκευάστηκε με τη γεννήτρια FLEX στο μέρος "Α3 Συμπλήρωση πρότυπου κώδικα FLEX" συνδέεται με τον συντακτικό αναλυτή που κατασκευάστηκε με τη γεννήτρια BISON. Οι κώδικες που δίνονται ως είσοδοι στις γεννήτριες παρουσιάζονται επαρκώς σχολιασμένοι και τεκμηριωμένοι. Επίσης, παρουσιάζονται και εξαντλητικοί έλεγχοι με σχολιασμένα αποτελέσματα για να αποδειχθεί η αξιοπιστία του μεταγλωττιστή. Τέλος, παρουσιάζεται και η ρητή αναφορά ως προς την εκτέλεση του κώδικα.

Οργάνωση ομάδας

Μέσα από το κλίμα συνεργασίας, όλα τα μέλη της ομάδας συνέβαλλαν στην συμπλήρωση του κώδικα BISON, ώστε να γίνει αντιληπτή η λειτουργία του. Οι εργασίες για την συγγραφή του έγγραφου τεκμηρίωσης μοιράστηκαν όπως και στο μέρος Α σε υπο-ομάδες, εργασίες όπως είναι ο σχολιασμός του κώδικα, οι επιλογές των δοκιμών εισόδου, αλλά και ο εκτενής σχολιασμός των αποτελεσμάτων. Οι εργασίες για την συγγράφη του κώδικα BISON και της προσαρμογής του κώδικα FLEX, ώστε να συνεργάζεται με τον BISON έγιναν απ' όλα τα μέλη της ομάδας.

Περιεχόμενο αναφοράς

Η παρουσίαση εστιάζει στην λειτουργικότητα του συντακτικού αναλυτή με τη γεννήτρια BISON, η οποία αναγνωρίζει τις εκφράσεις της γλώσσας Uni-C. Τα κεφάλαια αναφοράς είναι τα εξής:

1. Τεκμηρίωση της γραμματικής και του σχεδιασμού
2. Τεκμηρίωση του κώδικα FLEX και BISON
3. Περιπτώσεις ελέγχου και σχολιασμός

Επιπρόσθετα, για κάθε εργασία αναγράφονται:

1. Ποια υπο-ομάδα την υλοποίησε
2. Σχολιασμός των περιπτώσεων ελέγχου
3. Σχόλια όσον αφορά τον τρόπο λειτουργίας της προσομοίωσης, τυχόν δυσκολίες που προέκυψαν και τρόποι αντιμετώπισης
4. Αναφορά τυχόν ελλείψεων και ορθότητας ως προς την εκτέλεση του κώδικα

ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

Ανάλυση αρμοδιοτήτων/ρόλων όλων των μελών της ομάδας

Αθανασίου Βασίλειος Ευάγγελος (ΠΑΔΑ-19390005)

Στον ρόλο του γενικού συντονιστή, ο φοιτητής συμμετείχε στις εργασίες:

1. **Κώδικας FLEX**

   - Προσαρμογή για την συνεργασία με τον κώδικα BISON
   - Διαχείριση λανθασμένων συμβολοσειρών

2. **Κώδικας BISON**

   - Συγγραφή συντακτικών κανόνων
   - Διαχείριση λανθασμένων εκφράσεων
   - Διαχείριση συντακτικών προειδοποιητικών λαθών (warnings)
   - Επιλογή δοκιμαστικών εκτελέσεων

3. **Έγγραφο τεκμηρίωσης**

   - Εισαγωγή
   - Τεκμηρίωση της γραμματικής και του σχεδιασμού
   - Ρητή αναφορά εκτέλεσης κώδικα
   - Στην υλοποίηση της τελικής έκδοσης του εγγράφου

Στην υλοποίηση των τελικών εκδόσεων των FLEX και BISON

Θεοχάρης Γεώργιος (ΠΑΔΑ-19390283)

Στον ρόλο του προγραμματιστή, ο φοιτητής συμμετείχε στις εργασίες:

1. **Κώδικας FLEX**

   - Προσαρμογή για την επικοινωνία με τον κώδικα BISON
   - Διαχείριση λανθασμένων συμβολοσειρών

2. **Κώδικας BISON**

   - Συγγραφή συντακτικών κανόνων
   - Διαχείριση λανθασμένων εκφράσεων
   - Διαχείριση συντακτικών προειδοποιητικών λαθών (warnings)
   - Επιλογή δοκιμαστικών εκτελέσεων

3. **Έγγραφο τεκμηρίωσης**

- Τεκμηρίωση του FLEX
- 2.5 Δήλωση συναρτήσεων χρήστη
- 2.6 Δηλώσεις απλών εκφράσεων

Τάτσης Παντελής (ΠΑΔΑ-20390226)

Στον ρόλο του δοκιμαστή (tester), ο φοιτητής συμμετείχε στις εργασίες:

1. **Κώδικας FLEX**

- Προσαρμογή για την επικοινωνία με τον κώδικα BISON
- Διαχείριση λανθασμένων συμβολοσειρών

2. **Κώδικας BISON**

- Συγγραφή συντακτικών κανόνων
- Διαχείριση λανθασμένων εκφράσεων
- Διαχείριση συντακτικών προειδοποιητικών λαθών (warnings)
- Επιλογή δοκιμαστικών εκτελέσεων

3. **Έγγραφο τεκμηρίωσης**

- 2.3 Πίνακες
- 2.4 Ενσωματωμένες απλές συναρτήσεις
- 2.10 Ενιαίο

Ηλιού Ιωάννης (ΠΑΔΑ-19390066)

Στον ρόλο του σχολιαστή, ο φοιτητής συμμετείχε στις εργασίες:

1. **Κώδικας FLEX**

- Προσαρμογή για την επικοινωνία με τον κώδικα BISON
- Διαχείριση λανθασμένων συμβολοσειρών

2. **Κώδικας BISON**

- Συγγραφή συντακτικών κανόνων
- Διαχείριση λανθασμένων εκφράσεων
- Διαχείριση συντακτικών προειδοποιητικών λαθών (warnings)

- Επιλογή δοκιμαστικών εκτελέσεων

3. **Έγγραφο τεκμηρίωσης**

- 2.2 Δηλώσεις Μεταβλητών
- 2.8 Συντακτικά Προειδοποιητικά λάθη
- 2.9 Συντακτικά Λάθη

Δομινάρης Βασίλειος (ΠΑΔΑ-21390055)

Στον ρόλο του αναλυτή, ο φοιτητής συμμετείχε στις εργασίες:

1. **Κώδικας FLEX**

- Προσαρμογή για την επικοινωνία με τον κώδικα BISON
- Διαχείριση λανθασμένων συμβολοσειρών

2. **Κώδικας BISON**

- Συγγραφή συντακτικών κανόνων
- Διαχείριση λανθασμένων εκφράσεων
- Διαχείριση συντακτικών προειδοποιητικών λαθών (warnings)
- Επιλογή δοκιμαστικών εκτελέσεων

3. **Έγγραφο τεκμηρίωσης**

- Τεκμηρίωση του BISON
- 2.7 Σύνθετες Δηλώσεις
- Σχολιασμός Στατιστικών Στοιχείων Ανάλυσης

Τεκμηρίωση της γραμματικής και του σχεδιασμού

Με κόκκινα γράμματα αποτυπώνονται οι αρχές γραμματικής που προδιαγραφεί η Uni-C αλλά και οι αρχές σχεδιασμού που προδιαγραφεί η κατασκευή ενός μεταγλωττιστή της γλώσσας Uni-C, όπου η ομάδα δεν κατάφερε να υλοποιήσει.

Γραμματική

Η τεκμηρίωση της γραμματικής για μια γλώσσα προγραμματισμού όπως η Uni-C, η οποία αποτελεί υποσύνολο της γλώσσας C, περιλαμβάνει την ανάλυση των στοιχείων της γλώσσας σε λεκτικό, συντακτικό και σημασιολογικό επίπεδο. Ακολουθεί αναλυτική περιγραφή για κάθε επίπεδο:

1. Λεκτική Ανάλυση

Η λεκτική ανάλυση περιλαμβάνει τον εντοπισμό λεκτικών μονάδων της γλώσσας Uni-C όπως:

- **Αναγνωριστικά (Identifiers):** Χρησιμοποιούνται για την ονομασία μεταβλητών, συναρτήσεων κλπ. (π.χ. var, foo).
- **Λέξεις-κλειδιά (Keywords):** Πρόκειται για δεσμευμένες λέξεις που έχουν ειδική σημασία στη γλώσσα Uni-C και γι' αυτό το λόγο απαγορεύεται να χρησιμοποιηθούν ως όνομα μεταβλητής του χρήστη ή ως όνομα δικής του συνάρτησης (π.χ. if, else, while)
- **Λεκτικά κυριολεκτικά (Strings):** Πρόκειται για απλές συμβολοσειρές οι οποίες περικλείονται μέσα σε διπλές αποστρόφους και περιλαμβάνουν οποιονδήποτε χαρακτήρα (π.χ. "Hello", "World").
- **Ακέραιοι (Integers):** Πρόκειται για σταθερές ακέραιες αριθμητικές τιμές που χρησιμοποιούνται σε κάποια μεταβλητή για την εκτέλεση πράξεων (αριθμητικών, σχεσιακών, λογικών) ή για ανάθεση σε κάποια μεταβλητή (π.χ. 1, 10, 50)
- **Αριθμοί κινούμενης υποδιαστολής (Floats):** Πρόκειται για σταθερές αριθμητικές τιμές που διαχωρίζονται με υποδιαστολή ή εκφράζονται σε μορφή δύναμης με την επιστημονική μορφή του εκθέτη δύναμης "E ή e" (π.χ. 3.14, 4e1, 3.5e-1)
- **Τελεστές (Operators):** Χρησιμοποιούνται για την υλοποίηση αριθμητικών, σχεσιακών και λογικών πράξεων (π.χ. >, +, &&)
- **Διαχωριστές (Delimiters):** Χρησιμοποιούνται για να διαχωρίσουν δύο ή και περισσότερες εντολές της Uni-C (π.χ. ;)
- **Ειδικοί χαρακτήρες (Special):** Πρόκειται για ειδικούς χαρακτήρες που χρησιμοποιούνται για την σύνταξη κάποιων εκφράσεων όπως την δήλωση πινάκων που χρησιμοποιείται ο ειδικός χαρακτήρας [και].

  Η λεκτική ανάλυση επίσης αναγνωρίζει και άλλες λεκτικές δομές όπως τα σχόλια και τους white_space χαρακτήρες, ωστόσο, δεν τα επιστρέφει ως λεκτικές μονάδες και απλά τα αγνοεί.

  Τέλος, η λεκτική ανάλυση μεριμνεί και την κατάσταση που αναγνωριστεί κάποια συμβολοσειρά εισόδου που δεν ταιριάζει με τους προαναφερόμενους λεκτικούς κανόνες. Η συμβολοσειρά αυτή αναγνωρίζεται ως TOKEN_ERROR και είναι στην κρίση της συντακτικής ανάλυσης το πως θα διαχειριστεί αυτή τη συμβολοσειρά.

2. Συντακτική Ανάλυση

Η συντακτική ανάλυση αφορά τη δομή των προγραμμάτων και τους κανόνες που διέπουν τη σύνταξη της Uni-C όπως:

- **Λογικές Γραμμές:** Πρόκειται για εκφράσεις που ολοκληρώνονται με τον διαχωριστή semi-colon (;).
- **Φυσικές Γραμμές:** Πρόκειται για μία ακολουθία χαρακτήρων που ολοκληρώνεται με τον χαρακτήρα αλλαγή γραμμής (\n)
- **Σύνδεση φυσικών γραμμών:** Πρόκειται για την σύνδεση 2 φυσικών γραμμών με τον χαρακτήρα backslash (\) στο τέλος της πρώτης γραμμής.
- **Κενές γραμμές:** Πρόκειται για μία γραμμή που δεν περιλαμβάνει λεκτικές μονάδες που αναγνωρίζει η λεκτική ανάλυση.
- **Δηλώσεις μεταβλητών:** Πρόκειται για εντολή που ο χρήστης δηλώνει μία μεταβλητή με χρήση μίας λέξης-κλειδί που αποτυπώνει τον τύπο δεδομένων της μεταβλητής (π.χ. float, int, double, short, long)
- **Πίνακες:** Πρόκειται για μια δομή δεδομένων που αποθηκεύει μεταβλητές, σταθερές αριθμητικές τιμές και συμβολοσειρές.
- **Ενσωματωμένες απλές συναρτήσεις:** Πρόκειται για ενσωματωμένες συναρτήσεις που ορίζονται στη Uni-C και έχουν ειδική λειτουργία με ειδική σύνταξη (π.χ. scan, len, cmp, print)
- **Δήλωση συναρτήσεων χρήστη:** Πρόκειται συναρτήσεις που μπορεί ο χρήστης να ορίσει με δικές του παραμέτρους, δικό του όνομα και δικό του κώδικα αρκεί να ξεκινάει με τη λέξη-κλειδί func.
- **Δηλώσεις απλών εκφράσεων:** Πρόκειται για αριθμητικές εκφράσεις, αναθέσεις τιμών σε μεταβλητές, συγκρίσεις και συνένωσης πινάκων.
- **Σύνθετες δηλώσεις:** Πρόκειται για δηλώσεις ελέγχου με if και δηλώσεις επανάληψης με while και for

3. Σημασιολογική ανάλυση

Η σημασιολογική ανάλυση αφορά τη σημασία και την ερμηνεία των προγραμμάτων στη Uni-C με βάση σημασιολογικούς κανόνες όπως

- **Έλεγχος Τύπων:** Επιβεβαίωση ότι οι πράξεις εκτελούνται μεταξύ συμβατών τύπων δεδομένων (π.χ. ακέραιοι με ακεραίους)
- **Εύρος Μεταβλητών:** Καθορίζει την ορατότητα και τη διάρκεια ζωής των μεταβλητών
- **Εύρος Συναρτήσεων:** Καθορίζει την ορατότητα μιας συνάρτησης και πόσες παραμέτρους και τι τύπου δεδομένων παραμέτρους μπορεί να δεχτεί
- **Στοιχεία Πίνακα:** Καθορίζει τα στοιχεία του πίνακα να είναι ίδιου τύπου δεδομένων

ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

Σχεδιασμός

Ο σχεδιασμός ενός μεταγλωττιστή της Uni-C περιλαμβάνει διάφορα στάδια και υποσυστήματα που συνεργάζονται για να μετατρέψουν τον πηγαίο κώδικα σε εκτελέσιμο αρχείο ή ενδιάμεσο κώδικα. Αναλυτικά ο σχεδιασμός περιλαμβάνει τα εξής στάδια:

1. **Κατασκευή του Λεκτικού Αναλυτή με τη γεννήτρια FLEX**

   Ο λεκτικός αναλυτής διαχωρίζει τον κώδικα σε λεκτικές μονάδες (tokens).

2. **Κατασκευή του Συντακτικού Αναλυτή με τη γεννήτρια BISON**

   Ο συντακτικός αναλυτής ελέγχει τη δομή του κώδικα σύμφωνα με τους συντακτικούς κανόνες.

3. **Κατασκευή Πίνακα Συμβόλων**

   Ο πίνακας συμβόλων περιέχει τα ονόματα που επιλέγει ο προγραμματιστής για τις διάφορες οντότητες, που πρόκειται να επεξεργαστεί ο μεταγλωττιστής και πληροφορίες για αυτά.

4. **Κατασκευή Αφηρημένου Συντακτικού Δένδρου**

   Το αφηρημένο συντακτικό δένδρο κατασκευάζεται συνδέοντας τα τερματικά ή μη τερματικά σύμβολα που προκύπτουν από μια παραγωγή με το μη-τερματικό σύμβολο από το οποίο προκύπτουν

5. **Παραγωγή Ενδιάμεσου Κώδικα**

   Ο ενδιάμεσος κώδικας είναι μια μορφή κώδικα ανάμεσα στο επίπεδο του πηγαίου κώδικα και της γλώσσας μηχανής και χρησιμοποιείται για να αφήσει περιθώρια βελτιστοποίησης.

Τεκμηρίωση του κώδικα FLEX και BISON

Λεκτικός Αναλυτής (FLEX)

Ο παρακάτω κώδικας υλοποιεί ένα Flex scanner για την αναγνώριση διάφορων token (λέξεις κλειδιά, τελεστές, αναγνωριστικά, κ.λπ.) από ένα αρχείο εισόδου και την αναφορά πιθανών σφαλμάτων.

Περιοχή Δηλώσεων

```
/* Η ανάγνωση περιορίζεται σε ένα μόνο αρχείο και τερματίζεται στο πρώτο EOF */
%option noyywrap
%x error

/* Κώδικας C για τον ορισμό των απαιτούμενων header files και των μεταβλητών.
   Οτιδήποτε ανάμεσα στα %{ και %} μεταφέρεται αυτόματα στο αρχείο C που
   θα δημιουργήσει το Flex. */

%{

#include <stdio.h>
#include <string.h>
#include <stdlib.h>


/* Αρχείο κεφαλίδας που περιέχει λίστα με όλα τα tokens */
#include "simple-bison-code.tab.h"

/* Ορισμός μετρητή τρέχουσας γραμμής */
extern int line;
extern int flag;

extern int correct_words;
extern int lex_warnings;

void yyerror (const char *msg);
void prn(char *token);

%}
```

Περιλαμβάνει τα απαραίτητα αρχεία κεφαλίδας για τις λειτουργίες εισόδου/εξόδου και χειρισμού συμβολοσειρών.

Δηλώνει εξωτερικές μεταβλητές και συναρτήσεις που χρησιμοποιούνται στο υπόλοιπο του προγράμματος, καθώς και στο πρόγραμμα του ΣΑ.

ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

Περιοχή Ορισμών

```
/* Ονόματα και αντίστοιχοι ορισμοί (υπό μορφή κανονικής έκφρασης).
   Μετά από αυτό, μπορεί να γίνει χρήση των ονομάτων (αριστερά) αντί των,
   συνεπώς ιδιαίτερα μακροσκελών και δυσνόητων, κανονικών εκφράσεων */

IDENTIFIER      [a-zA-Z_][a-zA-Z0-9_]{0,31}
STRING          \"([^"\\]*(\\[\\n"][^"\\]*)*)\"
INTEGER         ([1-9][0-9]*|0[x|X][0-9A-F]+|0[0-7]+|0)
FLOAT           (?:[1-9][0-9]*|0)(?:\.(?:[1-9][0-9]*|0*[1-9]+))?(?:[eE](?:-?[1-
9][0-9]*|0))?
COMMENT         (\/\/.*|\/\*[^*]*\*+([^/*][^*]*\*+)*\/)
DELIMITER       \;
WHITESPACE      [ \t]+
NEWLINE         \n
TOKEN_ERROR     [^ \t\r\n;]


%%
```

Στην περιοχή ορισμών, ορίζονται οι κανονικές εκφράσεις για διάφορες λεκτικές μονάδες όπως αναγνωριστικά, συμβολοσειρές, ακέραιους, αριθμοί κινητής υποδιαστολής, σχόλια, διαχωριστικά, κενά και νέα γραμμή.

