

**ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ**

**ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ**

**ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ, ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

**ΛΕΚΤΙΚΟΣ ΚΑΙ ΣΥΝΤΑΚΤΙΚΟΣ ΑΝΑΛΥΤΗΣ**

**ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ**

**Ομαδική Εργασία:**

Αναστασιάδης Αλκίνοος (20003)

Ζήνα Ελένη (20046)

Επιβλέπων:

Λάντζος Θεόδωρος

**ΣΕΡΡΕΣ****,** **2024**

Περιεχόμενα

[Εισαγωγή 3](#_Toc167568764)

[Σκοπός της εργασίας 4](#_Toc167568765)

[Μεθοδολογία 5](#_Toc167568766)

[Υλοποίηση 6](#_Toc167568767)

[Συμπεράσματα 19](#_Toc167568768)

[Μελλοντικές επεκτάσεις 19](#_Toc167568769)

[Βιβλιογραφία 20](#_Toc167568770)

# Εισαγωγή

Ο λεκτικός και συντακτικός αναλυτής είναι βασικά εργαλεία για την ανάπτυξη μεταγλωττιστών και γλωσσών προγραμματισμού. Το Flex χρησιμοποιείται για τη δημιουργία λεκτικών αναλυτών που αναγνωρίζουν μοτίβα κειμένου και τα μετατρέπουν σε tokens, ενώ το Bison χρησιμοποιείται για τη δημιουργία συντακτικών αναλυτών που οργανώνουν αυτά τα tokens σύμφωνα με συντακτικούς κανόνες της γλώσσας. Μέσω της συνδυαστικής χρήσης τους, μπορούμε να δημιουργήσουμε προγράμματα που κατανοούν και επεξεργάζονται σύνθετες γλωσσικές δομές, επιτρέποντας την ανάπτυξη πλήρως λειτουργικών μεταγλωττιστών.

# Σκοπός της εργασίας

Ο σκοπός της εργασίας μας ήταν η δημιουργία ενός απλού λεκτικού και συντακτικού αναλυτή για μια νέα γλώσσα προγραμματισμού, την Greek Programming Language (GPL). Η GPL σχεδιάστηκε με λέξεις-κλειδιά και σύνταξη βασισμένα στην ελληνική γλώσσα, προσφέροντας μια πιο προσιτή και φιλική προς τους ελληνόφωνους χρήστες εμπειρία προγραμματισμού. Μέσω της χρήσης των εργαλείων Flex και Bison, αναπτύξαμε έναν αναλυτή που αναγνωρίζει τα μοτίβα και τις συντακτικές δομές της GPL, επιτρέποντας στους προγραμματιστές να γράφουν και να εκτελούν προγράμματα στη νέα αυτή γλώσσα.

# Μεθοδολογία

Η μεθοδολογία που ακολουθήσαμε για την ανάπτυξη της GPL περιλάμβανε τη χρήση κανονικών εκφράσεων (regex) στο Flex για την αναγνώριση και εξαγωγή των λεκτικών μονάδων (tokens) από τον κώδικα που γράφτηκε στη γλώσσα μας. Αυτές οι λεκτικές μονάδες μεταβιβάστηκαν στη συντακτική ανάλυση, που υλοποιήθηκε με το Bison, όπου ορίσαμε τη σωστή σύνταξη των δομών της GPL. Μέσω αυτής της διαδικασίας, διασφαλίσαμε ότι κάθε πρόγραμμα που γράφτηκε στη GPL ακολουθεί τους προκαθορισμένους συντακτικούς κανόνες, επιτρέποντας την αναγνώριση και εκτέλεση έγκυρων προγραμμάτων στη νέα μας γλώσσα. Ο παρακάτω πίνακας δείχνει τις βασικές δομές που χρησιμοποιήσαμε στην GPL:

|  |  |  |
| --- | --- | --- |
| **Δομή** | **Περιγραφή** | **Παράδειγμα** |
| # lib | Δήλωση βιβλιοθηκών | (όπως είναι) |
| kyrio\_meros() {…} | Κύρια συνάρτηση | kyrio\_meros() {…} |
| grapse(“…”); | Εμφάνιση τιμών ή μηνυμάτων | grapse(“Enter a number:”); ή grapse(a,b,c); |
| diabase(“…”); | Εγγραφή τιμών | diabase(a,b,c); |
| epestrepse …; | Επιστροφή τιμών | epestrepse 0; ή epestrepse x; |
| (\*…\*) | Σχόλια | (\*This is a comment\*) |
| akeraios | Τύπος δεδομένων | akeraios x1,x2; |
| pragmatikos | Τύπος δεδομένων | pragmatikos y; |
| leksh | Τύπος δεδομένων | leksh name; |
| an (…) {…} alliws {…} | Δομή επανάληψης | an (x > 0) { grapse("Positive"); } alliws { grapse("Negative"); } |
| oso (…) {…} | Δομή επανάληψης | oso (x < 10) { x = x + 1; } |
| gia (…; …; …) {…} | Δομή επανάληψης | gia (i = 0; i < 10; i = i + 1) { grapse(i); } |
| +,-,\*,/,= | Αριθμητικοί τελεστές | a = b + c; |
| <,>,==,<=,>=,!= | Συγκριτικοί τελεστές | an (a != b) { grapse("Different"); } |
| (,),{,},(κόμμα),(ερωτηματικό) | Σημεία στίξης | a = (b + c) \* d; |