Περιοχή Κανόνων

```
"break"         { correct_words++; prn("KEYWORD"); return SBREAK; }
"case"          { correct_words++; prn("KEYWORD"); return SCASE; }
"const"         { correct_words++; prn("KEYWORD"); return SCONST; }
"continue"      { correct_words++; prn("KEYWORD"); return SCONTINUE; }
"do"            { correct_words++; prn("KEYWORD"); return SDO; }
"double"        { correct_words++; prn("KEYWORD"); return SDOUBLE; }
"else"          { correct_words++; prn("KEYWORD"); return SELSE; }
"float"         { correct_words++; prn("KEYWORD"); return SFLOAT; }
"for"           { correct_words++; prn("KEYWORD"); return SFOR; }
"if"            { correct_words++; prn("KEYWORD"); return SIF; }
"int"           { correct_words++; prn("KEYWORD"); return SINT; }
"long"          { correct_words++; prn("KEYWORD"); return SLONG; }
"return"        { correct_words++; prn("KEYWORD"); return SRETURN; }
"sizeof"        { correct_words++; prn("KEYWORD"); return SSIZEOF; }
"struct"        { correct_words++; prn("KEYWORD"); return SSTRUCT; }
"switch"        { correct_words++; prn("KEYWORD"); return SSWITCH; }
"void"          { correct_words++; prn("KEYWORD"); return SVOID; }
"while"         { correct_words++; prn("KEYWORD"); return SWHILE; }
"func"          { correct_words++; prn("KEYWORD"); return SFUNC; }
"short"         { correct_words++; prn("KEYWORD"); return SSHORT; }
"+"             { prn("OPERATOR"); return PLUS; }
"*="            { prn("OPERATOR"); return MULEQ; }
"--"            { prn("OPERATOR"); return PMINEQ; }
"-"             { prn("OPERATOR"); return MINUS; }
```

```
"/="            { prn("OPERATOR"); return DIVEQ; }
"<"             { prn("OPERATOR"); return LT; }
"*"             { prn("OPERATOR"); return MUL; }
"!"             { prn("OPERATOR"); return NOT; }
">"             { prn("OPERATOR"); return GT; }
"/"             { prn("OPERATOR"); return DIV; }
"&&"            { prn("OPERATOR"); return AND; }
"<="            { prn("OPERATOR"); return LEQ; }
"%"             { prn("OPERATOR"); return MOD; }
"||"            { prn("OPERATOR"); return OR; }
">="            { prn("OPERATOR"); return GREQ; }
"="             { prn("OPERATOR"); return ASSIGN; }
"=="            { prn("OPERATOR"); return EQUAL; }
"&"             { prn("OPERATOR"); return ADDR; }
"+="            { prn("OPERATOR"); return PLUSEQ; }
"!="            { prn("OPERATOR"); return NOTEQ; }
"-="            { prn("OPERATOR"); return MINEQ; }
"++"            { prn("OPERATOR"); return PPLUSEQ; }
"("             { prn("SPECIAL"); return OPENPAR; }
")"             { prn("SPECIAL"); return CLOSEPAR; }
"["             { prn("SPECIAL"); return OPENSQBRA; }
"]"             { prn("SPECIAL"); return CLOSESQBRA; }
"{"             { prn("SPECIAL"); return OPENCURBRA; }
"}"             { prn("SPECIAL"); return CLOSECURBRA; }
","             { prn("SPECIAL"); return COMMA; }
"\\"            { prn("SPECIAL"); return BACKSLASH; }
"scan"          { correct_words++; prn("FUNCTION"); return SSCAN; }
"len"           { correct_words++; prn("FUNCTION"); return SLEN; }
"print"         { correct_words++; prn("FUNCTION"); return SPRINT; }
"cmp"           { correct_words++; prn("FUNCTION"); return SCMP; }
{DELIMITER}     { prn("DELIMITER"); return DELIMITER; }
{IDENTIFIER}    { correct_words++; prn("IDENTIFIER"); return IDENTIFIER; }
{STRING}        { correct_words++; prn("STRING"); return STRING; }
{INTEGER}       { prn("INTEGER"); return INTEGER; }
{FLOAT}         { prn("FLOAT"); return FLOAT; }
{COMMENT}       { ECHO; }
{WHITESPACE}    { }
{NEWLINE}       { line++; return NEWLINE; }
<<EOF>>         { return EOF; }


{TOKEN_ERROR}           { lex_warnings++; yyerror("Token error"); BEGIN(error);
return TOKEN_ERROR; }
<error>{WHITESPACE}+    { BEGIN(0); fprintf(yyout, "\t%d character(s) ignored so
far\n", lex_warnings); }
<error>.                { lex_warnings++; }
<error>\n               { line++; BEGIN(0); fprintf(yyout, "\t\t%d character(s)
ignored so far\n", lex_warnings); return NEWLINE;}
```

```
%%
```

Στην περιοχή κανόνων αναγράφονται οι κανόνες που μπορούν να ανιστοιχούν οι συμβολοσειρές εισόδου. Κάθε κανόνας αποτελείται από την κανονίκη έκφραση που ορίστηκε στην περιοχή ορισμών και ενέργειες-εντολές σε C.

Μία ενέργεια που εκτελείαι σε ορισμένους κανόνες για την διαχείριση λαθών είναι η ενημέρωση των μεταβλητών μετρητών **correct_words** για τις σωστές λέξεις και **lex_warnings** για τις λανθασμένες λέξεις. Επίσης, άλλη μία σημαντική ενέργεια που εκτελείται στους κανόνες των λεκτικών μονάδων είναι η εντολή **return** που επιστρέφει το token στον ΣΑ και αποτελεί τον κύριο κώδικα επικοινωνίας ΛΑ και ΣΑ.

Διαχειρίζεται τα σφάλματα με το TOKEN_ERROR και μεταβαίνει σε κατάσταση error όταν εντοπίζεται μη αναγνωρίσιμος χαρακτήρας.

Κώδικας Χρήστη

```
/* Η συνάρτηση yyerror χρησιμοποιείται για την αναφορά σφαλμάτων. Συγκεκριμένα
καλείται
   από την yyparse όταν υπάρξει κάποιο συντακτικό λάθος. Στην παρακάτω περίπτωση η
   συνάρτηση επί της ουσίας τυπώνει μήνυμα λάθους στην οθόνη. */
void yyerror(const char *msg)
{
   fprintf(yyout, "!! %s !! -> at Line=%d\n", msg, line);
   return;
}

void prn(char *token)
{
   fprintf(yyout, "\t[FLEX] Line=%d, token=%s, value=\"%s\"\n", line, token,
yytext);
   return;
}
```

Εδώ ο κώδικας του χρήστη απουσιάζει καθώς ο ΣΑ είναι αυτός που καλεί τον ΛΑ για να αναγνωρίζει συμβολοσειρές. Υπάρχουν δύο συναρτήσεις χειρισμού των συμβολοσειρών:

void yyerror(const char *msg): Τυπώνει μήνυμα σφάλματος με την τρέχουσα γραμμή.

void prn(char *token): Τυπώνει τα αναγνωρισμένα tokens με την τρέχουσα γραμμή και το περιεχόμενο του token.

Ο κώδικας αυτός είναι σχεδιασμένος να αναγνωρίζει διάφορα πρότυπα από την είσοδο και να παράγει τα αντίστοιχα tokens, ενώ παράλληλα παρακολουθεί και αναφέρει σφάλματα και άλλες πληροφορίες, όπως ο αριθμός γραμμής.

Συντακτικός Αναλυτής (BISON)

Αυτός ο κώδικας Bison ορίζει έναν αναλυτή σύνταξης για μια προσαρμοσμένη γλώσσα τη Uni-C. Ο κώδικας αποτελείται από διάφορα βασικά τμήματα: προοίμιο, tokens, κανόνες προτεραιότητας, κανόνες γραμματικής και την κύρια συνάρτηση. Παρακάτω παρατίθεται μια λεπτομερής εξήγηση για κάθε τμήμα.

Περιοχή Δηλώσεων

```
%{
/* Ορισμοί και δηλώσεις γλώσσας C. Οτιδήποτε έχει να κάνει με ορισμό ή αρχικοποίηση
   μεταβλητών & συναρτήσεων, αρχεία header και δηλώσεις #define μπαίνει σε αυτό το
σημείο */

        #include <stdio.h>
        #include <string.h>
        #include <stdlib.h>
        #define YYSTYPE char*
        #define YYDEBUG 1

        int line = 1;
        int errflag = 0;

        extern char *yytext;
        int correct_words = 0;
        int correct_exprs = 0;
        int fatal_errors = 0;
        int par_warnings = 0;
        int lex_warnings = 0;

        int yylex();
        void yyerror(const char *msg);

        /* Ο δείκτης yyin είναι αυτός που "δείχνει" στο αρχείο εισόδου. Εάν δεν
γίνει χρήση
   του yyin, τότε η είσοδος γίνεται αποκλειστικά από το standard input
(πληκτρολόγιο) */

        extern FILE *yyin;
        extern FILE *yyout;
%}
```

Ορισμοί και Δηλώσεις στη γλώσσα C: Περιλαμβάνει τις τυπικές βιβλιοθήκες C, ορίζει τη σταθερά YYSTYPE για σημασιολογικές τιμές και ρυθμίζει και τη σταθερά για debug του προγράμματος με τη σταθερά YYDEBUG.

Global Μεταβλητές: Δηλώνει global μεταβλητές για την παρακολούθηση γραμμών, σφαλμάτων, προειδοποιήσεων και σωστών εκφράσεων.
Εξωτερικές Δηλώσεις: Δηλώνει εξωτερικές (extern) μεταβλητές και συναρτήσεις που χρησιμοποιούνται και στον λεκτικό αναλυτή (FLEX).

Ορισμός Tokens

```
/* Ορισμός των αναγνωρίσιμων λεκτικών μονάδων. */
%token  IDENTIFIER STRING
%token  INTEGER
%token  FLOAT
%token  SBREAK SDO SIF SSIZEOF SCASE SDOUBLE SINT SSTRUCT SFUNC SELSE SLONG SSWITCH
SCONST SFLOAT SRETURN SVOID SCONTINUE SFOR SSHORT SWHILE
%token PLUS "+"
%token MULEQ "*="
%token PMINEQ "--"
%token MINUS "-"
%token DIVEQ "/="
%token LT "<"
%token MUL "*"
%token NOT "!"
%token GT ">"
%token DIV "/"
%token AND "&&"
%token LEQ "<="
%token MOD "%"
%token OR "||"
%token GREQ ">="
%token ASSIGN "="
%token EQUAL "=="
%token ADDR "&"
%token PLUSEQ "+="
%token NOTEQ "!="
%token MINEQ "-="
%token PPLUSEQ "++"
%token OPENPAR "("
%token CLOSEPAR ")"
%token OPENSQBRA "["
%token CLOSESQBRA "]"
%token OPENCURBRA "{"
%token CLOSECURBRA "}"
%token COMMA ","
%token BACKSLASH "\\"
%token DELIMITER ";"
%token SSCAN SPRINT SLEN SCMP
%token NEWLINE
%token TOKEN_ERROR
```

Δηλώσεις Tokens: Παραθέτει όλα τα token που αναγνωρίζονται από τον λεκτικό αναλυτή, με ορισμένα token να συνδέονται με συγκεκριμένες συμβολοσειρές.

Προτεραιότητα Τελεστών

```
/* Ορισμός προτεραιοτήτων στα tokens */
%left ","
%right "*=" "/=" "+=" "-=" "="
%left "||"
%left "&&"
%left "==" "!="
%left "<" ">" "<=" ">="
%left "+" "-"
%left "*" "/" "%"
%right "&" "!"
%left "++" "--"
```

Προτεραιότητα και Συνδετικότητα: Ορίζει την προτεραιότητα και τη συνδετικότητα για τους τελεστές για την επίλυση αμφισημιών.

Έναρξη προγράμματος

```
/* Έναρξη προγραμμάτος*/
%start program

%%
```

Δηλώνει ότι ο κανόνας **program** θα αποτελεί το πρώτο σύμβολο εισόδου στην Στοίβα.

Κανόνες Γραμματικής

```
* Ορισμός των γραμματικών κανόνων. Κάθε φορά που αντιστοιχίζεται ένας γραμματικός
   κανόνας με τα δεδομένα εισόδου, εκτελείται ο κώδικας C που βρίσκεται ανάμεσα στα
   αγκύλια. Η αναμενόμενη σύνταξη είναι:
                                όνομα : κανόνας { κώδικας C } */
program:
        program decl_statements NEWLINE                  { correct_exprs++; if ($2
!= "\n") fprintf(yyout, "[BISON] Line=%d, expression=%s\n\n", line-1, $2); }
        | program error NEWLINE                           { fatal_errors++; errflag
= 1; yyerrok; }
        | program merge_arr TOKEN_ERROR merge_arr NEWLINE { yyerrok; }
        |                                                 { }
        ;
```

Δομή Προγράμματος: ορίζει το '**program**' ως μια σειρά από '**decl_statements**' χωρισμένα με '**NEWLINE**'.

```
/* ============== [2.2] Δηλώσεις Μεταβλητών ============== */
decl_var:
        type var { $$ = strdup(yytext); }
        ;

type:
        SINT              { $$ = strdup(yytext); }
        | SFLOAT          { $$ = strdup(yytext); }
        | SDOUBLE         { $$ = strdup(yytext); }
        | SSHORT          { $$ = strdup(yytext); }
        | SLONG           { $$ = strdup(yytext); }
        /* ## Warning ## -> Έξτρα keyword στην δήλωση μεταβλητών */
        | SFLOAT SFLOAT   { par_warnings++; $$ = strdup(yytext); fprintf(yyout, "##
Warning ## -> Double float detected at Line=%d\n", line); }
        | SDOUBLE SDOUBLE { par_warnings++; $$ = strdup(yytext); fprintf(yyout, "##
Warning ## -> Double double detected at Line=%d\n", line); }
        | SINT SINT       { par_warnings++; $$ = strdup(yytext); fprintf(yyout, "##
Warning ## -> Double int detected at Line=%d\n", line); }
        | SLONG SLONG     { par_warnings++; $$ = strdup(yytext); fprintf(yyout, "##
Warning ## -> Double long detected at Line=%d\n", line); }
        | SSHORT SSHORT   { par_warnings++; $$ = strdup(yytext); fprintf(yyout, "##
Warning ## -> Double short detected at Line=%d\n", line); }
        /* ################################################## */
        ;

var:
        IDENTIFIER     { $$ = strdup(yytext); }
        | var "," var  { $$ = strdup(yytext); }
        ;
```

Δηλώσεις Μεταβλητών: Καθορίζει το τρόπο με τον οποίο δηλώνονται οι μεταβλητές και οι τύποι τους, συμπεριλαμβανόμενων προειδοποιήσεων για διπλές λέξεις-κλειδιά.

```
/* ============== [2.3] Πίνακες ============== */
pos_elem:
        IDENTIFIER "[" INTEGER "]"      { $$ = strdup(yytext); }
        | IDENTIFIER "[" IDENTIFIER "]" { $$ = strdup(yytext); }
        ;


arr_elements:
        "[" "]"           { $$ = strdup(yytext); }
        | "[" integ "]"   { $$ = strdup(yytext); }
        | "[" fl "]"      { $$ = strdup(yytext); }
        | "[" str "]"     { $$ = strdup(yytext); }
        | "[" var "]"     { $$ = strdup(yytext); }
        ;

integ:
```

```
        INTEGER                 { $$ = strdup(yytext); }
        | integ "," integ       { $$ = strdup(yytext); }
        ;

fl:
        FLOAT                   { $$ = strdup(yytext); }
        | fl "," fl             { $$ = strdup(yytext); }
        ;

str:
        STRING                  { $$ = strdup(yytext); }
        | str "," str           { $$ = strdup(yytext); }
        ;
```

Χειρισμός Πινάκων: Καθορίζει κανόνες για τα στοιχεία των πινάκων και τις θέσεις τους.

```
/* ============== [2.4] Ενσωματωμένες απλές συναρτήσεις ============== */
build_func:
        func  { $$ = strdup(yytext); }
        ;

func:
        SSCAN "(" scan_params ")"          { $$ = strdup(yytext); }
        | SLEN "(" len_params ")"          { $$ = strdup(yytext); }
        | SCMP "(" cmp_params ")"          { $$ = strdup(yytext); }
        | SPRINT "(" print_params ")"      { $$ = strdup(yytext); }
        | IDENTIFIER "(" print_params ")" { $$ = strdup(yytext); }
        ;

scan_params:
        IDENTIFIER      { $$ = strdup(yytext); }
        ;

len_params:
        arr_elements    { $$ = strdup(yytext); }
        | STRING        { $$ = strdup(yytext); }
        | IDENTIFIER    { $$ = strdup(yytext); }
        ;

cmp_params:
        STRING                          { $$ = strdup(yytext); }
        | IDENTIFIER                    { $$ = strdup(yytext); }
        | cmp_params "," cmp_params     { $$ = strdup(yytext); }
        ;

print_params:
        STRING                          { $$ = strdup(yytext); }
        | IDENTIFIER                    { $$ = strdup(yytext); }
        | INTEGER                       { $$ = strdup(yytext); }
```

```
        | FLOAT                          { $$ = strdup(yytext); }
        | func                           { $$ = strdup(yytext); }
        | pos_elem                       { $$ = strdup(yytext); }
        | print_params "," print_params { $$ = strdup(yytext); }
        ;
```

Κλήσεις Ενσωματωμένων απλών Συναρτήσεων: Χειρίζεται τους ορισμούς ενσωματωμένων απλών συναρτήσεων της γλώσσας Uni-C όπως είναι η **print, len, cmp, scan**

```
/* ============== [2.5] Δήλωση συναρτήσεων χρήστη ============== */
decl_func:
        name_func decl_statement { $$ = strdup(yytext); }
        ;

name_func:
        SFUNC                                   { $$ = strdup(yytext); }
        /* ## Warning ## -> Τύπος επιστροφής στις συναρτήσεις */
        | SFUNC type                            { par_warnings++; $$ =
strdup(yytext); fprintf(yyout, "## Warning ## -> Return type unnecessary at
Line=%d\n", line); }
        /* ################################################## */
        | name_func IDENTIFIER params NEWLINE   { $$ = strdup(yytext); }
        ;

params:
        "(" ")"                 { $$ = strdup(yytext); }
        |"(" type_params ")"    { $$ = strdup(yytext); }
        ;

type_params:
        type IDENTIFIER                 { $$ = strdup(yytext); }
        | type_params "," type_params   { $$ = strdup(yytext); }
        ;
```

Χειρισμός συναρτήσεων χρήστη: Χειρίζεται τους ορισμούς συναρτήσεων του χρήστη σύμφωνα με τα πρότυπα της γλώσσας Uni-C.

```
/* ============== [2.6] Δηλώσεις απλών εκφράσεων ============== */
/* [2.6.1] Αριθμητικές εκφράσεις */
sign:
        INTEGER         { $$ = strdup(yytext); }
        | FLOAT         { $$ = strdup(yytext); }
        | IDENTIFIER    { $$ = strdup(yytext); }
        | "+" sign      { $$ = strdup(yytext); }
        | "-" sign      { $$ = strdup(yytext); }
        ;

arithm_expr:
        sign                            { $$ = strdup(yytext); }
```

```
        | arithm_expr "+" arithm_expr   { $$ = strdup(yytext); }
        | arithm_expr "-" arithm_expr   { $$ = strdup(yytext); }
        | arithm_expr "*" arithm_expr   { $$ = strdup(yytext); }
        | arithm_expr "/" arithm_expr   { $$ = strdup(yytext); }
        | arithm_expr "%" arithm_expr   { $$ = strdup(yytext); }
        ;

number:
        INTEGER     { $$ = strdup(yytext); }
        | FLOAT     { $$ = strdup(yytext); }
        | pos_elem  { $$ = strdup(yytext); }
        ;


/* [2.6.2] Αναθέσεις τιμών σε μεταβλητή */
assign:
        var "=" val                 { $$ = strdup(yytext); }
        | var "=" cmp_expr          { $$ = strdup(yytext); }
        | var "=" arithm_expr       { $$ = strdup(yytext); }
        | var "=" merge_arr         { $$ = strdup(yytext); }
        | oper_eq                   { $$ = strdup(yytext); }
        ;

oper_eq:
        var "++"                    { $$ = strdup(yytext); }
        | var "--"                  { $$ = strdup(yytext); }
        | "++" var                  { $$ = strdup(yytext); }
        | "--" var                  { $$ = strdup(yytext); }
        | var "+=" val              { $$ = strdup(yytext); }
        | var "-=" val2             { $$ = strdup(yytext); }
        | var "*=" val2             { $$ = strdup(yytext); }
        | var "/=" val2             { $$ = strdup(yytext); }
        ;

val2:
    number              { $$ = strdup(yytext); }
    | IDENTIFIER        { $$ = strdup(yytext); }
    ;

val:
    number              { $$ = strdup(yytext); }
    | IDENTIFIER        { $$ = strdup(yytext); }
    | STRING            { $$ = strdup(yytext); }
    | arr_elements      { $$ = strdup(yytext); }
    | val "," val       { $$ = strdup(yytext); }
    ;

/* [2.6.3] Συγκρίσεις */
cmp_expr:
```

```
        INTEGER                  { $$ = strdup(yytext); }
        | FLOAT                  { $$ = strdup(yytext); }
        | IDENTIFIER             { $$ = strdup(yytext); }
        | cmp_expr ">" cmp_expr  { $$ = strdup(yytext); }
        | cmp_expr "<" cmp_expr  { $$ = strdup(yytext); }
        | cmp_expr "<=" cmp_expr { $$ = strdup(yytext); }
        | cmp_expr ">=" cmp_expr { $$ = strdup(yytext); }
        | cmp_expr "==" cmp_expr { $$ = strdup(yytext); }
        | cmp_expr "!=" cmp_expr { $$ = strdup(yytext); }
        | cmp_expr "||" cmp_expr { $$ = strdup(yytext); }
        | cmp_expr "&&" cmp_expr { $$ = strdup(yytext); }
        | "!" cmp_expr           { $$ = strdup(yytext); }
        /* ## Warning ## -> Διπλό σύμβολο σύγκρισης */
        | cmp_expr ">" ">" cmp_expr { par_warnings++; $$ = strdup(yytext);
fprintf(yyout, "## Warning ## -> Double > detected at Line=%d\n", line-1); }
        | cmp_expr "<" "<" cmp_expr { par_warnings++; $$ = strdup(yytext);
fprintf(yyout, "## Warning ## -> Double < detected at Line=%d\n", line-1); }
        /* ################################################## */
        ;

/* [2.6.4] Συνένωση Πινάκων */
merge_arr:
        arr_elements                        { $$ = strdup(yytext); }
        | merge_arr "+" merge_arr           { $$ = strdup(yytext); }
        /* ## Warning ## -> Άκυροι χαρακτήρες στη συνένωση πινάκων */
        | merge_arr TOKEN_ERROR "+" merge_arr { par_warnings++; $$ = strdup(yytext);
fprintf(yyout, "## Warning ## -> Invalid character in array merge detected at
Line=%d\n", line); }
        | merge_arr "+" TOKEN_ERROR merge_arr { par_warnings++; $$ = strdup(yytext);
fprintf(yyout, "## Warning ## -> Invalid character in array merge detected at
Line=%d\n", line); }
        /* ################################################## */
        ;
```

Χειρισμός Εκφράσεων: Ασχολείται με διάφορες εκφράσεις, συμπεριλαμβανόμενων αριθμητικών, συγκριτικών και εκφράσεων ανάθεσης τιμής σε μεταβλητή.

```
/* ============== [2.7] Σύνθετες δηλώσεις ============== */
decl_statements:
        decl_statement                  { $$ = $1;  }
        | decl_statements decl_statement { $$ = $2;  if ($2 != "\n") fprintf(yyout,
"[BISON] Line=%d, expression=%s\n\n", line-1, $2); }
        ;

decl_statement:
        if_statement                    { $$ = "\"Δήλωση if\""; }
        | while_statement               { $$ = "\"Δήλωση while\""; }
        | for_statement                 { $$ = "\"Δήλωση for\""; }
```

```
        | decl_var ";"                  { $$ = "\"Δήλωση μεταβλητής\""; }
        | build_func ";"                { $$ = "\"Κλήση συνάρτησης\""; }
        | decl_func                     { $$ = "\"Δήλωση συναρτήσεων χρήστη\""; }
        | assign ";"                    { $$ = "\"Ανάθεση τιμής σε μεταβλητή\""; }
        | arithm_expr                   { $$ = "\"Αριθμητική έκφραση\""; }
        | cmp_expr                      { $$ = "\"Σύγκριση\""; }
        | merge_arr                     { $$ = "\"Συνένωση πινάκων\""; }
        | block_statement               { $$ = "\"Σύνθετες δηλώσεις\""; }
        | NEWLINE                       { $$ = "\n"; }
        ;

/* [2.7.1] Η δήλωση if */
if_statement:
        SIF condition decl_statement     { $$ = strdup(yytext); }
        ;

condition:
        cmp_expr               { $$ = strdup(yytext); }
        | "(" condition ")"    { $$ = strdup(yytext); }
        /* ## Warning ## -> Έξτρα παρενθέσεις στις if, while δηλώσεις */
        | "(" "(" condition ")" ")"   { par_warnings++; $$ = strdup(yytext);
fprintf(yyout, "## Warning ## -> Double parethensis detected at Line=%d\n", line); }
        /* ##################################################### */
        ;

block_statement:
        "{" decl_statements "}" { $$ = strdup(yytext); }
        ;

/* [2.7.2] Η δήλωση while */
while_statement:
        SWHILE condition decl_statement { $$ = strdup(yytext); }
        ;

/* [2.7.3] Η δήλωση for */
for_statement:
        SFOR "(" assign ";" cmp_expr ";" oper_eq ")" decl_statement { $$ =
strdup(yytext); }
        ;

%%
```

Χειρισμός Σύνθετων Δηλώσεων: Ασχολείται με την σύνθεση δομών ελέγχου if και δομών επανάληψης while και for.

Κώδικας Χρήστη

```
/* Η συνάρτηση main που αποτελεί και το σημείο εκκίνησης του προγράμματος.
   Στην συγκεκριμένη περίπτωση απλά καλεί τη συνάρτηση yyparse του Bison
   για να ξεκινήσει η συντακτική ανάλυση. */
int main(int argc, char **argv)
{
        yydebug = 0;

        if (argc == 3)
        {
                if (!(yyin = fopen(argv[1], "r")))
                {
                        fprintf(stderr, "Cannot read file: %s\n", argv[1]);
                        return 1;
                }
                if (!(yyout = fopen(argv[2], "w")))
                {
                        fprintf(stderr, "Cannot create file: %s\n", argv[2]);
                        return 1;
                }
        }

        int parse = yyparse();

        fprintf(yyout, "\n\n\t\tΣΤΑΤΙΣΤΙΚΑ ΣΥΝΤΑΚΤΙΚΗΣ ΑΝΑΛΥΣΗΣ\n\n");

        if (errflag == 0 && parse == 0)
                fprintf(yyout, "BISON -> Η συντακτική ανάλυση ολοκλήρωθηκε με
επιτυχία\n");
        else
                fprintf(yyout, "BISON -> Η συντακτική ανάλυση ολοκλήρωθηκε με
αποτυχία\n");

        if (par_warnings > 0)
                fprintf(yyout, "\t\t(με %d warnings)\n\n", par_warnings);


        fprintf(yyout, "\t\tΣΩΣΤΕΣ ΛΕΞΕΙΣ: %d\n", correct_words);
        fprintf(yyout, "\t\tΣΩΣΤΕΣ ΕΚΦΡΑΣΕΙΣ: %d\n", correct_exprs);
        fprintf(yyout, "\t\tΛΑΘΟΣ ΛΕΞΕΙΣ: %d\n", lex_warnings);
        fprintf(yyout, "\t\tΛΑΘΟΣ ΕΚΦΡΑΣΕΙΣ: %d\n", fatal_errors);
        fprintf(yyout, "\n");

        fclose(yyin);
        fclose(yyout);

        return 0;
}
```

Διαδικασία Κύριας Συνάρτησης: Ελέγχει τα ορίσματα γραμμής εντολών, ανοίγει τα αρχεία εισόδου/εξόδου για να διαβάσει τα δεδομένα εισόδου και να εκτυπώσει την ανάλυση από τον ΛΑ και από τον ΣΑ για τις συμβολοσειρές εισόδου. Η ΣΑ εκτελείται με την κλήση της συνάρτησης **yyparse**. Τέλος, τυπώνονται στην έξοδο τα Στατιστικά Στοιχεία Ανάλυσης που περιλαμβάνουν τα αποτελέσματα από την διαχείριση λαθών (σωστές λέξεις, σωστές εκφράσεις, λάθος λέξεις, λάθος εκφράσεις)

Αυτός ο κώδικας Bison ορίζει έναν αναλυτή για την γλώσσα Uni-C με συγκεκριμένα tokens, κανόνες προτεραιότητας, κανόνες γραμματικής και χειρισμό σφαλμάτων. Ο αναλυτής διαβάζει από ένα αρχείο εισόδου και γράφει τα αποτελέσματα ανάλυσης σε ένα αρχείο εξόδου, αναφέροντας τον αριθμό των επεξεργασμένων γραμμών, των κρίσιμων σφαλμάτων, των προειδοποιήσεων και των σωστών εκφράσεων. Ο κώδικας περιλαμβάνει λεπτομερή χειρισμό για μεταβλητές, πίνακες, συναρτήσεις, αριθμητικές και λογικές εκφράσεις, καθώς και δηλώσεις ελέγχου.

Περιπτώσεις ελέγχου και σχολιασμός

2.2 Δηλώσεις Μεταβλητών

Αρχείο Εισόδου (input.txt)

```
int a;

double var;

float c;

int a, b, c;

long var1,var2,var3,var4;

short sh1;
```

Αρχείο Εξόδου (output.txt)

```
	[FLEX] Line=3, token=KEYWORD, value="int"

	[FLEX] Line=3, token=IDENTIFIER, value="a"

	[FLEX] Line=3, token=DELIMITER, value=";"


[BISON] Line=3, expression="Δήλωση μεταβλητής"
```

Η γραμμή αυτή περιέχει μια απλή δήλωση μεταβλητής τύπου int με όνομα a. Ο lexer (FLEX) εντοπίζει σωστά τα tokens: λέξη-κλειδί int, αναγνωριστικό a, και τον οριοθέτη ;. Το parser (BISON) αναγνωρίζει αυτή τη δήλωση ως "Δήλωση μεταβλητής".

```
	[FLEX] Line=4, token=KEYWORD, value="double"

	[FLEX] Line=4, token=IDENTIFIER, value="var"

	[FLEX] Line=4, token=DELIMITER, value=";"


[BISON] Line=4, expression="Δήλωση μεταβλητής"
```

Αυτή η γραμμή δηλώνει μια μεταβλητή τύπου double με όνομα var. Τα tokens ανιχνεύονται σωστά και ο parser αναγνωρίζει τη δήλωση ως "Δήλωση μεταβλητής"

```
	[FLEX] Line=5, token=KEYWORD, value="float"

	[FLEX] Line=5, token=IDENTIFIER, value="c"

	[FLEX] Line=5, token=DELIMITER, value=";"


[BISON] Line=5, expression="Δήλωση μεταβλητής"
```

Εδώ δηλώνεται μια μεταβλητή τύπου float με όνομα c. Τα tokens ανιχνεύονται σωστά και ο parser αναγνωρίζει τη δήλωση ως "Δήλωση μεταβλητής".

```
	[FLEX] Line=6, token=KEYWORD, value="int"

	[FLEX] Line=6, token=IDENTIFIER, value="a"

	[FLEX] Line=6, token=SPECIAL, value=","

	[FLEX] Line=6, token=IDENTIFIER, value="b"
```

```
      [FLEX] Line=6, token=SPECIAL, value=","

      [FLEX] Line=6, token=IDENTIFIER, value="c"

      [FLEX] Line=6, token=DELIMITER, value=";"


[BISON] Line=6, expression="Δήλωση μεταβλητής"
```

Αυτή η γραμμή δηλώνει πολλές μεταβλητές τύπου int (a, b, c). Τα tokens ανιχνεύονται σωστά: λέξη-κλειδί int, αναγνωριστικά a, b, c, και οι οριοθέτες , και ;. Ο parser αναγνωρίζει τη δήλωση ως "Δήλωση μεταβλητής".

```
      [FLEX] Line=7, token=KEYWORD, value="long"

      [FLEX] Line=7, token=IDENTIFIER, value="var1"

      [FLEX] Line=7, token=SPECIAL, value=","

      [FLEX] Line=7, token=IDENTIFIER, value="var2"

      [FLEX] Line=7, token=SPECIAL, value=","

      [FLEX] Line=7, token=IDENTIFIER, value="var3"

      [FLEX] Line=7, token=SPECIAL, value=","

      [FLEX] Line=7, token=IDENTIFIER, value="var4"

      [FLEX] Line=7, token=DELIMITER, value=";"


[BISON] Line=7, expression="Δήλωση μεταβλητής"
```

Αυτή η γραμμή δηλώνει πολλές μεταβλητές τύπου long (var1, var2, var3, var4). Τα tokens ανιχνεύονται σωστά: λέξη-κλειδί long, αναγνωριστικά var1, var2, var3, var4, και οι οριοθέτες , και ;. Το parser αναγνωρίζει τη δήλωση ως "Δήλωση μεταβλητής".

```
      [FLEX] Line=8, token=KEYWORD, value="short"

      [FLEX] Line=8, token=IDENTIFIER, value="sh1"

      [FLEX] Line=8, token=DELIMITER, value=";"


[BISON] Line=8, expression="Δήλωση μεταβλητής"
```

Εδώ δηλώνεται μια μεταβλητή τύπου short με όνομα sh1. Τα tokens ανιχνεύονται σωστά και το parser αναγνωρίζει τη δήλωση ως "Δήλωση μεταβλητής".

2.3 Πίνακες

Αρχείο Εισόδου (input.txt)

```
pin1 = [1, 2, 3, 4, 5 ];

pin2 = ["a", "b", "c", "d"];

pin3 = [4.5, 4e1, 4E-1, 0e0, 5.67];

pin4 = [a, b, c, d];
```

Αρχείο Εξόδου (output.txt)

Οι παραπάνω είσοδοι βλέπουμε ότι αναγνωρίζονται σωστά από τον συντακτικό αναλυτή ως ανάθεση τιμής σε μεταβλητή, καθώς αυτός είναι ο τρόπος που ορίζουμε τους πίνακες στην γλώσσα Uni-C.

```
	[FLEX] Line=12, token=IDENTIFIER, value="pin1"

	[FLEX] Line=12, token=OPERATOR, value="="

	[FLEX] Line=12, token=SPECIAL, value="["

	[FLEX] Line=12, token=INTEGER, value="1"

	[FLEX] Line=12, token=SPECIAL, value=","

	[FLEX] Line=12, token=INTEGER, value="2"

	[FLEX] Line=12, token=SPECIAL, value=","

	[FLEX] Line=12, token=INTEGER, value="3"

	[FLEX] Line=12, token=SPECIAL, value=","

	[FLEX] Line=12, token=INTEGER, value="4"

	[FLEX] Line=12, token=SPECIAL, value=","

	[FLEX] Line=12, token=INTEGER, value="5"

	[FLEX] Line=12, token=SPECIAL, value="]"

	[FLEX] Line=12, token=DELIMITER, value=";"
```

```
[BISON] Line=11, expression="Ανάθεση τιμής σε μεταβλητή"



[BISON] Line=12, expression="Ανάθεση τιμής σε μεταβλητή"
```

Τα tokens ανιχνεύονται ένα προς ένα από τον λεκτικό αναλυτή. Πρόκειται για μία απλή δήλωση πίνακα ακεραίων αριθμών στη μεταβλητή pin1. Το τέλος της εντολής οριοθετείται από το delimiter (;), ενώ τα στοιχεία του πίνακα που βρίσκονται εντός των αγκύλων χωρίζονται από το ‘,’.

```
	[FLEX] Line=13, token=IDENTIFIER, value="pin2"

	[FLEX] Line=13, token=OPERATOR, value="="

	[FLEX] Line=13, token=SPECIAL, value="["

	[FLEX] Line=13, token=STRING, value=""a""

	[FLEX] Line=13, token=SPECIAL, value=","

	[FLEX] Line=13, token=STRING, value=""b""

	[FLEX] Line=13, token=SPECIAL, value=","

	[FLEX] Line=13, token=STRING, value=""c""

	[FLEX] Line=13, token=SPECIAL, value=","

	[FLEX] Line=13, token=STRING, value=""d""

	[FLEX] Line=13, token=SPECIAL, value="]"

	[FLEX] Line=13, token=DELIMITER, value=";"



[BISON] Line=13, expression="Ανάθεση τιμής σε μεταβλητή"
```

Τα tokens ανιχνεύονται από τον λεκτικό αναλυτή και η έκφραση από τον συντακτικό αναλυτή. Είναι μία απλή δήλωση strings εντός ενός πίνακα και η εκχώρηση αυτών των στοιχείων σε μία μεταβλητή-πίνακα την pin2. Οι κανόνες είναι ίδιοι μετα το προηγούμενο παράδειγμα.

```
	[FLEX] Line=14, token=IDENTIFIER, value="pin3"

	[FLEX] Line=14, token=OPERATOR, value="="

	[FLEX] Line=14, token=SPECIAL, value="["

	[FLEX] Line=14, token=FLOAT, value="4.5"
```

```
	[FLEX] Line=14, token=SPECIAL, value=","

	[FLEX] Line=14, token=FLOAT, value="4e1"

	[FLEX] Line=14, token=SPECIAL, value=","

	[FLEX] Line=14, token=FLOAT, value="4E-1"

	[FLEX] Line=14, token=SPECIAL, value=","

	[FLEX] Line=14, token=FLOAT, value="0e0"

	[FLEX] Line=14, token=SPECIAL, value=","

	[FLEX] Line=14, token=FLOAT, value="5.67"

	[FLEX] Line=14, token=SPECIAL, value="]"

	[FLEX] Line=14, token=DELIMITER, value=";"


[BISON] Line=14, expression="Ανάθεση τιμής σε μεταβλητή"
```

Τα tokens πάλι εγκρίνονται σωστά από τον λεκτικό αναλυτή και το. Αυτή τη φορά είναι η δήλωση ενός πίνακα με float τιμές, σε μία μεταβλητή pin3. Η οριοθέτηση είναι παρόμοια με τα προηγούμενα παραδείγματα του κανόνα αυτού.

```
	[FLEX] Line=15, token=IDENTIFIER, value="pin4"

	[FLEX] Line=15, token=OPERATOR, value="="

	[FLEX] Line=15, token=SPECIAL, value="["

	[FLEX] Line=15, token=IDENTIFIER, value="a"

	[FLEX] Line=15, token=SPECIAL, value=","

	[FLEX] Line=15, token=IDENTIFIER, value="b"

	[FLEX] Line=15, token=SPECIAL, value=","

	[FLEX] Line=15, token=IDENTIFIER, value="c"

	[FLEX] Line=15, token=SPECIAL, value=","

	[FLEX] Line=15, token=IDENTIFIER, value="d"

	[FLEX] Line=15, token=SPECIAL, value="]"

	[FLEX] Line=15, token=DELIMITER, value=";"
```

```
[BISON] Line=15, expression="Ανάθεση τιμής σε μεταβλητή"
```

Η αναγνώριση των tokens γίνεται από τον λεκτικό αναλυτή. Οι τιμές που δηλώνουμε τώρα σε μία μεταβλητή-πίνακα 3 είναι άλλες μεταβλητές. Η οριοθέτηση είναι κοινή με τα προηγούμενα παραδείγματα ωστόσο δεν ξέρουμε αν όλες οι μεταβλητές είναι ίδιου τύπου. Δεν μας ενδιαφέρει ωστόσο αυτό καθώς ασχολούμαστε μέχρι το επίπεδο της αναγνώρισης του συντακτικού κανόνα.

2.4 Ενσωματωμένες απλές συναρτήσεις

Αρχείο Εισόδου (input.txt)

```
/* --- Test case [#2.4.1] : SCAN --- */

scan(x);
scan(MyVariable);

/* --- Test case [#2.4.2] : LEN --- */

len([10, 20, 30, 40, 50]);
len("This is a string");
len(StringVariable);

/* --- Test case [#2.4.3] : CMP --- */

cmp("test", "best");
cmp(str1, str2);

/* --- Test case [#2.4.4] : PRINT --- */

```

```
print("Hello World");

print(x, "=", 100);

print(cmp(str1, str2));

print(len("This is a string"));

print(pin[0]);
```

Αρχείο Εξόδου (output.txt)

Ο συντακτικός αναλυτής αναγνωρίζει την κλήση των ενσωματωμένων συναρτήσεων (scan,len,cmp,print,scan). Οι συντακτικοί κανόνες είναι κοινοί :όνομα συνάρτησης → παρένθεση → περιεχόμενο συνάρτησης → παρένθεση → delimiter(;).

```
/* --- Test case [#2.4.1] : SCAN --- */


      [FLEX] Line=20, token=FUNCTION, value="scan"

      [FLEX] Line=20, token=SPECIAL, value="("

      [FLEX] Line=20, token=IDENTIFIER, value="x"

      [FLEX] Line=20, token=SPECIAL, value=")"

      [FLEX] Line=20, token=DELIMITER, value=";"


[BISON] Line=20, expression="Κλήση συνάρτησης"
```

```
      [FLEX] Line=21, token=FUNCTION, value="scan"

      [FLEX] Line=21, token=SPECIAL, value="("

      [FLEX] Line=21, token=IDENTIFIER, value="MyVariable"

      [FLEX] Line=21, token=SPECIAL, value=")"

      [FLEX] Line=21, token=DELIMITER, value=";"


[BISON] Line=21, expression="Κλήση συνάρτησης"
```

Αναγνώριση των tokens από τον λεκτικό αναλυτή σε 2 συναρτήσεις scan(). Η έκφραση που αναγνωρίζεται είναι η εκτύπωση της τιμής μιας μεταβλητής x (στη γραμμή 20) και της τιμής μιας μεταβλητής MyVariable (στη γραμμή 21). Οι εκφράσεις αναγνωρίζονται ως 'Κλήση συνάρτησης".

```
/* --- Test case [#2.4.2] : LEN --- */


      [FLEX] Line=25, token=FUNCTION, value="len"

      [FLEX] Line=25, token=SPECIAL, value="("

      [FLEX] Line=25, token=SPECIAL, value="["

      [FLEX] Line=25, token=INTEGER, value="10"

      [FLEX] Line=25, token=SPECIAL, value=","

      [FLEX] Line=25, token=INTEGER, value="20"

      [FLEX] Line=25, token=SPECIAL, value=","

      [FLEX] Line=25, token=INTEGER, value="30"

      [FLEX] Line=25, token=SPECIAL, value=","

      [FLEX] Line=25, token=INTEGER, value="40"

      [FLEX] Line=25, token=SPECIAL, value=","

      [FLEX] Line=25, token=INTEGER, value="50"

      [FLEX] Line=25, token=SPECIAL, value="]"

      [FLEX] Line=25, token=SPECIAL, value=")"

      [FLEX] Line=25, token=DELIMITER, value=";"

[BISON] Line=24, expression="Κλήση συνάρτησης"



[BISON] Line=25, expression="Κλήση συνάρτησης"
```

```
      [FLEX] Line=26, token=FUNCTION, value="len"

      [FLEX] Line=26, token=SPECIAL, value="("

      [FLEX] Line=26, token=STRING, value=""This is a string""
```

```
	[FLEX] Line=26, token=SPECIAL, value=")"

	[FLEX] Line=26, token=DELIMITER, value=";"


[BISON] Line=26, expression="Κλήση συνάρτησης"
```

```
	[FLEX] Line=27, token=FUNCTION, value="len"

	[FLEX] Line=27, token=SPECIAL, value="("

	[FLEX] Line=27, token=IDENTIFIER, value="StringVariable"

	[FLEX] Line=27, token=SPECIAL, value=")"

	[FLEX] Line=27, token=DELIMITER, value=";"


[BISON] Line=27, expression="Κλήση συνάρτησης"
```

Αναγνώριση των tokens από τον λεκτικό αναλυτή σε 3 len() συναρτήσεις. Η έκφραση που αναγνωρίζεται είναι η μέτρηση του αριθμού των στοιχείων του πίνακα (γραμμή 25), η μέτρηση των χαρακτήρων του string (γραμμή 26) και τέλος η μέτρηση του πλήθους χαρακτήρων μιας string μεταβλητής (γραμμή 27). Οι εκφράσεις αναγνωρίζονται και αυτές σωστά ως "Κλήση συνάρτησης".

```
/* --- Test case [#2.4.3] : CMP --- */


	[FLEX] Line=31, token=FUNCTION, value="cmp"

	[FLEX] Line=31, token=SPECIAL, value="("

	[FLEX] Line=31, token=STRING, value=""test""

	[FLEX] Line=31, token=SPECIAL, value=","

	[FLEX] Line=31, token=STRING, value=""best""

	[FLEX] Line=31, token=SPECIAL, value=")"

	[FLEX] Line=31, token=DELIMITER, value=";"

[BISON] Line=30, expression="Κλήση συνάρτησης"
```

```
[BISON] Line=31, expression="Κλήση συνάρτησης"
```

```
      [FLEX] Line=32, token=FUNCTION, value="cmp"

      [FLEX] Line=32, token=SPECIAL, value="("

      [FLEX] Line=32, token=IDENTIFIER, value="str1"

      [FLEX] Line=32, token=SPECIAL, value=","

      [FLEX] Line=32, token=IDENTIFIER, value="str2"

      [FLEX] Line=32, token=SPECIAL, value=")"

      [FLEX] Line=32, token=DELIMITER, value=";"


[BISON] Line=32, expression="Κλήση συνάρτησης"
```

Αναγνώριση των tokens από τον λεκτικό αναλυτή σε 2 cmp() συναρτήσεις. Η έκφραση που αναγνωρίζεται είναι η σύγκριση 2 strings (γραμμή 31) και η σύγκριση της τιμής 2 μεταβλητών (γραμμή 32). Οι εκφρασεις αναγνωρίζονται σωστά ως “Κλήση συνάρτησης”.

```
/* --- Test case [#2.4.4] : PRINT --- */


      [FLEX] Line=36, token=FUNCTION, value="print"

      [FLEX] Line=36, token=SPECIAL, value="("

      [FLEX] Line=36, token=STRING, value=""Hello World""

      [FLEX] Line=36, token=SPECIAL, value=")"

      [FLEX] Line=36, token=DELIMITER, value=";"

[BISON] Line=35, expression="Κλήση συνάρτησης"




[BISON] Line=36, expression="Κλήση συνάρτησης"
```

```
      [FLEX] Line=37, token=FUNCTION, value="print"
```

```
   [FLEX] Line=37, token=SPECIAL, value="("

      [FLEX] Line=37, token=IDENTIFIER, value="x"

      [FLEX] Line=37, token=SPECIAL, value=","

      [FLEX] Line=37, token=STRING, value=""=""

      [FLEX] Line=37, token=SPECIAL, value=","

      [FLEX] Line=37, token=INTEGER, value="100"

      [FLEX] Line=37, token=SPECIAL, value=")"

      [FLEX] Line=37, token=DELIMITER, value=";"


[BISON] Line=37, expression="Κλήση συνάρτησης"
```

```
      [FLEX] Line=38, token=FUNCTION, value="print"

      [FLEX] Line=38, token=SPECIAL, value="("

      [FLEX] Line=38, token=FUNCTION, value="cmp"

      [FLEX] Line=38, token=SPECIAL, value="("

      [FLEX] Line=38, token=IDENTIFIER, value="str1"

      [FLEX] Line=38, token=SPECIAL, value=","

      [FLEX] Line=38, token=IDENTIFIER, value="str2"

      [FLEX] Line=38, token=SPECIAL, value=")"

      [FLEX] Line=38, token=SPECIAL, value=")"

      [FLEX] Line=38, token=DELIMITER, value=";"


[BISON] Line=38, expression="Κλήση συνάρτησης"
```

```
      [FLEX] Line=39, token=FUNCTION, value="print"

      [FLEX] Line=39, token=SPECIAL, value="("

      [FLEX] Line=39, token=FUNCTION, value="len"
```

```
      [FLEX] Line=39, token=SPECIAL, value="("

      [FLEX] Line=39, token=STRING, value=""This is a string""

      [FLEX] Line=39, token=SPECIAL, value=")"

      [FLEX] Line=39, token=SPECIAL, value=")"

      [FLEX] Line=39, token=DELIMITER, value=";"


[BISON] Line=39, expression="Κλήση συνάρτησης"
```

```
      [FLEX] Line=40, token=FUNCTION, value="print"

      [FLEX] Line=40, token=SPECIAL, value="("

      [FLEX] Line=40, token=IDENTIFIER, value="pin"

      [FLEX] Line=40, token=SPECIAL, value="["

      [FLEX] Line=40, token=INTEGER, value="0"

      [FLEX] Line=40, token=SPECIAL, value="]"

      [FLEX] Line=40, token=SPECIAL, value=")"

      [FLEX] Line=40, token=DELIMITER, value=";"


[BISON] Line=40, expression="Κλήση συνάρτησης"
```

Αναγνώριση των tokens από τον λεκτικό αναλυτή σε 5 print() συναρτήσεις. Η έκφραση που αναγνωρίζεται είναι η εκτύπωση μιας συμβολοσειράς (γραμμή 36), η εκτύπωση μιας μεταβλητής, χαρακτήρα, τιμής (γραμμή 37). Αναγνωρίζεται επίσης συνάρτηση cmp εντός της έκφρασης print (γραμμή 38), όπως και η συνάρτηση len εντός της print (γραμμή 39). Τέλος, στην γραμμή 40 αναγνωρίζεται η εκτύπωση του πρώτου στοιχείου ενός πίνακα με όνομα pin. Όλες οι παραπάνω εκφράσεις αναγνωρίζονται σωστά ως “Κλήση συνάρτησης”.

2.5 Δήλωση Συναρτήσεων Χρήστη

Αρχείο Εισόδου (input.txt)

```
func myfunc()

{

}

func myfunc2(int paramA, long paramB, short paramC)

{



}

func main()

{

    print("Hello World\n");

}
```

Αρχείο Εξόδου (output.txt)

```
      [FLEX] Line=44, token=KEYWORD, value="func"

      [FLEX] Line=44, token=IDENTIFIER, value="myfunc"

      [FLEX] Line=44, token=SPECIAL, value="("

      [FLEX] Line=44, token=SPECIAL, value=")"



      [FLEX] Line=45, token=SPECIAL, value="{"



      [FLEX] Line=46, token=SPECIAL, value="}"
```

```
[BISON] Line=45, expression="Δήλωση συναρτήσεων χρήστη"



[BISON] Line=46, expression="Δήλωση συναρτήσεων χρήστη"
```

Οι token που αναγνωρίστηκαν από το Flex είναι σωστοί: func ως KEYWORD, myfunc ως IDENTIFIER, τα (,) και {, } ως SPECIAL.

Οι εκφράσεις που αναγνωρίστηκαν από το Bison (Γραμμές 45 και 46) είναι σωστές ως "Δήλωση συναρτήσεων χρήστη", καθώς η συνάρτηση myfunc δηλώθηκε σωστά χωρίς παραμέτρους.

```
      [FLEX] Line=47, token=KEYWORD, value="func"

      [FLEX] Line=47, token=IDENTIFIER, value="myfunc2"

      [FLEX] Line=47, token=SPECIAL, value="("

      [FLEX] Line=47, token=KEYWORD, value="int"

      [FLEX] Line=47, token=IDENTIFIER, value="paramA"

      [FLEX] Line=47, token=SPECIAL, value=","

      [FLEX] Line=47, token=KEYWORD, value="long"

      [FLEX] Line=47, token=IDENTIFIER, value="paramB"

      [FLEX] Line=47, token=SPECIAL, value=","

      [FLEX] Line=47, token=KEYWORD, value="short"

      [FLEX] Line=47, token=IDENTIFIER, value="paramC"

      [FLEX] Line=47, token=SPECIAL, value=")"


      [FLEX] Line=48, token=SPECIAL, value="{"




      [FLEX] Line=50, token=SPECIAL, value="}"


[BISON] Line=50, expression="Δήλωση συναρτήσεων χρήστη"
```

Οι token που αναγνωρίστηκαν από το Flex είναι σωστοί: func ως KEYWORD, myfunc2 ως IDENTIFIER, τα (,), {, }, , ως SPECIAL, και οι τύποι δεδομένων int, long, short ως KEYWORD.

Η έκφραση που αναγνωρίστηκε από το Bison (Γραμμή 50) είναι σωστή ως "Δήλωση συναρτήσεων χρήστη", καθώς η συνάρτηση myfunc2 δηλώθηκε σωστά με παραμέτρους.

```
      [FLEX] Line=51, token=KEYWORD, value="func"

      [FLEX] Line=51, token=IDENTIFIER, value="main"

      [FLEX] Line=51, token=SPECIAL, value="("

      [FLEX] Line=51, token=SPECIAL, value=")"



      [FLEX] Line=52, token=SPECIAL, value="{"



      [FLEX] Line=53, token=FUNCTION, value="print"

      [FLEX] Line=53, token=SPECIAL, value="("

      [FLEX] Line=53, token=STRING, value=""Hello World\n""

      [FLEX] Line=53, token=SPECIAL, value=")"

      [FLEX] Line=53, token=DELIMITER, value=";"

[BISON] Line=52, expression="Κλήση συνάρτησης"




      [FLEX] Line=54, token=SPECIAL, value="}"



[BISON] Line=54, expression="Δήλωση συναρτήσεων χρήστη"
```

Οι token που αναγνωρίστηκαν από το Flex είναι σωστοί: func ως KEYWORD, main ως IDENTIFIER, τα (,), {, }, ; ως SPECIAL, η συνάρτηση print ως FUNCTION, και η συμβολοσειρά "Hello World\n" ως STRING.

Η έκφραση που αναγνωρίστηκε από το Bison (Γραμμή 52) ως "Κλήση συνάρτησης" είναι σωστή, καθώς η κλήση της συνάρτησης print είναι έγκυρη μέσα στη συνάρτηση main.

Η έκφραση που αναγνωρίστηκε από το Bison (Γραμμή 54) ως "Δήλωση συναρτήσεων χρήστη" είναι επίσης σωστή, καθώς η συνάρτηση main δηλώθηκε και έκλεισε σωστά.

2.6 Δηλώσεις Απλών Εκφράσεων

Αρχείο Εισόδου (input.txt)

```
/* --- Test case [#2.6.1] : ARITHMETIC-EXPRESSIONS --- */


a1 * a2

a3 + a4

-5 + 10

15 + a5 - 9

+5 - a1 -16

+10 -5 + myvar - myvar2

5 % 1 / 2

+1 -4 / 7

-varA +varB * 1


/* --- Test case [#2.6.2] : VARIABLES-INITIALIZE --- */


x1=0;

x1, x2 = 0, 1;

x1, list1 = 0, ["A", "B", "C"];

x1, list1, string = 0, ["A", "B", "C"], "HELLO";

var++;

++var;

--var;

var--;

list1 += [1, 2, 3];
```

```
x -= 4e1;

y *= 6.7;

c /= b;


/* --- Test case [#2.6.3] : ARRAY-MERGE --- */


[1, 2, 3] + [4, 5, 6]

[1, 3] + [5, 7, 9, 11]

["What", "a"] + ["Wonderful", "World"]

[4.5, 5.6, 1.1] + [0e0, 4.5E-1, 4E0]


/* --- Test case [#2.6.4] : COMPARISONS --- */


x1 > x2

a < b

myvar >= 52

var1 <= var2

isEven == isOdd

a != C

isTrue1 || isTrue2

isFalse1 && isFalse2

!not
```

Αρχείο Εξόδου (output.txt)

```
/* --- Test case [#2.6.1] : ARITHMETIC-EXPRESSIONS --- */


      [FLEX] Line=59, token=IDENTIFIER, value="a1"

      [FLEX] Line=59, token=OPERATOR, value="*"

      [FLEX] Line=59, token=IDENTIFIER, value="a2"


[BISON] Line=59, expression="Αριθμητική έκφραση"
```

Αναγνωρίζεται ως αριθμητική έκφραση με δύο μεταβλητές (a1 και a2) και τον τελεστή πολλαπλασιασμού (*).

Ο ΛΑ αναγνωρίζει τα a1 και a2 ως IDENTIFIER και το * ως OPERATOR.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Αριθμητική έκφραση".

```
      [FLEX] Line=60, token=IDENTIFIER, value="a3"

      [FLEX] Line=60, token=OPERATOR, value="+"

      [FLEX] Line=60, token=IDENTIFIER, value="a4"


[BISON] Line=60, expression="Αριθμητική έκφραση"
```

Αναγνωρίζεται ως αριθμητική έκφραση με δύο μεταβλητές (a3 και a4) και τον τελεστή πρόσθεσης (+).

Ο ΛΑ αναγνωρίζει τα a3 και a4 ως IDENTIFIER και το + ως OPERATOR.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Αριθμητική έκφραση".

```
      [FLEX] Line=61, token=OPERATOR, value="-"

      [FLEX] Line=61, token=INTEGER, value="5"

      [FLEX] Line=61, token=OPERATOR, value="+"

      [FLEX] Line=61, token=INTEGER, value="10"
```

```
[BISON] Line=61, expression="Αριθμητική έκφραση"
```

Αναγνωρίζεται ως αριθμητική έκφραση με δύο ακέραιους (5 και 10) και τους τελεστές αφαίρεσης (-) και πρόσθεσης (+).

Ο ΛΑ αναγνωρίζει τους 5 και 10 ως INTEGER και τους - και + ως OPERATOR.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Αριθμητική έκφραση".

```
	[FLEX] Line=62, token=INTEGER, value="15"

	[FLEX] Line=62, token=OPERATOR, value="+"

	[FLEX] Line=62, token=IDENTIFIER, value="a5"

	[FLEX] Line=62, token=OPERATOR, value="-"

	[FLEX] Line=62, token=INTEGER, value="9"


[BISON] Line=62, expression="Αριθμητική έκφραση"
```

Αναγνωρίζεται ως αριθμητική έκφραση με έναν ακέραιο (15), μία μεταβλητή (a5) και έναν άλλο ακέραιο (9) και τους τελεστές πρόσθεσης (+) και αφαίρεσης (-).

Ο ΛΑ αναγνωρίζει τον 15 και τον 9 ως INTEGER, το a5 ως IDENTIFIER και τους + και - ως OPERATOR.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Αριθμητική έκφραση".

```
	[FLEX] Line=63, token=OPERATOR, value="+"

	[FLEX] Line=63, token=INTEGER, value="5"

	[FLEX] Line=63, token=OPERATOR, value="-"

	[FLEX] Line=63, token=IDENTIFIER, value="a1"

	[FLEX] Line=63, token=OPERATOR, value="-"

	[FLEX] Line=63, token=INTEGER, value="16"


[BISON] Line=63, expression="Αριθμητική έκφραση"
```

Αναγνωρίζεται ως αριθμητική έκφραση με έναν ακέραιο (5), μία μεταβλητή (a1) και έναν άλλο ακέραιο (16) και τους τελεστές πρόσθεσης (+) και αφαίρεσης (-).

Ο ΛΑ αναγνωρίζει τον 5 και τον 16 ως INTEGER, το a1ως IDENTIFIER και το - ως OPERATOR.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Αριθμητική έκφραση".

```
[FLEX] Line=64, token=OPERATOR, value="+"

[FLEX] Line=64, token=INTEGER, value="10"

[FLEX] Line=64, token=OPERATOR, value="-"

[FLEX] Line=64, token=INTEGER, value="5"

[FLEX] Line=64, token=OPERATOR, value="+"

[FLEX] Line=64, token=IDENTIFIER, value="myvar"

[FLEX] Line=64, token=OPERATOR, value="-"

[FLEX] Line=64, token=IDENTIFIER, value="myvar2"


[BISON] Line=64, expression="Αριθμητική έκφραση"
```

Αναγνωρίζεται ως αριθμητική έκφραση τους ακέραιους (10 και 5) και τις μεταβλητές myvar και mayvar2 και τους τελεστές πρόσθεσης (+) και αφαίρεσης (-).

Ο ΛΑ αναγνωρίζει το 5 και το 10 ως INTEGER, το myvar και myvar2 ως IDENTIFIER και το + και - ως OPERATOR.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Αριθμητική έκφραση".

```
[FLEX] Line=65, token=INTEGER, value="5"

[FLEX] Line=65, token=OPERATOR, value="%"

[FLEX] Line=65, token=INTEGER, value="1"

[FLEX] Line=65, token=OPERATOR, value="/"

[FLEX] Line=65, token=INTEGER, value="2"


[BISON] Line=65, expression="Αριθμητική έκφραση"
```

Αναγνωρίζεται ως αριθμητική έκφραση τους ακέραιους (5 ,1 και 2) και του τελεστές διαίρεσης % και /

Ο ΛΑ αναγνωριζει το 5 το 1 και το 2 σαν INTEGER το % και / ως OPERATOR.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Αριθμητική έκφραση".

```
      [FLEX] Line=66, token=OPERATOR, value="+"

      [FLEX] Line=66, token=INTEGER, value="1"

      [FLEX] Line=66, token=OPERATOR, value="-"

      [FLEX] Line=66, token=INTEGER, value="4"

      [FLEX] Line=66, token=OPERATOR, value="/"

      [FLEX] Line=66, token=INTEGER, value="7"


[BISON] Line=66, expression="Αριθμητική έκφραση"
```

Αναγνωριζεται ως αριθμητικη εκφραση τους ακεραιους (1 ,4,7) και του τελεστες προσθεσης (+) ,αφαίρεσης και διαίρεσης /

Ο ΛΑ αναγνωριζει το 1 το 4 και το 7 σαν INTEGER και το / ως OPERATOR.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Αριθμητική έκφραση".

```
      [FLEX] Line=67, token=OPERATOR, value="-"

      [FLEX] Line=67, token=IDENTIFIER, value="varA"

      [FLEX] Line=67, token=OPERATOR, value="+"

      [FLEX] Line=67, token=IDENTIFIER, value="varB"

      [FLEX] Line=67, token=OPERATOR, value="*"

      [FLEX] Line=67, token=INTEGER, value="1"


[BISON] Line=67, expression="Αριθμητική έκφραση"
```

Αναγνωριζεται αριθμητική εκφραση τους ακεραιους 1 και τις μεταβλητες varA και varB και τους μεταβλητες προσθεσης (+) αφαίρεσης (-) και του πολλαπλασιασμού (*).

Ο ΛΑ αναγνωριζει το varA και το varB ως IDENTIFIER και το 1 σαν INTEGER και το – το + και * ως OPERATOR.

Ο ΣΑ αναγνώριζει την έκφραση ως “Αριθμητική Έκφραση”.

```
/* --- Test case [#2.6.2] : VARIABLES-INITIALIZE --- */


      [FLEX] Line=71, token=IDENTIFIER, value="x1"

      [FLEX] Line=71, token=OPERATOR, value="="

      [FLEX] Line=71, token=INTEGER, value="0"

      [FLEX] Line=71, token=DELIMITER, value=";"

[BISON] Line=70, expression="Ανάθεση τιμής σε μεταβλητή"




[BISON] Line=71, expression="Ανάθεση τιμής σε μεταβλητή"
```

Αναγνωρίζεται ως έκφραση ανάθεσης τιμής σε μεταβλητή (x1) με τιμή 0.

Ο ΛΑ αναγνωρίζει το x1 ως IDENTIFIER, το = ως OPERATOR, το 0 ως INTEGER και το ; ως DELIMITER.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Ανάθεση τιμής σε μεταβλητή".

```
      [FLEX] Line=72, token=IDENTIFIER, value="x1"

      [FLEX] Line=72, token=SPECIAL, value=","

      [FLEX] Line=72, token=IDENTIFIER, value="x2"

      [FLEX] Line=72, token=OPERATOR, value="="

      [FLEX] Line=72, token=INTEGER, value="0"

      [FLEX] Line=72, token=SPECIAL, value=","

      [FLEX] Line=72, token=INTEGER, value="1"

      [FLEX] Line=72, token=DELIMITER, value=";"


[BISON] Line=72, expression="Ανάθεση τιμής σε μεταβλητή"
```

Αναγνωρίζεται ως έκφραση πολλαπλής ανάθεσης τιμής σε μεταβλητές (x1, x2) με τιμές 0 και 1.

Ο ΛΑ αναγνωρίζει τα x1 και x2 ως IDENTIFIER, το = ως OPERATOR, τα 0 και 1 ως INTEGER, και τα , και ; ως και DELIMITER αντίστοιχα.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Ανάθεση τιμής σε μεταβλητή".

```
	[FLEX] Line=73, token=IDENTIFIER, value="x1"

	[FLEX] Line=73, token=SPECIAL, value=","

	[FLEX] Line=73, token=IDENTIFIER, value="list1"

	[FLEX] Line=73, token=OPERATOR, value="="

	[FLEX] Line=73, token=INTEGER, value="0"

	[FLEX] Line=73, token=SPECIAL, value=","

	[FLEX] Line=73, token=SPECIAL, value="["

	[FLEX] Line=73, token=STRING, value=""A""

	[FLEX] Line=73, token=SPECIAL, value=","

	[FLEX] Line=73, token=STRING, value=""B""

	[FLEX] Line=73, token=SPECIAL, value=","

	[FLEX] Line=73, token=STRING, value=""C""

	[FLEX] Line=73, token=SPECIAL, value="]"

	[FLEX] Line=73, token=DELIMITER, value=";"


[BISON] Line=73, expression="Ανάθεση τιμής σε μεταβλητή"
```

Αναγνωρίζεται ως έκφραση πολλαπλής ανάθεσης τιμής σε μεταβλητές (x1, list1) με τιμές 0 και μια λίστα.

Ο ΛΑ αναγνωρίζει τα x1 και list1 ως IDENTIFIER, το = ως OPERATOR, το 0 ως INTEGER, τα [και] ως SPECIAL, και τα "A", "B", "C" ως STRING.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Ανάθεση τιμής σε μεταβλητή".

```
	[FLEX] Line=74, token=IDENTIFIER, value="x1"

	[FLEX] Line=74, token=SPECIAL, value=","

	[FLEX] Line=74, token=IDENTIFIER, value="list1"

	[FLEX] Line=74, token=SPECIAL, value=","

	[FLEX] Line=74, token=IDENTIFIER, value="string"

	[FLEX] Line=74, token=OPERATOR, value="="
```

```
	[FLEX] Line=74, token=INTEGER, value="0"

	[FLEX] Line=74, token=SPECIAL, value=","

	[FLEX] Line=74, token=SPECIAL, value="["

	[FLEX] Line=74, token=STRING, value=""A""

	[FLEX] Line=74, token=SPECIAL, value=","

	[FLEX] Line=74, token=STRING, value=""B""

	[FLEX] Line=74, token=SPECIAL, value=","

	[FLEX] Line=74, token=STRING, value=""C""

	[FLEX] Line=74, token=SPECIAL, value="]"

	[FLEX] Line=74, token=SPECIAL, value=","

	[FLEX] Line=74, token=STRING, value=""HELLO""

	[FLEX] Line=74, token=DELIMITER, value=";"


[BISON] Line=74, expression="Ανάθεση τιμής σε μεταβλητή"
```

Αναγνωρίζεται ως έκφραση πολλαπλής ανάθεσης τιμής σε μεταβλητές (x1, list1, string) με τιμές 0, μια λίστα και μια συμβολοσειρά.

Ο ΛΑ αναγνωρίζει τα x1, list1 και string ως IDENTIFIER, το = ως OPERATOR, το 0 ως INTEGER, τα [και] ως SPECIAL, τα "A", "B", "C" ως STRING, και το "HELLO" ως STRING.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Ανάθεση τιμής σε μεταβλητή".

```
	[FLEX] Line=75, token=IDENTIFIER, value="var"

	[FLEX] Line=75, token=OPERATOR, value="++"

	[FLEX] Line=75, token=DELIMITER, value=";"


[BISON] Line=75, expression="Ανάθεση τιμής σε μεταβλητή"
```

Αναγνωρίζεται ως έκφραση αύξησης τιμής της μεταβλητής (var) κατά 1.

Ο ΛΑ αναγνωρίζει το var ως IDENTIFIER, το ++ ως OPERATOR και το ; ως DELIMITER.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Ανάθεση τιμής σε μεταβλητή".

```
	[FLEX] Line=76, token=OPERATOR, value="++"

	[FLEX] Line=76, token=IDENTIFIER, value="var"

	[FLEX] Line=76, token=DELIMITER, value=";"


[BISON] Line=76, expression="Ανάθεση τιμής σε μεταβλητή"
```

Αναγνωρίζεται ως έκφραση αύξησης τιμής της μεταβλητής (var) κατά 1.

Ο ΛΑ αναγνωρίζει το ++ ως OPERATOR, το var ως IDENTIFIER και το ; ως DELIMITER.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Ανάθεση τιμής σε μεταβλητή".

```
	[FLEX] Line=77, token=OPERATOR, value="--"

	[FLEX] Line=77, token=IDENTIFIER, value="var"

	[FLEX] Line=77, token=DELIMITER, value=";"


[BISON] Line=77, expression="Ανάθεση τιμής σε μεταβλητή"
```

Αναγνωρίζεται ως έκφραση μείωσης τιμής της μεταβλητής (var) κατά 1.

Ο ΛΑ αναγνωρίζει το -- ως OPERATOR, το var ως IDENTIFIER και το ; ως DELIMITER.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Ανάθεση τιμής σε μεταβλητή".

```
	[FLEX] Line=78, token=IDENTIFIER, value="var"

	[FLEX] Line=78, token=OPERATOR, value="--"

	[FLEX] Line=78, token=DELIMITER, value=";"


[BISON] Line=78, expression="Ανάθεση τιμής σε μεταβλητή"
```

Αναγνωρίζεται ως έκφραση μείωσης τιμής της μεταβλητής (var) κατά 1.

Ο ΛΑ αναγνωρίζει το var ως IDENTIFIER, το -- ως OPERATOR και το ; ως DELIMITER.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Ανάθεση τιμής σε μεταβλητή".

```
	[FLEX] Line=79, token=IDENTIFIER, value="list1"
```

```
	[FLEX] Line=79, token=OPERATOR, value="+="

	[FLEX] Line=79, token=SPECIAL, value="["

	[FLEX] Line=79, token=INTEGER, value="1"

	[FLEX] Line=79, token=SPECIAL, value=","

	[FLEX] Line=79, token=INTEGER, value="2"

	[FLEX] Line=79, token=SPECIAL, value=","

	[FLEX] Line=79, token=INTEGER, value="3"

	[FLEX] Line=79, token=SPECIAL, value="]"

	[FLEX] Line=79, token=DELIMITER, value=";"


[BISON] Line=79, expression="Ανάθεση τιμής σε μεταβλητή"
```

Αναγνωρίζεται ως έκφραση προσθήκης στοιχείων (1, 2, 3) στη λίστα (list1).

Ο ΛΑ αναγνωρίζει το list1 ως IDENTIFIER, το += ως OPERATOR, τα [και] ως SPECIAL, και τους 1, 2 και 3 ως INTEGER.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Ανάθεση τιμής σε μεταβλητή".

```
	[FLEX] Line=80, token=IDENTIFIER, value="x"

	[FLEX] Line=80, token=OPERATOR, value="-="

	[FLEX] Line=80, token=FLOAT, value="4e1"

	[FLEX] Line=80, token=DELIMITER, value=";"


[BISON] Line=80, expression="Ανάθεση τιμής σε μεταβλητή"
```

Αναγνωρίζεται ως έκφραση μειωμένης ανάθεσης τιμής της μεταβλητής (x) κατά 40.

Ο ΛΑ αναγνωρίζει το x ως IDENTIFIER, το -= ως OPERATOR, το 4e1 ως FLOAT και το ; ως DELIMITER.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Ανάθεση τιμής σε μεταβλητή".

```
	[FLEX] Line=81, token=IDENTIFIER, value="y"
```

```
      [FLEX] Line=81, token=OPERATOR, value="*="

      [FLEX] Line=81, token=FLOAT, value="6.7"

      [FLEX] Line=81, token=DELIMITER, value=";"


[BISON] Line=81, expression="Ανάθεση τιμής σε μεταβλητή"
```

Αναγνωρίζεται ως έκφραση ανάθεσης τιμής της μεταβλητής (y) πολλαπλασιασμένης με 6.7

Ο ΛΑ αναγνωρίζει το y ως IDENTIFIER, το *= ως OPERATOR, το 6.7 ως FLOAT και το ; ως DELIMITER.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Ανάθεση τιμής σε μεταβλητή".

```
      [FLEX] Line=82, token=IDENTIFIER, value="c"

      [FLEX] Line=82, token=OPERATOR, value="/="

      [FLEX] Line=82, token=IDENTIFIER, value="b"

      [FLEX] Line=82, token=DELIMITER, value=";"


[BISON] Line=82, expression="Ανάθεση τιμής σε μεταβλητή"
```

Αναγνωρίζεται ως έκφραση ανάθεσης τιμής της μεταβλητής (c) διαιρεμένης με τη μεταβλητή (b).

Ο ΛΑ αναγνωρίζει τα c και b ως IDENTIFIER, το /= ως OPERATOR και το ; ως DELIMITER.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Ανάθεση τιμής σε μεταβλητή".

```
/* --- Test case [#2.6.3] : ARRAY-MERGE --- */


      [FLEX] Line=86, token=SPECIAL, value="["

      [FLEX] Line=86, token=INTEGER, value="1"

      [FLEX] Line=86, token=SPECIAL, value=","

      [FLEX] Line=86, token=INTEGER, value="2"

      [FLEX] Line=86, token=SPECIAL, value=","

      [FLEX] Line=86, token=INTEGER, value="3"
```

```
      [FLEX] Line=86, token=SPECIAL, value="]"

      [FLEX] Line=86, token=OPERATOR, value="+"

      [FLEX] Line=86, token=SPECIAL, value="["

      [FLEX] Line=86, token=INTEGER, value="4"

      [FLEX] Line=86, token=SPECIAL, value=","

      [FLEX] Line=86, token=INTEGER, value="5"

      [FLEX] Line=86, token=SPECIAL, value=","

      [FLEX] Line=86, token=INTEGER, value="6"

      [FLEX] Line=86, token=SPECIAL, value="]"


[BISON] Line=86, expression="Συνένωση πινάκων"


[BISON] Line=86, expression="Συνένωση πινάκων"
```

Αναγνωρίζεται ως έκφραση συγχώνευσης πινάκων με τα περιεχόμενα των πινάκων [1, 2, 3] και [4, 5, 6].

Ο ΛΑ αναγνωρίζει τα [και] ως SPECIAL και τα 1, 2, 3, 4, 5, 6 ως INTEGER.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Συγχώνευση πινάκων".

```
      [FLEX] Line=87, token=SPECIAL, value="["

      [FLEX] Line=87, token=INTEGER, value="1"

      [FLEX] Line=87, token=SPECIAL, value=","

      [FLEX] Line=87, token=INTEGER, value="3"

      [FLEX] Line=87, token=SPECIAL, value="]"

      [FLEX] Line=87, token=OPERATOR, value="+"

      [FLEX] Line=87, token=SPECIAL, value="["

      [FLEX] Line=87, token=INTEGER, value="5"

      [FLEX] Line=87, token=SPECIAL, value=","

      [FLEX] Line=87, token=INTEGER, value="7"
```

```
	[FLEX] Line=87, token=SPECIAL, value=","

	[FLEX] Line=87, token=INTEGER, value="9"

	[FLEX] Line=87, token=SPECIAL, value=","

	[FLEX] Line=87, token=INTEGER, value="11"

	[FLEX] Line=87, token=SPECIAL, value="]"


[BISON] Line=87, expression="Συνένωση πινάκων"
```

Αναγνωρίζεται ως έκφραση συγχώνευσης πινάκων με τα περιεχόμενα των πινάκων [1, 3] και [5, 7, 9, 11].

Το Flex αναγνωρίζει τα [και] ως SPECIAL και τα 1, 3, 5, 7, 9, 11 ως INTEGER.

Το Bison ταυτοποιεί τη γραμμή ως "Συγχώνευση πινάκων".

```
	[FLEX] Line=88, token=SPECIAL, value="["

	[FLEX] Line=88, token=STRING, value=""What""

	[FLEX] Line=88, token=SPECIAL, value=","

	[FLEX] Line=88, token=STRING, value=""a""

	[FLEX] Line=88, token=SPECIAL, value="]"

	[FLEX] Line=88, token=OPERATOR, value="+"

	[FLEX] Line=88, token=SPECIAL, value="["

	[FLEX] Line=88, token=STRING, value=""Wonderful""

	[FLEX] Line=88, token=SPECIAL, value=","

	[FLEX] Line=88, token=STRING, value=""World""

	[FLEX] Line=88, token=SPECIAL, value="]"


[BISON] Line=88, expression="Συνένωση πινάκων"
```

Αναγνωρίζεται ως έκφραση συγχώνευσης πινάκων με τα περιεχόμενα των πινάκων ["What", "a"] και ["Wonderful", "World"].

Ο ΛΑ αναγνωρίζει τα [και] ως SPECIAL και τα "What", "a", "Wonderful", "World" ως STRING.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Συγχώνευση πινάκων".

```
      [FLEX] Line=89, token=SPECIAL, value="["

      [FLEX] Line=89, token=FLOAT, value="4.5"

      [FLEX] Line=89, token=SPECIAL, value=","

      [FLEX] Line=89, token=FLOAT, value="5.6"

      [FLEX] Line=89, token=SPECIAL, value=","

      [FLEX] Line=89, token=FLOAT, value="1.1"

      [FLEX] Line=89, token=SPECIAL, value="]"

      [FLEX] Line=89, token=OPERATOR, value="+"

      [FLEX] Line=89, token=SPECIAL, value="["

      [FLEX] Line=89, token=FLOAT, value="0e0"

      [FLEX] Line=89, token=SPECIAL, value=","

      [FLEX] Line=89, token=FLOAT, value="4.5E-1"

      [FLEX] Line=89, token=SPECIAL, value=","

      [FLEX] Line=89, token=FLOAT, value="4E0"

      [FLEX] Line=89, token=SPECIAL, value="]"


[BISON] Line=89, expression="Συνένωση πινάκων"
```

Αναγνωρίζεται ως έκφραση συγχώνευσης πινάκων με τα περιεχόμενα των πινάκων [4.5, 5.6, 1.1] και [0e0, 4.5E-1, 4E0].

Ο ΛΑ αναγνωρίζει τα [και] ως SPECIAL, τα 4.5, 5.6, 1.1, 0e0, 4.5E-1 και 4E0 ως FLOAT.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Συγχώνευση πινάκων".

```
/* --- Test case [#2.6.4] : COMPARISONS --- */


      [FLEX] Line=93, token=IDENTIFIER, value="x1"

      [FLEX] Line=93, token=OPERATOR, value=">"

      [FLEX] Line=93, token=IDENTIFIER, value="x2"
```

```
[BISON] Line=93, expression="Σύγκριση"


[BISON] Line=93, expression="Σύγκριση"
```

Αναγνωρίζεται ως έκφραση σύγκρισης όπου η μεταβλητή x1 είναι μεγαλύτερη από τη μεταβλητή x2.

Ο ΛΑ αναγνωρίζει τα x1 και x2 ως IDENTIFIER και το > ως OPERATOR.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Σύγκριση"

```
	[FLEX] Line=94, token=IDENTIFIER, value="a"

	[FLEX] Line=94, token=OPERATOR, value="<"

	[FLEX] Line=94, token=IDENTIFIER, value="b"


[BISON] Line=94, expression="Σύγκριση"
```

Αναγνωρίζεται ως έκφραση σύγκρισης όπου η μεταβλητή a είναι μικρότερη από τη μεταβλητή b.

Ο ΛΑ αναγνωρίζει τα a και b ως IDENTIFIER και το < ως OPERATOR.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Σύγκριση".

```
	[FLEX] Line=95, token=IDENTIFIER, value="myvar"

	[FLEX] Line=95, token=OPERATOR, value=">="

	[FLEX] Line=95, token=INTEGER, value="52"


[BISON] Line=95, expression="Σύγκριση"
```

Αναγνωρίζεται ως έκφραση σύγκρισης όπου η μεταβλητή myvar είναι μεγαλύτερη ή ίση με τον αριθμό 52.

Ο ΛΑ αναγνωρίζει το myvar ως IDENTIFIER, το >= ως OPERATOR και το 52 ως INTEGER.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Σύγκριση".

```
	[FLEX] Line=96, token=IDENTIFIER, value="var1"
```

```
	[FLEX] Line=96, token=OPERATOR, value="<="

	[FLEX] Line=96, token=IDENTIFIER, value="var2"



[BISON] Line=96, expression="Σύγκριση"
```

Αναγνωρίζεται ως έκφραση σύγκρισης όπου η μεταβλητή var1 είναι μικρότερη ή ίση με τη μεταβλητή var2.

Ο ΛΑ αναγνωρίζει τα var1 και var2 ως IDENTIFIER και το <= ως OPERATOR.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Σύγκριση".

```
	[FLEX] Line=97, token=IDENTIFIER, value="isEven"

	[FLEX] Line=97, token=OPERATOR, value="=="

	[FLEX] Line=97, token=IDENTIFIER, value="isOdd"



[BISON] Line=97, expression="Σύγκριση"
```

Αναγνωρίζεται ως έκφραση σύγκρισης όπου η μεταβλητή isEven είναι ίση με τη μεταβλητή isOdd.

Ο ΛΑ αναγνωρίζει τα isEven και isOdd ως IDENTIFIER και το == ως OPERATOR.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Σύγκριση".

```
	[FLEX] Line=98, token=IDENTIFIER, value="a"

	[FLEX] Line=98, token=OPERATOR, value="!="

	[FLEX] Line=98, token=IDENTIFIER, value="C"



[BISON] Line=98, expression="Σύγκριση"
```

Αναγνωρίζεται ως έκφραση σύγκρισης όπου η μεταβλητή a είναι διαφορετική από την μεταβλητή C.

Ο ΛΑ αναγνωρίζει τα a και C ως IDENTIFIER και το != ως OPERATOR.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Σύγκριση".

```
	[FLEX] Line=99, token=IDENTIFIER, value="isTrue1"
```

```
	[FLEX] Line=99, token=OPERATOR, value="||"

	[FLEX] Line=99, token=IDENTIFIER, value="isTrue2"



[BISON] Line=99, expression="Σύγκριση"
```

Αναγνωρίζεται ως έκφραση σύγκρισης όπου ή η μεταβλητή isTrue1 ή η μεταβλητή isTrue2 είναι αληθείς.

Ο ΛΑ αναγνωρίζει τα isTrue1 και isTrue2 ως IDENTIFIER και το || ως OPERATOR.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Σύγκριση"

```
	[FLEX] Line=100, token=IDENTIFIER, value="isFalse1"

	[FLEX] Line=100, token=OPERATOR, value="&&"

	[FLEX] Line=100, token=IDENTIFIER, value="isFalse2"



[BISON] Line=100, expression="Σύγκριση"
```

Αναγνωρίζεται ως έκφραση σύγκρισης όπου τόσο η μεταβλητή isFalse1 όσο και η μεταβλητή isFalse2 είναι αληθείς.

Ο ΛΑ αναγνωρίζει τα isFalse1 και isFalse2 ως IDENTIFIER και το && ως OPERATOR.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Σύγκριση"

```
	[FLEX] Line=101, token=OPERATOR, value="!"

	[FLEX] Line=101, token=IDENTIFIER, value="not"



[BISON] Line=101, expression="Σύγκριση"
```

Αναγνωρίζεται ως έκφραση σύγκρισης όπου η μεταβλητή not είναι αληθής (λογική άρνηση).

Ο ΛΑ αναγνωρίζει το ! ως OPERATOR και το not ως IDENTIFIER.

Ο ΣΑ ταυτοποιεί τη γραμμή ως "Σύγκριση".

2.7 Σύνθετες Δηλώσεις

Αρχείο Εισόδου (input.txt)

```
/* --- Test case [#2.7.1] : IF-STATEMENTS --- */


if (var == 100) print("Value of expression is 100");

if (x < y < z) { print(x);  print(y);  print(z);  }


if (x == 0)

{

    int i;

    i = 1;

    if (i > 1)

        print("i is greater than 1");

}


/* --- Test case [#2.7.2] : WHILE-STATEMENTS --- */

while (i < 10)

{

print("i is: ", i);

}


/* --- Test case [#2.7.3] : FOR-STATEMENTS --- */

for (i = 0; i < 10; i++)

{

print("i = ", i);
```

```
}
```

Αρχείο Εξόδου (output.txt)

```
/* --- Test case [#2.7.1] : IF-STATEMENTS --- */


      [FLEX] Line=106, token=KEYWORD, value="if"

      [FLEX] Line=106, token=SPECIAL, value="("

      [FLEX] Line=106, token=IDENTIFIER, value="var"

      [FLEX] Line=106, token=OPERATOR, value="=="

      [FLEX] Line=106, token=INTEGER, value="100"

      [FLEX] Line=106, token=SPECIAL, value=")"

      [FLEX] Line=106, token=FUNCTION, value="print"

      [FLEX] Line=106, token=SPECIAL, value="("

      [FLEX] Line=106, token=STRING, value=""Value of expression is 100""

      [FLEX] Line=106, token=SPECIAL, value=")"

      [FLEX] Line=106, token=DELIMITER, value=";"


[BISON] Line=106, expression="Δήλωση if"
```

Η γραμμή περιέχει μια if-δήλωση με σύγκριση τιμής και κλήση συνάρτησης.

```
      [FLEX] Line=107, token=KEYWORD, value="if"

      [FLEX] Line=107, token=SPECIAL, value="("

      [FLEX] Line=107, token=IDENTIFIER, value="x"

      [FLEX] Line=107, token=OPERATOR, value="<"

      [FLEX] Line=107, token=IDENTIFIER, value="y"
```

```
      [FLEX] Line=107, token=OPERATOR, value="<"

      [FLEX] Line=107, token=IDENTIFIER, value="z"

      [FLEX] Line=107, token=SPECIAL, value=")"

      [FLEX] Line=107, token=SPECIAL, value="{"

      [FLEX] Line=107, token=FUNCTION, value="print"

      [FLEX] Line=107, token=SPECIAL, value="("

      [FLEX] Line=107, token=IDENTIFIER, value="x"

      [FLEX] Line=107, token=SPECIAL, value=")"

      [FLEX] Line=107, token=DELIMITER, value=";"

      [FLEX] Line=107, token=FUNCTION, value="print"

      [FLEX] Line=107, token=SPECIAL, value="("

      [FLEX] Line=107, token=IDENTIFIER, value="y"

      [FLEX] Line=107, token=SPECIAL, value=")"

      [FLEX] Line=107, token=DELIMITER, value=";"

[BISON] Line=106, expression="Κλήση συνάρτησης"


      [FLEX] Line=107, token=FUNCTION, value="print"

      [FLEX] Line=107, token=SPECIAL, value="("

      [FLEX] Line=107, token=IDENTIFIER, value="z"

      [FLEX] Line=107, token=SPECIAL, value=")"

      [FLEX] Line=107, token=DELIMITER, value=";"

[BISON] Line=106, expression="Κλήση συνάρτησης"


      [FLEX] Line=107, token=SPECIAL, value="}"


[BISON] Line=107, expression="Δήλωση if"
```

Η γραμμή περιέχει μια if-δήλωση με σύνθετη λογική έκφραση και πολλαπλές κλήσεις συναρτήσεων.

```
	[FLEX] Line=109, token=KEYWORD, value="if"
	[FLEX] Line=109, token=SPECIAL, value="("
	[FLEX] Line=109, token=IDENTIFIER, value="x"
	[FLEX] Line=109, token=OPERATOR, value="=="
	[FLEX] Line=109, token=INTEGER, value="0"
	[FLEX] Line=109, token=SPECIAL, value=")"

[BISON] Line=109, expression="Δήλωση if"

	[FLEX] Line=110, token=SPECIAL, value="{"

	[FLEX] Line=111, token=KEYWORD, value="int"
	[FLEX] Line=111, token=IDENTIFIER, value="i"
	[FLEX] Line=111, token=DELIMITER, value=";"
[BISON] Line=110, expression="Δήλωση μεταβλητής"

	[FLEX] Line=112, token=IDENTIFIER, value="i"
	[FLEX] Line=112, token=OPERATOR, value="="
	[FLEX] Line=112, token=INTEGER, value="1"
	[FLEX] Line=112, token=DELIMITER, value=";"
[BISON] Line=111, expression="Ανάθεση τιμής σε μεταβλητή"

	[FLEX] Line=113, token=KEYWORD, value="if"
	[FLEX] Line=113, token=SPECIAL, value="("
	[FLEX] Line=113, token=IDENTIFIER, value="i"
```

```
      [FLEX] Line=113, token=OPERATOR, value=">"

      [FLEX] Line=113, token=INTEGER, value="1"

      [FLEX] Line=113, token=SPECIAL, value=")"


[BISON] Line=113, expression="Δήλωση if"


      [FLEX] Line=114, token=FUNCTION, value="print"

      [FLEX] Line=114, token=SPECIAL, value="("

      [FLEX] Line=114, token=STRING, value=""i is greater than 1""

      [FLEX] Line=114, token=SPECIAL, value=")"

      [FLEX] Line=114, token=DELIMITER, value=";"

[BISON] Line=113, expression="Κλήση συνάρτησης"




      [FLEX] Line=115, token=SPECIAL, value="}"

[BISON] Line=114, expression="Σύνθετες δηλώσεις"




[BISON] Line=115, expression="Σύνθετες δηλώσεις"
```

Η γραμμή περιέχει μια if-δήλωση που περιέχει άλλες δηλώσεις και μια nested if-δήλωση.

```
/* --- Test case [#2.7.2] : WHILE-STATEMENTS --- */

      [FLEX] Line=118, token=KEYWORD, value="while"

      [FLEX] Line=118, token=SPECIAL, value="("

      [FLEX] Line=118, token=IDENTIFIER, value="i"

      [FLEX] Line=118, token=OPERATOR, value="<"

      [FLEX] Line=118, token=INTEGER, value="10"

      [FLEX] Line=118, token=SPECIAL, value=")"
```

```
      [FLEX] Line=119, token=SPECIAL, value="{"


      [FLEX] Line=120, token=FUNCTION, value="print"
      [FLEX] Line=120, token=SPECIAL, value="("
      [FLEX] Line=120, token=STRING, value=""i is: ""
      [FLEX] Line=120, token=SPECIAL, value=","
      [FLEX] Line=120, token=IDENTIFIER, value="i"
      [FLEX] Line=120, token=SPECIAL, value=")"
      [FLEX] Line=120, token=DELIMITER, value=";"
[BISON] Line=119, expression="Κλήση συνάρτησης"




      [FLEX] Line=121, token=SPECIAL, value="}"
[BISON] Line=120, expression="Σύνθετες δηλώσεις"



[BISON] Line=121, expression="Σύνθετες δηλώσεις"
```

Η γραμμή περιέχει μια while-δήλωση με ενσωματωμένο μπλοκ κώδικα που εκτελεί κλήση συνάρτησης.

```
/* --- Test case [#2.7.3] : FOR-STATEMENTS --- */
      [FLEX] Line=124, token=KEYWORD, value="for"
      [FLEX] Line=124, token=SPECIAL, value="("
      [FLEX] Line=124, token=IDENTIFIER, value="i"
      [FLEX] Line=124, token=OPERATOR, value="="
      [FLEX] Line=124, token=INTEGER, value="0"
      [FLEX] Line=124, token=DELIMITER, value=";"
      [FLEX] Line=124, token=IDENTIFIER, value="i"
```

```
      [FLEX] Line=124, token=OPERATOR, value="<"

      [FLEX] Line=124, token=INTEGER, value="10"

      [FLEX] Line=124, token=DELIMITER, value=";"

      [FLEX] Line=124, token=IDENTIFIER, value="i"

      [FLEX] Line=124, token=OPERATOR, value="++"

      [FLEX] Line=124, token=SPECIAL, value=")"


      [FLEX] Line=125, token=SPECIAL, value="{"


      [FLEX] Line=126, token=FUNCTION, value="print"

      [FLEX] Line=126, token=SPECIAL, value="("

      [FLEX] Line=126, token=STRING, value=""i = ""

      [FLEX] Line=126, token=SPECIAL, value=","

      [FLEX] Line=126, token=IDENTIFIER, value="i"

      [FLEX] Line=126, token=SPECIAL, value=")"

      [FLEX] Line=126, token=DELIMITER, value=";"

[BISON] Line=125, expression="Κλήση συνάρτησης"




      [FLEX] Line=127, token=SPECIAL, value="}"

[BISON] Line=126, expression="Σύνθετες δηλώσεις"




[BISON] Line=127, expression="Σύνθετες δηλώσεις"
```

Η γραμμή περιέχει μια for-δήλωση με ενσωματωμένο μπλοκ κώδικα που εκτελεί κλήση συνάρτησης.

- Ο ΛΑ και ο ΣΑ (FLEX και BISON) φαίνεται να λειτουργούν σωστά για την ανάλυση των if, while και for δηλώσεων.

- Οι εκφράσεις αναγνωρίζονται και κατηγοριοποιούνται επαρκώς και οι κλήσεις συναρτήσεων εντοπίζονται και περιγράφονται σωστά.
- Τα σύνθετα μπλοκ κώδικα (όπως nested-if δηλώσεις και μπλοκ εντολών μέσα σε βρόχους) αναλύονται και αποδίδονται με ακρίβεια.

Η διαδικασία της ανάλυσης μέσω FLEX και BISON επιβεβαιώνει τη σωστή λειτουργία του αναλυτή για τις δοκιμαστικές περιπτώσεις που δόθηκαν, με αναγνώριση και απόδοση των tokens και των εκφράσεων, όπως αναμένεται.

2.8 Συντακτικά Προειδοποιητικά Λάθη (Warnings)

Αρχείο Εισόδου (input.txt)

```
/* --- Test case [#2.8.1] : DOUBLE COMPARE --- */


x >> y

x << y


/* --- Test case [#2.8.2] : DOUBLE PARENTHESIS --- */


if ((x == 0))

{

}

while ((y == 0))

{

}


/* --- Test case [#2.8.3] : RETURN TYPE IN FUNCTIONS --- */


func int main()
```

```
{

}

func float avg(int parA, int parB)

{

    calc_avg(parA, parB);

}

func double pi()

{

    print("3.14");

}


/* --- Test case [#2.8.4] : INVALID INPUT IN MERGE ARRAYS --- */


[3, 4, 6] # + [1, 5, 6]

[1,2,8] + $ [7,9,1]


/* --- Test case [#2.8.5] : DOUBLE USE OF KEYWORD IN DELCARATION --- */


int int x;

float float y;

double double far;

short short imp;
```

Αρχείο Εξόδου (output.txt)

```
/* --- Test case [#2.8.1] : DOUBLE COMPARE --- */

```

```
      [FLEX] Line=132, token=IDENTIFIER, value="x"

      [FLEX] Line=132, token=OPERATOR, value=">"

      [FLEX] Line=132, token=OPERATOR, value=">"

      [FLEX] Line=132, token=IDENTIFIER, value="y"

## Warning ## -> Double > detected at Line=131


[BISON] Line=132, expression="Σύγκριση"
```

```
      [FLEX] Line=133, token=IDENTIFIER, value="x"

      [FLEX] Line=133, token=OPERATOR, value="<"

      [FLEX] Line=133, token=OPERATOR, value="<"

      [FLEX] Line=133, token=IDENTIFIER, value="y"

## Warning ## -> Double < detected at Line=132


[BISON] Line=133, expression="Σύγκριση"
```

Αποδέχεται το παράδειγμα, αλλά σημειώνει ότι υπάρχει προειδοποίηση για διπλά σύμβολα σύγκρισης (>> και <<). Οι γραμμές αναγνωρίζονται ως εκφράσεις σύγκρισης.

```
/* --- Test case [#2.8.2] : DOUBLE PARENTHESIS --- */


      [FLEX] Line=137, token=KEYWORD, value="if"

      [FLEX] Line=137, token=SPECIAL, value="("

      [FLEX] Line=137, token=SPECIAL, value="("

      [FLEX] Line=137, token=IDENTIFIER, value="x"

      [FLEX] Line=137, token=OPERATOR, value="=="

      [FLEX] Line=137, token=INTEGER, value="0"

      [FLEX] Line=137, token=SPECIAL, value=")"

      [FLEX] Line=137, token=SPECIAL, value=")"
```

```
## Warning ## -> Double parethensis detected at Line=137


[BISON] Line=137, expression="Δήλωση if"


	[FLEX] Line=138, token=SPECIAL, value="{"


	[FLEX] Line=139, token=SPECIAL, value="}"
[BISON] Line=138, expression="Σύνθετες δηλώσεις"




[BISON] Line=139, expression="Σύνθετες δηλώσεις"
```

```
	[FLEX] Line=140, token=KEYWORD, value="while"
	[FLEX] Line=140, token=SPECIAL, value="("
	[FLEX] Line=140, token=SPECIAL, value="("
	[FLEX] Line=140, token=IDENTIFIER, value="y"
	[FLEX] Line=140, token=OPERATOR, value="=="
	[FLEX] Line=140, token=INTEGER, value="0"
	[FLEX] Line=140, token=SPECIAL, value=")"
	[FLEX] Line=140, token=SPECIAL, value=")"
## Warning ## -> Double parethensis detected at Line=140


	[FLEX] Line=141, token=SPECIAL, value="{"


	[FLEX] Line=142, token=SPECIAL, value="}"
[BISON] Line=141, expression="Σύνθετες δηλώσεις"
```

```
[BISON] Line=142, expression="Σύνθετες δηλώσεις"
```

Αποδέχεται το παράδειγμα, αλλά σημειώνει προειδοποιήσεις για διπλές παρενθέσεις. Οι γραμμές αναγνωρίζονται ως δηλώσεις if και while.

```
/* --- Test case [#2.8.3] : RETURN TYPE IN FUNCTIONS --- */


      [FLEX] Line=146, token=KEYWORD, value="func"

      [FLEX] Line=146, token=KEYWORD, value="int"

      [FLEX] Line=146, token=IDENTIFIER, value="main"

## Warning ## -> Return type unnecessary at Line=146

      [FLEX] Line=146, token=SPECIAL, value="("

      [FLEX] Line=146, token=SPECIAL, value=")"


      [FLEX] Line=147, token=SPECIAL, value="{"


      [FLEX] Line=148, token=SPECIAL, value="}"

[BISON] Line=147, expression="Δήλωση συναρτήσεων χρήστη"



[BISON] Line=148, expression="Δήλωση συναρτήσεων χρήστη"
```

```
      [FLEX] Line=149, token=KEYWORD, value="func"

      [FLEX] Line=149, token=KEYWORD, value="float"

      [FLEX] Line=149, token=IDENTIFIER, value="avg"

## Warning ## -> Return type unnecessary at Line=149

      [FLEX] Line=149, token=SPECIAL, value="("

      [FLEX] Line=149, token=KEYWORD, value="int"
```

```
	[FLEX] Line=149, token=IDENTIFIER, value="parA"
	[FLEX] Line=149, token=SPECIAL, value=","
	[FLEX] Line=149, token=KEYWORD, value="int"
	[FLEX] Line=149, token=IDENTIFIER, value="parB"
	[FLEX] Line=149, token=SPECIAL, value=")"

	[FLEX] Line=150, token=SPECIAL, value="{"

	[FLEX] Line=151, token=IDENTIFIER, value="calc_avg"
	[FLEX] Line=151, token=SPECIAL, value="("
	[FLEX] Line=151, token=IDENTIFIER, value="parA"
	[FLEX] Line=151, token=SPECIAL, value=","
	[FLEX] Line=151, token=IDENTIFIER, value="parB"
	[FLEX] Line=151, token=SPECIAL, value=")"
	[FLEX] Line=151, token=DELIMITER, value=";"
[BISON] Line=150, expression="Κλήση συνάρτησης"

	[FLEX] Line=152, token=SPECIAL, value="}"

[BISON] Line=152, expression="Δήλωση συναρτήσεων χρήστη"
```

```
	[FLEX] Line=153, token=KEYWORD, value="func"
	[FLEX] Line=153, token=KEYWORD, value="double"
	[FLEX] Line=153, token=IDENTIFIER, value="pi"
## Warning ## -> Return type unnecessary at Line=153
	[FLEX] Line=153, token=SPECIAL, value="("
```

```
      [FLEX] Line=153, token=SPECIAL, value=")"


      [FLEX] Line=154, token=SPECIAL, value="{"


      [FLEX] Line=155, token=FUNCTION, value="print"
      [FLEX] Line=155, token=SPECIAL, value="("
      [FLEX] Line=155, token=STRING, value=""3.14""
      [FLEX] Line=155, token=SPECIAL, value=")"
      [FLEX] Line=155, token=DELIMITER, value=";"
[BISON] Line=154, expression="Κλήση συνάρτησης"




      [FLEX] Line=156, token=SPECIAL, value="}"


[BISON] Line=156, expression="Δήλωση συναρτήσεων χρήστη"
```

Αποδέχεται τις δηλώσεις των συναρτήσεων, αλλά σημειώνει προειδοποιήσεις ότι οι τύποι επιστροφής δεν είναι απαραίτητοι. Αναγνωρίζονται ως δηλώσεις συναρτήσεων χρήστη και κλήσεις συναρτήσεων.

```
/* --- Test case [#2.8.4] : INVALID INPUT IN MERGE ARRAYS --- */


      [FLEX] Line=160, token=SPECIAL, value="["
      [FLEX] Line=160, token=INTEGER, value="3"
      [FLEX] Line=160, token=SPECIAL, value=","
      [FLEX] Line=160, token=INTEGER, value="4"
      [FLEX] Line=160, token=SPECIAL, value=","
      [FLEX] Line=160, token=INTEGER, value="6"
      [FLEX] Line=160, token=SPECIAL, value="]"
!! Token error !! -> at Line=160
```

```
	1 character(s) ignored so far
	[FLEX] Line=160, token=OPERATOR, value="+"
	[FLEX] Line=160, token=SPECIAL, value="["
	[FLEX] Line=160, token=INTEGER, value="1"
	[FLEX] Line=160, token=SPECIAL, value=","
	[FLEX] Line=160, token=INTEGER, value="5"
	[FLEX] Line=160, token=SPECIAL, value=","
	[FLEX] Line=160, token=INTEGER, value="6"
	[FLEX] Line=160, token=SPECIAL, value="]"

## Warning ## -> Invalid character in array merge detected at Line=161
[BISON] Line=160, expression="Συνένωση πινάκων"

[BISON] Line=160, expression="Συνένωση πινάκων"
```

```
	[FLEX] Line=161, token=SPECIAL, value="["
	[FLEX] Line=161, token=INTEGER, value="1"
	[FLEX] Line=161, token=SPECIAL, value=","
	[FLEX] Line=161, token=INTEGER, value="2"
	[FLEX] Line=161, token=SPECIAL, value=","
	[FLEX] Line=161, token=INTEGER, value="8"
	[FLEX] Line=161, token=SPECIAL, value="]"
	[FLEX] Line=161, token=OPERATOR, value="+"
!! Token error !! -> at Line=161
	2 character(s) ignored so far
	[FLEX] Line=161, token=SPECIAL, value="["
	[FLEX] Line=161, token=INTEGER, value="7"
```

```
      [FLEX] Line=161, token=SPECIAL, value=","

      [FLEX] Line=161, token=INTEGER, value="9"

      [FLEX] Line=161, token=SPECIAL, value=","

      [FLEX] Line=161, token=INTEGER, value="1"

      [FLEX] Line=161, token=SPECIAL, value="]"


## Warning ## -> Invalid character in array merge detected at Line=162

[BISON] Line=161, expression="Συνένωση πινάκων"
```

Αποδέχεται τις δηλώσεις, αλλά σημειώνει σφάλματα χαρακτήρων που αγνοήθηκαν και προειδοποιήσεις για μη έγκυρους χαρακτήρες στη συνένωση πινάκων. Αναγνωρίζονται ως εκφράσεις συνένωσης πινάκων.

```
/* --- Test case [#2.8.5] : DOUBLE USE OF KEYWORD IN DELCARATION --- */


      [FLEX] Line=165, token=KEYWORD, value="int"

      [FLEX] Line=165, token=KEYWORD, value="int"

## Warning ## -> Double int detected at Line=165

      [FLEX] Line=165, token=IDENTIFIER, value="x"

      [FLEX] Line=165, token=DELIMITER, value=";"

[BISON] Line=164, expression="Δήλωση μεταβλητής"




[BISON] Line=165, expression="Δήλωση μεταβλητής"
```

```
      [FLEX] Line=166, token=KEYWORD, value="float"

      [FLEX] Line=166, token=KEYWORD, value="float"

## Warning ## -> Double float detected at Line=166

      [FLEX] Line=166, token=IDENTIFIER, value="y"

      [FLEX] Line=166, token=DELIMITER, value=";"
```

```
[BISON] Line=166, expression="Δήλωση μεταβλητής"
```

```
	[FLEX] Line=167, token=KEYWORD, value="double"

	[FLEX] Line=167, token=KEYWORD, value="double"

## Warning ## -> Double double detected at Line=167

	[FLEX] Line=167, token=IDENTIFIER, value="far"

	[FLEX] Line=167, token=DELIMITER, value=";"


[BISON] Line=167, expression="Δήλωση μεταβλητής"
```

```
	[FLEX] Line=168, token=KEYWORD, value="short"

	[FLEX] Line=168, token=KEYWORD, value="short"

## Warning ## -> Double short detected at Line=168

	[FLEX] Line=168, token=IDENTIFIER, value="imp"

	[FLEX] Line=168, token=DELIMITER, value=";"


[BISON] Line=168, expression="Δήλωση μεταβλητής"
```

Αποδέχεται τις δηλώσεις, αλλά σημειώνει προειδοποιήσεις για διπλά keywords. Αναγνωρίζονται ως δηλώσεις μεταβλητών.

2.9 Συντακτικά Λάθη

Αρχείο Εισόδου (input.txt)

```
intx a;

pin1 = [1, 3, "Hello", 4, 5];

scan(&x);

len(1variable);

cmp(x, 5);

printf("Avg is %f", avg);

func myFunc(long x, short y) {

	int c;

	c = x / y;

	return c;

}

myFunc(x, y, 10);

5++;

x -= "Hello";

y *= [h, e, l, l, o];

50 = x;

list1, list2 = [x, b, c], 5;

x1 === x2;

array += [1,3,4];

[1,3,5] ++ [1,2];

if var == 100:

    print("var", "is", var);

while(1)
```

```
{
    print("Hello Wolrld\n");
}
for(;;)
{
    print("Hello World\n");
}
```

Αρχείο Εξόδου (output.txt)

```
      [FLEX] Line=172, token=IDENTIFIER, value="intx"
      [FLEX] Line=172, token=IDENTIFIER, value="a"
[BISON] Line=171, expression="Αριθμητική έκφραση"

      [FLEX] Line=172, token=DELIMITER, value=";"
[BISON] Line=171, expression="Αριθμητική έκφραση"

!! syntax error !! -> at Line=172
```

Προσπαθεί να δηλώσει έναν ακέραιο με το όνομα a, ωστόσο υπάρχει ένα συντακτικό λάθος εξαιτίας του x αντί του int.

```
      [FLEX] Line=173, token=IDENTIFIER, value="pin1"
      [FLEX] Line=173, token=OPERATOR, value="="
      [FLEX] Line=173, token=SPECIAL, value="["
      [FLEX] Line=173, token=INTEGER, value="1"
      [FLEX] Line=173, token=SPECIAL, value=","
      [FLEX] Line=173, token=INTEGER, value="3"
      [FLEX] Line=173, token=SPECIAL, value=","
```

```
      [FLEX] Line=173, token=STRING, value=""Hello""

!! syntax error !! -> at Line=173

      [FLEX] Line=173, token=SPECIAL, value=","

      [FLEX] Line=173, token=INTEGER, value="4"

      [FLEX] Line=173, token=SPECIAL, value=","

      [FLEX] Line=173, token=INTEGER, value="5"

      [FLEX] Line=173, token=SPECIAL, value="]"

      [FLEX] Line=173, token=DELIMITER, value=";"
```

Προσπαθεί να εκχωρήσει μια λίστα στη μεταβλητή pin1, αλλά το "Hello" δεν είναι ακέραιος αριθμός. Τα περιεχόμενα ενός πίνακα στη Uni-C είναι αυστηρά ίδιου τύπου δεδομένων.

```
      [FLEX] Line=174, token=FUNCTION, value="scan"

      [FLEX] Line=174, token=SPECIAL, value="("

      [FLEX] Line=174, token=OPERATOR, value="&"

!! syntax error !! -> at Line=174

      [FLEX] Line=174, token=IDENTIFIER, value="x"

      [FLEX] Line=174, token=SPECIAL, value=")"

      [FLEX] Line=174, token=DELIMITER, value=";"
```

Προσπαθεί να σκανάρει έναν ακέραιο από το χρήστη και να τον αποθηκεύσει στη μεταβλητή x. Στη Uni-C οι μεταβλητές δεν διαβάζονται με το τελεστή διεύθυνσης &.

```
      [FLEX] Line=175, token=FUNCTION, value="len"

      [FLEX] Line=175, token=SPECIAL, value="("

      [FLEX] Line=175, token=INTEGER, value="1"

!! syntax error !! -> at Line=175

      [FLEX] Line=175, token=IDENTIFIER, value="variable"

      [FLEX] Line=175, token=SPECIAL, value=")"

      [FLEX] Line=175, token=DELIMITER, value=";"
```

Προσπαθεί να καλέσει μια συνάρτηση με το όνομα len με την παράμετρο 1variable, αλλά η σύνταξη φαίνεται σωστή. Ο λόγος που ο parser βγάζει συντακτικό λάθος είναι στην αναγνώριση του

αναγνωρίσματος “1variable”. Ο ΛΑ δεν αναγνωρίζει αναγνωρίσματα που ξεκινάνε με ψηφίο και γι’ αυτό το λόγο επιστρέφει 2 tokens, έναν ακέραιο (1) και ένα αναγνώρισμα (variable). Έτσι, ο ΣΑ αναζητεί κανόνα στις παραμέτρους της len() που να έχει ακέραιο και αναγνώρισμα, όπου δεν βρίσκει και επομένως θεωρεί τη σύνταξη λανθασμένη.

```
      [FLEX] Line=176, token=FUNCTION, value="cmp"

      [FLEX] Line=176, token=SPECIAL, value="("

      [FLEX] Line=176, token=IDENTIFIER, value="x"

      [FLEX] Line=176, token=SPECIAL, value=","

      [FLEX] Line=176, token=INTEGER, value="5"

!! syntax error !! -> at Line=176

      [FLEX] Line=176, token=SPECIAL, value=")"

      [FLEX] Line=176, token=DELIMITER, value=";"
```

Προσπαθεί να συγκρίνει το x με το 5, ωστόσο, η cmp συγκρίνει δύο συμβολοσειρές αν είναι ίσες και όχι αριθμούς

```
      [FLEX] Line=177, token=IDENTIFIER, value="printf"

      [FLEX] Line=177, token=SPECIAL, value="("

      [FLEX] Line=177, token=STRING, value=""Avg is %f""

      [FLEX] Line=177, token=SPECIAL, value=","

      [FLEX] Line=177, token=IDENTIFIER, value="avg"

      [FLEX] Line=177, token=SPECIAL, value=")"

      [FLEX] Line=177, token=DELIMITER, value=";"


[BISON] Line=177, expression="Κλήση συνάρτησης"
```

Υπάρχει η ενσωματωμένη απλή συνάρτηση “print”, ωστόσο ο χρήστης μπορούσε να ορίσει μία δική του συνάρτηση με όνομα “printf”. Γι’ αυτό το λόγο ο ΣΑ δεν βγάζει συντακτικό λάθος στο όνομα της “print”

```
      [FLEX] Line=178, token=KEYWORD, value="func"

      [FLEX] Line=178, token=IDENTIFIER, value="myFunc"

      [FLEX] Line=178, token=SPECIAL, value="("
```

```
	[FLEX] Line=178, token=KEYWORD, value="long"
	[FLEX] Line=178, token=IDENTIFIER, value="x"
	[FLEX] Line=178, token=SPECIAL, value=","
	[FLEX] Line=178, token=KEYWORD, value="short"
	[FLEX] Line=178, token=IDENTIFIER, value="y"
	[FLEX] Line=178, token=SPECIAL, value=")"
	[FLEX] Line=178, token=SPECIAL, value="{"
!! syntax error !! -> at Line=178

	[FLEX] Line=179, token=KEYWORD, value="int"
	[FLEX] Line=179, token=IDENTIFIER, value="c"
	[FLEX] Line=179, token=DELIMITER, value=";"

[BISON] Line=179, expression="Δήλωση μεταβλητής"

	[FLEX] Line=180, token=IDENTIFIER, value="c"
	[FLEX] Line=180, token=OPERATOR, value="="
	[FLEX] Line=180, token=IDENTIFIER, value="x"
	[FLEX] Line=180, token=OPERATOR, value="/"
	[FLEX] Line=180, token=IDENTIFIER, value="y"
	[FLEX] Line=180, token=DELIMITER, value=";"

[BISON] Line=180, expression="Ανάθεση τιμής σε μεταβλητή"

	[FLEX] Line=181, token=KEYWORD, value="return"
!! syntax error !! -> at Line=181
	[FLEX] Line=181, token=IDENTIFIER, value="c"
```

```
	[FLEX] Line=181, token=DELIMITER, value=";"



	[FLEX] Line=182, token=SPECIAL, value="}"

!! syntax error !! -> at Line=182
```

Ορίζει μια συνάρτηση με το όνομα myFunc που δέχεται έναν ακέραιο και ένα short, αλλά η σύνταξη είναι λανθασμένη, καθώς, η Uni-C δεν υποστηρίζει τα άγκιστρα για το σώμα της συνάρτησης να ανοίγουν στην ίδια γραμμή που ορίζονται οι παράμετροι της συνάρτησης. Επίσης, η Uni-C δεν υποστηρίζει επιστροφές τιμών με την χρήση του keyword “return”. Ο ΣΑ συνεχίζει την αναγνώριση συντακτικών εκφράσεων και μετά από τα συντακτικά λάθη, και προσπαθεί να βρει κανόνα που να περιέχει αναγνώρισμα (c), οριοθέτη (;) και άγκιστρο (}). Προφανώς, δεν βρίσκει και γι’ αυτό τον λόγο έχουμε άλλο ένα συντακτικό σφάλμα.

```
	[FLEX] Line=183, token=IDENTIFIER, value="myFunc"

	[FLEX] Line=183, token=SPECIAL, value="("

	[FLEX] Line=183, token=IDENTIFIER, value="x"

	[FLEX] Line=183, token=SPECIAL, value=","

	[FLEX] Line=183, token=IDENTIFIER, value="y"

	[FLEX] Line=183, token=SPECIAL, value=","

	[FLEX] Line=183, token=INTEGER, value="10"

	[FLEX] Line=183, token=SPECIAL, value=")"

	[FLEX] Line=183, token=DELIMITER, value=";"



[BISON] Line=183, expression="Κλήση συνάρτησης"
```

Καλεί τη συνάρτηση myFunc με τρεις παραμέτρους. Η σύνταξη είναι σωστή, αλλά σημασιολογικά θα περίμενε κανείς ότι δεν είναι, καθώς έχει δηλωθεί μία συνάρτηση myFunc με 2 παραμέτρους, ενώ τη καλούμε με 3.

```
	[FLEX] Line=184, token=INTEGER, value="5"

	[FLEX] Line=184, token=OPERATOR, value="++"

	[FLEX] Line=184, token=DELIMITER, value=";"

!! syntax error !! -> at Line=184
```

Προσπαθεί να αυξήσει το 5, αλλά αυτό είναι μια σταθερά και δεν μπορεί να τροποποιηθεί.

```
      [FLEX] Line=185, token=IDENTIFIER, value="x"

      [FLEX] Line=185, token=OPERATOR, value="-="

      [FLEX] Line=185, token=STRING, value=""Hello""

!! syntax error !! -> at Line=185

      [FLEX] Line=185, token=DELIMITER, value=";"
```

Προσπαθεί να αφαιρέσει τη συμβολοσειρά "Hello" από τον ακέραιο x. Η Uni-C δεν υποστηρίζει αφαιρέσεις με συμβολοσειρές παρά μόνο προσθέσεις π.χ. “Hello” + “World”.

```
      [FLEX] Line=186, token=IDENTIFIER, value="y"

      [FLEX] Line=186, token=OPERATOR, value="*="

      [FLEX] Line=186, token=SPECIAL, value="["

!! syntax error !! -> at Line=186

      [FLEX] Line=186, token=IDENTIFIER, value="h"

      [FLEX] Line=186, token=SPECIAL, value=","

      [FLEX] Line=186, token=IDENTIFIER, value="e"

      [FLEX] Line=186, token=SPECIAL, value=","

      [FLEX] Line=186, token=IDENTIFIER, value="l"

      [FLEX] Line=186, token=SPECIAL, value=","

      [FLEX] Line=186, token=IDENTIFIER, value="l"

      [FLEX] Line=186, token=SPECIAL, value=","

      [FLEX] Line=186, token=IDENTIFIER, value="o"

      [FLEX] Line=186, token=SPECIAL, value="]"

      [FLEX] Line=186, token=DELIMITER, value=";"
```

Προσπαθεί να πολλαπλασιάσει τον ακέραιο y με μια λίστα, που δεν είναι έγκυρη λειτουργία.

```
      [FLEX] Line=187, token=INTEGER, value="50"

      [FLEX] Line=187, token=OPERATOR, value="="

!! syntax error !! -> at Line=187

      [FLEX] Line=187, token=IDENTIFIER, value="x"
```

```
      [FLEX] Line=187, token=DELIMITER, value=";"
```

Προσπαθεί να ορίσει την τιμή 50 στον ακέραιο x, αλλά η αντιστροφή των αντικειμένων στην ανάθεση είναι λανθασμένη.

```
      [FLEX] Line=188, token=IDENTIFIER, value="list1"

      [FLEX] Line=188, token=SPECIAL, value=","

      [FLEX] Line=188, token=IDENTIFIER, value="list2"

      [FLEX] Line=188, token=OPERATOR, value="="

      [FLEX] Line=188, token=SPECIAL, value="["

      [FLEX] Line=188, token=IDENTIFIER, value="x"

      [FLEX] Line=188, token=SPECIAL, value=","

      [FLEX] Line=188, token=IDENTIFIER, value="b"

      [FLEX] Line=188, token=SPECIAL, value=","

      [FLEX] Line=188, token=IDENTIFIER, value="c"

      [FLEX] Line=188, token=SPECIAL, value="]"

      [FLEX] Line=188, token=SPECIAL, value=","

      [FLEX] Line=188, token=INTEGER, value="5"

      [FLEX] Line=188, token=DELIMITER, value=";"


[BISON] Line=188, expression="Ανάθεση τιμής σε μεταβλητή"
```

Προσπαθεί να ορίσει δύο λίστες και να τις αντιστοιχήσει σε δύο μεταβλητές. Η δεύτερη μεταβλητή όμως είναι ακέραιος αριθμός, οπότε σημασιολογικά η έκφραση δεν είναι σωστή. Συντακτικά είναι και γι' αυτό ο ΣΑ δεν παραπονιέται.

```
      [FLEX] Line=189, token=IDENTIFIER, value="x1"

      [FLEX] Line=189, token=OPERATOR, value="=="

      [FLEX] Line=189, token=OPERATOR, value="="

!! syntax error !! -> at Line=189

      [FLEX] Line=189, token=IDENTIFIER, value="x2"

      [FLEX] Line=189, token=DELIMITER, value=";"
```

Προσπαθεί να εκτελέσει ένα τριπλό έλεγχο ισότητας, αλλά αυτό δεν είναι έγκυρο στην περίπτωση της Uni-C.

```
	[FLEX] Line=190, token=IDENTIFIER, value="array"

	[FLEX] Line=190, token=OPERATOR, value="+="

	[FLEX] Line=190, token=SPECIAL, value="["

	[FLEX] Line=190, token=INTEGER, value="1"

	[FLEX] Line=190, token=SPECIAL, value=","

	[FLEX] Line=190, token=INTEGER, value="3"

	[FLEX] Line=190, token=SPECIAL, value=","

	[FLEX] Line=190, token=INTEGER, value="4"

	[FLEX] Line=190, token=SPECIAL, value="]"

	[FLEX] Line=190, token=DELIMITER, value=";"


[BISON] Line=190, expression="Ανάθεση τιμής σε μεταβλητή"
```

Προσπαθεί να προσθέσει μια λίστα σε έναν πίνακα. Η λειτουργία αυτή είναι συντακτικά σωστή καθώς πρόκειται για μία παραλλαγή της συνένωσης πινάκων. Σημασιολογικά θα μπορούσε να ελεγχθεί το περιεχόμενο της “array”.

```
	[FLEX] Line=191, token=SPECIAL, value="["

	[FLEX] Line=191, token=INTEGER, value="1"

	[FLEX] Line=191, token=SPECIAL, value=","

	[FLEX] Line=191, token=INTEGER, value="3"

	[FLEX] Line=191, token=SPECIAL, value=","

	[FLEX] Line=191, token=INTEGER, value="5"

	[FLEX] Line=191, token=SPECIAL, value="]"

	[FLEX] Line=191, token=OPERATOR, value="++"

	[FLEX] Line=191, token=SPECIAL, value="["

!! syntax error !! -> at Line=191

	[FLEX] Line=191, token=INTEGER, value="1"
```

```
	[FLEX] Line=191, token=SPECIAL, value=","

	[FLEX] Line=191, token=INTEGER, value="2"

	[FLEX] Line=191, token=SPECIAL, value="]"

	[FLEX] Line=191, token=DELIMITER, value=";"
```

Προσπαθεί να εκτελέσει συνένωση δύο λιστών με τον τελεστή ++, αλλά αυτή η σύνταξη δεν είναι έγκυρη στην Uni-C. Η συνένωση πινάκων γίνεται με την χρήση του τελεστή + και όχι του τελεστή προσαύξησης/μετααύξησης ++.

```
	[FLEX] Line=192, token=KEYWORD, value="if"

	[FLEX] Line=192, token=IDENTIFIER, value="var"

	[FLEX] Line=192, token=OPERATOR, value="=="

	[FLEX] Line=192, token=INTEGER, value="100"

!! Token error !! -> at Line=192

!! syntax error !! -> at Line=192

		4 character(s) ignored so far

	[FLEX] Line=193, token=FUNCTION, value="print"

	[FLEX] Line=193, token=SPECIAL, value="("

	[FLEX] Line=193, token=STRING, value=""var""

	[FLEX] Line=193, token=SPECIAL, value=","

	[FLEX] Line=193, token=STRING, value=""is""

	[FLEX] Line=193, token=SPECIAL, value=","

	[FLEX] Line=193, token=IDENTIFIER, value="var"

	[FLEX] Line=193, token=SPECIAL, value=")"

	[FLEX] Line=193, token=DELIMITER, value=";"


[BISON] Line=193, expression="Κλήση συνάρτησης"
```

Η σύνταξη της if είναι λάθος, καθώς, απουσιάζουν οι παρενθέσεις στη συνθήκη “var == 100”. Επίσης, ο ΛΑ αναγνωρίζει έναν άκυρο χαρακτήρα την άνω-κάτω τελεία “:” και γι’ αυτό το λόγο επιστρέφει και token error. Η σύνταξη της “print” συνάρτησης είναι σωστή.

```
	[FLEX] Line=194, token=KEYWORD, value="while"

	[FLEX] Line=194, token=SPECIAL, value="("

	[FLEX] Line=194, token=INTEGER, value="1"

	[FLEX] Line=194, token=SPECIAL, value=")"


	[FLEX] Line=195, token=SPECIAL, value="{"


	[FLEX] Line=196, token=FUNCTION, value="print"

	[FLEX] Line=196, token=SPECIAL, value="("

	[FLEX] Line=196, token=STRING, value=""Hello Wolrld\n""

	[FLEX] Line=196, token=SPECIAL, value=")"

	[FLEX] Line=196, token=DELIMITER, value=";"

[BISON] Line=195, expression="Κλήση συνάρτησης"




	[FLEX] Line=197, token=SPECIAL, value="}"

[BISON] Line=196, expression="Σύνθετες δηλώσεις"




[BISON] Line=197, expression="Σύνθετες δηλώσεις"
```

Δημιουργεί έναν ατέρμονα βρόχο εκτέλεσης while. Ο ΣΑ θα περίμενε κανείς ότι θα έβγαζε συντακτικό σφάλμα, καθώς, η while δεν έχει συνθήκη. Ωστόσο, η σύνταξη είναι σωστή, καθώς σταθερές στις συνθήκες των σύνθετων δηλώσεων (if, while), επιτρέπονται στη Uni-C. Οποιαδήποτε συνθήκη δίνει αποτέλεσμα 1 ή 0 είναι αποδεκτή ακόμα και αν είναι σταθερά.

```
	[FLEX] Line=198, token=KEYWORD, value="for"

	[FLEX] Line=198, token=SPECIAL, value="("

	[FLEX] Line=198, token=DELIMITER, value=";"

!! syntax error !! -> at Line=198
```

```
      [FLEX] Line=198, token=DELIMITER, value=";"

      [FLEX] Line=198, token=SPECIAL, value=")"



      [FLEX] Line=199, token=SPECIAL, value="{"



      [FLEX] Line=200, token=FUNCTION, value="print"

      [FLEX] Line=200, token=SPECIAL, value="("

      [FLEX] Line=200, token=STRING, value=""Hello World\n""

      [FLEX] Line=200, token=SPECIAL, value=")"

      [FLEX] Line=200, token=DELIMITER, value=";"

[BISON] Line=199, expression="Κλήση συνάρτησης"




      [FLEX] Line=201, token=SPECIAL, value="}"



[BISON] Line=201, expression="Σύνθετες δηλώσεις"
```

Δημιουργεί έναν ατέρμονα βρόχο εκτέλεσης for, αλλά ο ΣΑ βγάζει συντακτικό σφάλμα. Η Uni-C δεν υποστηρίζει ατέρμονους βρόχους σε δηλώσεις for, παρά μόνο σε δηλώσεις while.

2.10 Ενιαίο

Αρχείο Εισόδου (input.txt)

```
func myAdd(int num1, int num2)

{

       int sum;

       sum = a + b;

    print("Sum is: ", sum);

}



func main()

{

       int a;

       int b;

    scan(a);

    scan(b);

    myAdd(a,b);

}



func errorHandling()

{

       long a, b, c, d, e;

       int i;

       int x;



    scan(a, b, c, d, e);
```

```
      pin1 = [a, b, c];

      pin2 = [d, e];

   pinFinal = [a, b, c] + $[d, e];


      for(i = 0; i < pinFinal.N; i++)

      {

      x = pinFinal[i];

      x /= 2;

      if ((x == 0))

            print("%d is Even\n", pinFinal[i]);

      }

}
```

Αρχείο Εξόδου (output.txt)

Δοκιμάσαμε τις δυνατότητες του συντακτικού και λεκτικού αναλυτή με παράδειγμα κάποιες συναρτήσεις που θα μπορούσε να δημιουργήσει κάποιος χρήστης.

```
      [FLEX] Line=204, token=KEYWORD, value="func"

      [FLEX] Line=204, token=IDENTIFIER, value="myAdd"

      [FLEX] Line=204, token=SPECIAL, value="("

      [FLEX] Line=204, token=KEYWORD, value="int"

      [FLEX] Line=204, token=IDENTIFIER, value="num1"

      [FLEX] Line=204, token=SPECIAL, value=","

      [FLEX] Line=204, token=KEYWORD, value="int"

      [FLEX] Line=204, token=IDENTIFIER, value="num2"

      [FLEX] Line=204, token=SPECIAL, value=")"


      [FLEX] Line=205, token=SPECIAL, value="{"
```

```
      [FLEX] Line=206, token=KEYWORD, value="int"
      [FLEX] Line=206, token=IDENTIFIER, value="sum"
      [FLEX] Line=206, token=DELIMITER, value=";"
[BISON] Line=205, expression="Δήλωση μεταβλητής"


      [FLEX] Line=207, token=IDENTIFIER, value="sum"
      [FLEX] Line=207, token=OPERATOR, value="="
      [FLEX] Line=207, token=IDENTIFIER, value="a"
      [FLEX] Line=207, token=OPERATOR, value="+"
      [FLEX] Line=207, token=IDENTIFIER, value="b"
      [FLEX] Line=207, token=DELIMITER, value=";"
[BISON] Line=206, expression="Ανάθεση τιμής σε μεταβλητή"


      [FLEX] Line=208, token=FUNCTION, value="print"
      [FLEX] Line=208, token=SPECIAL, value="("
      [FLEX] Line=208, token=STRING, value=""Sum is: ""
      [FLEX] Line=208, token=SPECIAL, value=","
      [FLEX] Line=208, token=IDENTIFIER, value="sum"
      [FLEX] Line=208, token=SPECIAL, value=")"
      [FLEX] Line=208, token=DELIMITER, value=";"
[BISON] Line=207, expression="Κλήση συνάρτησης"


      [FLEX] Line=209, token=SPECIAL, value="}"
```

```
[BISON] Line=209, expression="Δήλωση συναρτήσεων χρήστη"
```

Στην πρώτη συνάρτηση με όνομα Myadd, αναγνωρίζονται η δήλωση μεταβλητής στη γραμμή 206, η ανάθεση τιμής σε μεταβλητή στη γραμμή 207, καθώς και τέλος η κλήση της συνάρτησης print στη γραμμή 208. Τέλος, στη γραμμή 209 ορίζεται η δηλωμένη από το χρήστη συνάρτηση με τον χαρακτήρα "}" ο οποίος "κλείνει" τη συνάρτηση Myadd.

```
      [FLEX] Line=211, token=KEYWORD, value="func"
      [FLEX] Line=211, token=IDENTIFIER, value="main"
      [FLEX] Line=211, token=SPECIAL, value="("
      [FLEX] Line=211, token=SPECIAL, value=")"

      [FLEX] Line=212, token=SPECIAL, value="{"

      [FLEX] Line=213, token=KEYWORD, value="int"
      [FLEX] Line=213, token=IDENTIFIER, value="a"
      [FLEX] Line=213, token=DELIMITER, value=";"
[BISON] Line=212, expression="Δήλωση μεταβλητής"


      [FLEX] Line=214, token=KEYWORD, value="int"
      [FLEX] Line=214, token=IDENTIFIER, value="b"
      [FLEX] Line=214, token=DELIMITER, value=";"
[BISON] Line=213, expression="Δήλωση μεταβλητής"


      [FLEX] Line=215, token=FUNCTION, value="scan"
      [FLEX] Line=215, token=SPECIAL, value="("
      [FLEX] Line=215, token=IDENTIFIER, value="a"
```

```
      [FLEX] Line=215, token=SPECIAL, value=")"
      [FLEX] Line=215, token=DELIMITER, value=";"
[BISON] Line=214, expression="Κλήση συνάρτησης"



      [FLEX] Line=216, token=FUNCTION, value="scan"
      [FLEX] Line=216, token=SPECIAL, value="("
      [FLEX] Line=216, token=IDENTIFIER, value="b"
      [FLEX] Line=216, token=SPECIAL, value=")"
      [FLEX] Line=216, token=DELIMITER, value=";"
[BISON] Line=215, expression="Κλήση συνάρτησης"



      [FLEX] Line=217, token=IDENTIFIER, value="myAdd"
      [FLEX] Line=217, token=SPECIAL, value="("
      [FLEX] Line=217, token=IDENTIFIER, value="a"
      [FLEX] Line=217, token=SPECIAL, value=","
      [FLEX] Line=217, token=IDENTIFIER, value="b"
      [FLEX] Line=217, token=SPECIAL, value=")"
      [FLEX] Line=217, token=DELIMITER, value=";"
[BISON] Line=216, expression="Κλήση συνάρτησης"



      [FLEX] Line=218, token=SPECIAL, value="}"
[BISON] Line=217, expression="Δήλωση συναρτήσεων χρήστη"
```

```
[BISON] Line=218, expression="Δήλωση συναρτήσεων χρήστη"
```

Στη δεύτερη συνάρτηση με όνομα main, αναγνωρίζονται εντολές όπως η δήλωση μεταβλητών στις γραμμές 213 και 214. Στη γραμμή 216 επίσης έχουμε την κλήση της ενσωματωμένης στη Uni-C συνάρτησης print. Τέλος, στη γραμμή 217 έχουμε την αναγνώριση της συνάρτησης myAdd ενώ ο ορισμός της συνάρτησης main γίνεται στη γραμμή 218 με τον χαρακτήρα "}".

```
      [FLEX] Line=220, token=KEYWORD, value="func"

      [FLEX] Line=220, token=IDENTIFIER, value="errorHandling"

      [FLEX] Line=220, token=SPECIAL, value="("

      [FLEX] Line=220, token=SPECIAL, value=")"



      [FLEX] Line=221, token=SPECIAL, value="{"



      [FLEX] Line=222, token=KEYWORD, value="long"

      [FLEX] Line=222, token=IDENTIFIER, value="a"

      [FLEX] Line=222, token=SPECIAL, value=","

      [FLEX] Line=222, token=IDENTIFIER, value="b"

      [FLEX] Line=222, token=SPECIAL, value=","

      [FLEX] Line=222, token=IDENTIFIER, value="c"

      [FLEX] Line=222, token=SPECIAL, value=","

      [FLEX] Line=222, token=IDENTIFIER, value="d"

      [FLEX] Line=222, token=SPECIAL, value=","

      [FLEX] Line=222, token=IDENTIFIER, value="e"

      [FLEX] Line=222, token=DELIMITER, value=";"

[BISON] Line=221, expression="Δήλωση μεταβλητής"



      [FLEX] Line=223, token=KEYWORD, value="int"

      [FLEX] Line=223, token=IDENTIFIER, value="i"
```

```
      [FLEX] Line=223, token=DELIMITER, value=";"
[BISON] Line=222, expression="Δήλωση μεταβλητής"



      [FLEX] Line=224, token=KEYWORD, value="int"
      [FLEX] Line=224, token=IDENTIFIER, value="x"
      [FLEX] Line=224, token=DELIMITER, value=";"
[BISON] Line=223, expression="Δήλωση μεταβλητής"



      [FLEX] Line=226, token=FUNCTION, value="scan"
      [FLEX] Line=226, token=SPECIAL, value="("
      [FLEX] Line=226, token=IDENTIFIER, value="a"
      [FLEX] Line=226, token=SPECIAL, value=","
!! syntax error !! -> at Line=226
      [FLEX] Line=226, token=IDENTIFIER, value="b"
      [FLEX] Line=226, token=SPECIAL, value=","
      [FLEX] Line=226, token=IDENTIFIER, value="c"
      [FLEX] Line=226, token=SPECIAL, value=","
      [FLEX] Line=226, token=IDENTIFIER, value="d"
      [FLEX] Line=226, token=SPECIAL, value=","
      [FLEX] Line=226, token=IDENTIFIER, value="e"
      [FLEX] Line=226, token=SPECIAL, value=")"
      [FLEX] Line=226, token=DELIMITER, value=";"

      [FLEX] Line=227, token=IDENTIFIER, value="pin1"
```

```
	[FLEX] Line=227, token=OPERATOR, value="="
	[FLEX] Line=227, token=SPECIAL, value="["
	[FLEX] Line=227, token=IDENTIFIER, value="a"
	[FLEX] Line=227, token=SPECIAL, value=","
	[FLEX] Line=227, token=IDENTIFIER, value="b"
	[FLEX] Line=227, token=SPECIAL, value=","
	[FLEX] Line=227, token=IDENTIFIER, value="c"
	[FLEX] Line=227, token=SPECIAL, value="]"
	[FLEX] Line=227, token=DELIMITER, value=";"

[BISON] Line=227, expression="Ανάθεση τιμής σε μεταβλητή"

	[FLEX] Line=228, token=IDENTIFIER, value="pin2"
	[FLEX] Line=228, token=OPERATOR, value="="
	[FLEX] Line=228, token=SPECIAL, value="["
	[FLEX] Line=228, token=IDENTIFIER, value="d"
	[FLEX] Line=228, token=SPECIAL, value=","
	[FLEX] Line=228, token=IDENTIFIER, value="e"
	[FLEX] Line=228, token=SPECIAL, value="]"
	[FLEX] Line=228, token=DELIMITER, value=";"

[BISON] Line=228, expression="Ανάθεση τιμής σε μεταβλητή"

	[FLEX] Line=229, token=IDENTIFIER, value="pinFinal"
	[FLEX] Line=229, token=OPERATOR, value="="
	[FLEX] Line=229, token=SPECIAL, value="["
	[FLEX] Line=229, token=IDENTIFIER, value="a"
```

```
      [FLEX] Line=229, token=SPECIAL, value=","

      [FLEX] Line=229, token=IDENTIFIER, value="b"

      [FLEX] Line=229, token=SPECIAL, value=","

      [FLEX] Line=229, token=IDENTIFIER, value="c"

      [FLEX] Line=229, token=SPECIAL, value="]"

      [FLEX] Line=229, token=OPERATOR, value="+"

!! Token error !! -> at Line=229

      8 character(s) ignored so far

      [FLEX] Line=229, token=IDENTIFIER, value="e"

!! syntax error !! -> at Line=229

      [FLEX] Line=229, token=SPECIAL, value="]"

      [FLEX] Line=229, token=DELIMITER, value=";"



      [FLEX] Line=231, token=KEYWORD, value="for"

      [FLEX] Line=231, token=SPECIAL, value="("

      [FLEX] Line=231, token=IDENTIFIER, value="i"

      [FLEX] Line=231, token=OPERATOR, value="="

      [FLEX] Line=231, token=INTEGER, value="0"

      [FLEX] Line=231, token=DELIMITER, value=";"

      [FLEX] Line=231, token=IDENTIFIER, value="i"

      [FLEX] Line=231, token=OPERATOR, value="<"

      [FLEX] Line=231, token=IDENTIFIER, value="pinFinal"

!! Token error !! -> at Line=231

!! syntax error !! -> at Line=231

      11 character(s) ignored so far

      [FLEX] Line=231, token=IDENTIFIER, value="i"
```

```
      [FLEX] Line=231, token=OPERATOR, value="++"
      [FLEX] Line=231, token=SPECIAL, value=")"

      [FLEX] Line=232, token=SPECIAL, value="{"

      [FLEX] Line=233, token=IDENTIFIER, value="x"
      [FLEX] Line=233, token=OPERATOR, value="="
      [FLEX] Line=233, token=IDENTIFIER, value="pinFinal"
      [FLEX] Line=233, token=SPECIAL, value="["
      [FLEX] Line=233, token=IDENTIFIER, value="i"
      [FLEX] Line=233, token=SPECIAL, value="]"
      [FLEX] Line=233, token=DELIMITER, value=";"
[BISON] Line=232, expression="Ανάθεση τιμής σε μεταβλητή"

      [FLEX] Line=234, token=IDENTIFIER, value="x"
      [FLEX] Line=234, token=OPERATOR, value="/="
      [FLEX] Line=234, token=INTEGER, value="2"
      [FLEX] Line=234, token=DELIMITER, value=";"
[BISON] Line=233, expression="Ανάθεση τιμής σε μεταβλητή"

      [FLEX] Line=235, token=KEYWORD, value="if"
      [FLEX] Line=235, token=SPECIAL, value="("
      [FLEX] Line=235, token=SPECIAL, value="("
      [FLEX] Line=235, token=IDENTIFIER, value="x"
      [FLEX] Line=235, token=OPERATOR, value="=="
```

```
	[FLEX] Line=235, token=INTEGER, value="0"

	[FLEX] Line=235, token=SPECIAL, value=")"

	[FLEX] Line=235, token=SPECIAL, value=")"

## Warning ## -> Double parethensis detected at Line=235


[BISON] Line=235, expression="Δήλωση if"


	[FLEX] Line=236, token=FUNCTION, value="print"

	[FLEX] Line=236, token=SPECIAL, value="("

	[FLEX] Line=236, token=STRING, value=""%d is Even\n""

	[FLEX] Line=236, token=SPECIAL, value=","

	[FLEX] Line=236, token=IDENTIFIER, value="pinFinal"

	[FLEX] Line=236, token=SPECIAL, value="["

	[FLEX] Line=236, token=IDENTIFIER, value="i"

	[FLEX] Line=236, token=SPECIAL, value="]"

	[FLEX] Line=236, token=SPECIAL, value=")"

	[FLEX] Line=236, token=DELIMITER, value=";"

[BISON] Line=235, expression="Κλήση συνάρτησης"




	[FLEX] Line=237, token=SPECIAL, value="}"


[BISON] Line=237, expression="Σύνθετες δηλώσεις"


	[FLEX] Line=238, token=SPECIAL, value="}"

!! syntax error !! -> at Line=238
```

Η τρίτη συνάρτηση που ορίζεται είναι η errorHandling, ορίζονται οι δηλώσεις μεταβλητών στις γραμμές 221,222,223. Στις γραμμές 227,228,233 ορίζονται οι πίνακες pin1,pin2,x αντίστοιχα.

Στη γραμμή 226 έχουμε ένα συντακτικό λάθος καθώς δεν ορίζεται σωστά η ενσωματωμένη συνάρτηση scan, διότι δεν μπορεί να δεχθεί παραπάνω από 1 ορίσματα. Στη γραμμή 229 έχουμε επίσης ένα συντακτικό λάθος στην πρόσθεση πινάκων γιατί ο χαρακτήρας "$" δεν αναγνωρίζεται από κάποιον γραμματικό κανόνα. Επίσης συντακτικό σφάλμα έχουμε και στην γραμμή 231, γιατί ο χαρακτήρας '.' δεν προβλέπεται στην ονομασία μεταβλητών. Στη γραμμή 234 έχουμε σωστή εκχώρηση τιμής σε μεταβλητή. Το syntax errors στις γραμμή 238 πρόκειται για bug του μεταγλωττιστή.

Στατιστικά Στοιχεία Ανάλυσης

Αρχείο Εξόδου (output.txt)

```
              ΣΤΑΤΙΣΤΙΚΑ ΣΥΝΤΑΚΤΙΚΗΣ ΑΝΑΛΥΣΗΣ


BISON -> Η συντακτική ανάλυση ολοκλήρωθηκε με αποτυχία

              (με 14 warnings)


              ΣΩΣΤΕΣ ΛΕΞΕΙΣ: 329

              ΣΩΣΤΕΣ ΕΚΦΡΑΣΕΙΣ: 116

              ΛΑΘΟΣ ΛΕΞΕΙΣ: 11

              ΛΑΘΟΣ ΕΚΦΡΑΣΕΙΣ: 20
```

Τα στατιστικά συντακτικής ανάλυσης που προκύπτουν από το αρχείο output.txt περιγράφουν την επιτυχία και τα προβλήματα που αντιμετώπισε ο αναλυτής στην ανάλυση του αρχείου input.txt με τη χρήση των εργαλείων FLEX και BISON.

Στατιστικά Συντακτικής Ανάλυσης:

1. **BISON -> Η συντακτική ανάλυση ολοκληρώθηκε με αποτυχία (με 14 warnings):**

- Αυτό αποδεικνύει ότι ο αναλυτής BISON δεν κατάφερε να ολοκληρώσει τη συντακτική ανάλυση του κώδικα.

- Τα 14 προειδοποιητικά μηνύματα δείχνουν ότι υπήρξαν κάποια συντακτικά προειδοποιητικά λάθη κατά την ανάλυση, αλλά ο κώδικας συνέχισε να εκτελείται με την εμφάνιση σωστών και λανθασμένων λέξεων και εκφράσεων.

2. **ΣΩΣΤΕΣ ΛΕΞΕΙΣ: 329**

- Αυτό δείχνει τον αριθμό των λεκτικών μονάδων που αναγνωρίστηκαν σωστά από τον ΛΑ.

3. **ΣΩΣΤΕΣ ΕΚΦΡΑΣΕΙΣ: 116**

- Αυτό αναφέρεται στον αριθμό των συντακτικών εκφράσεων που αναγνωρίστηκαν σωστά από τον ΣΑ.

4. **ΛΑΘΟΣ ΛΕΞΕΙΣ: 10**

- Παρουσιάζει τον αριθμό των λανθασμένων λεκτικών μονάδων που ανιχνεύθηκαν από τον ΛΑ.

5. **ΛΑΘΟΣ ΕΚΦΡΑΣΕΙΣ: 20**

- Παρουσιάζει τον αριθμό των λανθασμένων συντακτικών εκφράσεων που παρατηρήθηκαν από τον ΣΑ.

Οι στατιστικές αυτές παρέχουν μια καλή εικόνα των αποτελεσμάτων της συντακτικής ανάλυσης. Η χρήση των warnings μπορεί να βοηθήσει στην αντιμετώπιση προβλημάτων και τη βελτίωση της ακρίβειας του αναλυτή, ενώ οι μετρήσεις σχετικά με τις σωστές και λανθασμένες λέξεις και εκφράσεις παρέχουν σημαντική ενδεικτική πληροφορία για την ποιότητα της επεξεργασίας του κώδικα.

Ρητή αναφορά εκτέλεσης κώδικα

ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

Οι κώδικες FLEX και BISON εκτελούνται με το αρχείο Makefile που επιτρέπει την αυτόματη μεταγλώττιση και εκέλεση του κώδικα.

```
compile:
    bison -d simple-bison-code.y
    flex simple-flex-code.l
    gcc -o compiler-Uni-C simple-bison-code.tab.c lex.yy.c -lfl
    ./compiler-Uni-C input.txt output.txt
clean:
    rm simple-bison-code.tab.c simple-bison-code.tab.h lex.yy.c compiler-Uni-C
```

Ωστόσο, κατά το στάδιο της παραγωγής του κώδικα .c του BISON παρουσιάζονται ορισμένα warnings.

```
bison -d simple-bison-code.y

simple-bison-code.y: warning: 19 shift/reduce conflicts [-Wconflicts-sr]

simple-bison-code.y: warning: 135 reduce/reduce conflicts [-Wconflicts-rr]

simple-bison-code.y: note: rerun with option '-Wcounterexamples' to generate conflict
counterexamples
```

Τα προειδοποιητικά μηνύματα αναφέρουν ότι υπάρχουν συγκρούσεις στους κανόνες γραμματικής και συγκεκριμένα:

- **Shift/Reduce Conflicts**

  19 συγκρούσεις shift/reduce όπου ο ΣΑ δεν μπορεί να αποφασίσει αν πρέπει να κάνει shift (να διαβάσει το επόμενο σύμβολο εισόδου) ή reduce (να εφαρμόσει έναν κανόνα παραγωγής της γραμματικής)

- **Reduce/Reduce Conflicts**

  135 συγκρούσεις reduce/reduce όπου ο ΣΑ δεν μπορεί να αποφασίσει ποιον κανόνα παραγωγής να εφαρμόσει όταν υπάρχουν πολλοί υποψήφιοι κανόνες παραγωγής που ταιριάζουν.

Η ομάδα προσπάθησε να επιλύσει αυτές τις συγκρούσεις κάνοντας debug τον κώδικα BISON με την εντολή

```
bison -d simple-bison-code.y --verbose
```

προκειμένου να εντοπίσει σε ποιες καταστάσεις ο ΣΑ δυσκολεύεται να αποφασίσει.

Ένα ενδεικτικό παράδειγμα shift/reduce conflict που παρουσιάστηκε στο αρχείο simple-bison-code.output του debug είναι το εξής:

```
State 8


    8 type: SDOUBLE •

   12     | SDOUBLE • SDOUBLE


    SDOUBLE  shift, and go to state 56


    SDOUBLE   [reduce using rule 8 (type)]

    $default  reduce using rule 8 (type)
```

Στον κανόνα της δήλωσης μεταβλητών όπου ο χρήστης επιλέγει το keyword που θα χρησιμοποιήσει για να δηλώσει τον τύπο δεδομένων της μεταβλητής, υπάρχει σύγκρουση με τον κανόνα που χειρίζεται πιθανό συντακτικό προειδοποιητικό λάθος από το χρήστη να βάλει 2 φορές το keyword. Ο ΣΑ δεν μπορεί να αποφασίσει αν μετά το keyword double, θα διαβάσει το επόμενο σύμβολο εισόδου (shift) ή θα χρησιμοποιήσει τον κανόνα παραγωγής type (reduce).

Η ομάδα προσπάθησε να επιλύσει τα επιμέρους conflicts, αλλά δεν τα κατάφερε, καθώς, οι ήδη υπάρχοντες συντακτικές εκφράσεις που αναγνώριζουν εκφράσεις της Uni-C χρειάστηκαν τροποποιήσεις και ο χρόνος ήταν περιορισμένος για να υλοποιηθεί.

Ωστόσο, κάνοντας τον απολογισμό του ΣΑ διαπιστώσαμε τις εξής παρατηρήσεις που οδηγούν σ' αυτές τις συγκρούσεις:

1. **Πολύ Γενικοί Κανόνες**

   Μερικοί κανόνες είναι πολύ γενικοί όπως οι program, decl_statements, decl_statement και καλύπτουν πολλές διαφορετικές περιπτώσεις με κοινά τερματικά και μη-τερματικά σύμβολα.

2. **Μη Διακριτοί Κανόνες**

   Ορισμένοι κανόνες δεν διακρίνονται εύκολα μεταξύ τους λόγω της δομής τους. Για παράδειγμα, οι κανόνες assign και cmp_expr μπορούν να καταλήξουν σε conflicts λόγω της χρήσης των ίδιων tokens (var, arithm_expr)

3. **Ελλιπής Κώδικας Λειτουργίας (σε C) στους Κανόνες**

Δεν δόθηκε ιδιαίτερη βαρύτητα στην συγγραφή κατάλληλου κώδικα C για την διαχείριση των συμβόλων όταν ο ΣΑ αναγνωρίζει κάποιον συντακτικό κανόνα. Εξού και στους περισσότερους κανόνες η ομάδα έχει επιλέξει αντιγραφή της σταθεράς yytext στο μη-τερματικό σύμβολο του κανόνα με την εντολή strdup(yytext). Συνεπώς, και στις αριθμητικές πράξεις δεν υπολογίζεται το αποτέλεσμα για τον λόγο που δόθηκε στα προαναφερόμενα.

Αποτέλεσμα και των προαναφερόμενων είναι ο ΣΑ να αναγνωρίζει την έκφραση

```
func main()

{

    print("Hello World\n");

}
```

και όχι την

```
}

func main() {

    print("Hello World\n");

}
```

Η ομάδα επέλεξε από άποψη χρόνου και δυσκολίας, να δώσει μεγαλύτερη βαρύτητα στην συγγραφή συντακτικών εκφράσεων και στην διαχείριση συντακτικών προειδοποιητικών λαθών που αποτελεί και το υποχρεωτικό μέρος της εργασίας.

ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

Σας ευχαριστούμε για την προσοχή σας.