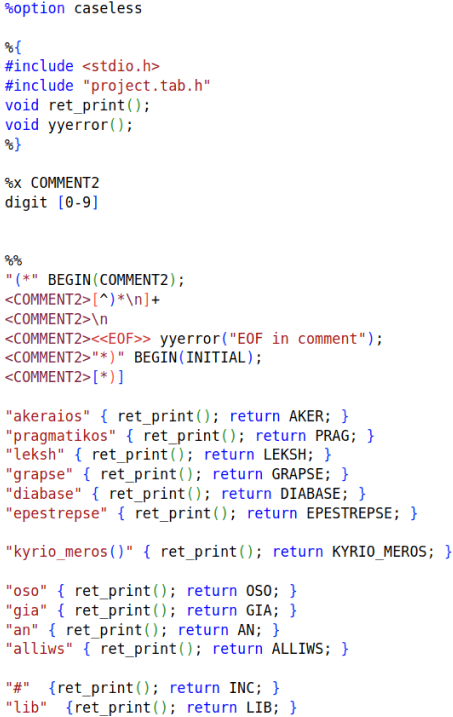
# Υλοποίηση

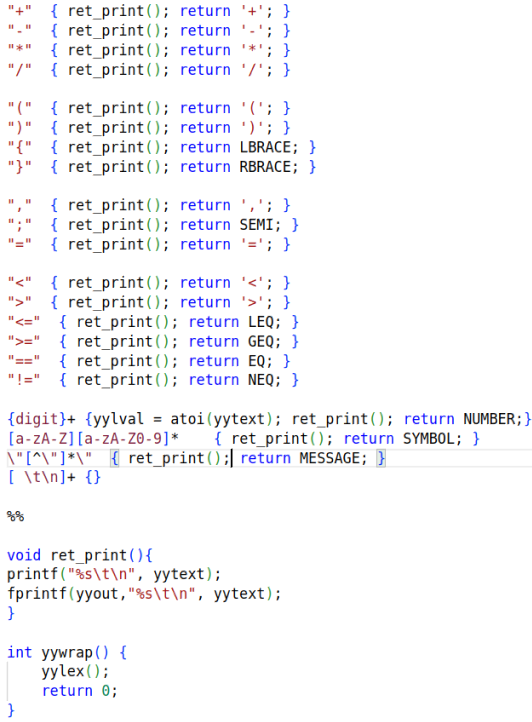
Κατά τη διαδικασία εκτέλεσης, ο λεκτικός αναλυτής διαβάζει το πρόγραμμα εισόδου και αναγνωρίζει τα διάφορα λεκτικά, ενώ ο συντακτικός αναλυτής χρησιμοποιεί τα tokens που επιστρέφονται για να αναγνωρίσει τη συντακτική δομή του προγράμματος. Ο κώδικας στον οποίο θα γίνει η ανάλυση, διαβάζεται από ένα αρχείο κειμένου και το αποτέλεσμα της ανάλυσης εκτυπώνεται στο τερματικό. Αν όλα πάνε καλά, το πρόγραμμα ολοκληρώνεται με επιτυχία, αλλιώς εμφανίζονται σφάλματα σύνταξης που πρέπει να διορθωθούν. Τα βήματα εκτέλεσης του λεκτικού και του συντακτικού αναλυτή είναι τα ακόλουθα:

1. **flex project.l**: Αυτή η εντολή χρησιμοποιεί το Flex (Fast Lexical Analyzer Generator) για να δημιουργήσει έναν λεκτικό αναλυτή από ένα αρχείο περιγραφής, το **project.l**. Το αρχείο αυτό περιέχει τους κανόνες που περιγράφουν πώς θα αναγνωρίζονται και θα επιστρέφονται τα tokens της γλώσσας προγραμματισμού που επεξεργάζεται το πρόγραμμα.
2. **bison -d project.y**: Αυτή η εντολή χρησιμοποιεί το Bison για να δημιουργήσει έναν συντακτικό αναλυτή από το αρχείο περιγραφής γραμματικής **project.y**. Αυτό το αρχείο περιέχει τους κανόνες που περιγράφουν τη σύνταξη της γλώσσας προγραμματισμού.
3. **gcc -o a.out project.tab.c lex.yy.c**: Αυτή η εντολή χρησιμοποιεί τον διερμηνέα GCC για να μεταγλωττίσει τα πηγαία αρχεία που παρήχθησαν από το Flex και το Bison, δηλαδή τα αρχεία **project.tab.c** και **lex.yy.c**, και να δημιουργήσει ένα εκτελέσιμο αρχείο που ονομάζεται **a.out**.
4. **./a.out**: Αυτή η εντολή εκτελεί το πρόγραμμα που παράχθηκε από το GCC, το εκτελέσιμο αρχείο **a.out**.

Όλος ο κώδικας, τα παραπάνω βήματα, καθώς και πολλά έγκυρα παραδείγματα εκτέλεσης, βρίσκονται στην ιστοσελίδα του GitHub όπου έχουμε το δικό μας repository [[1]](#_Βιβλιογραφία).

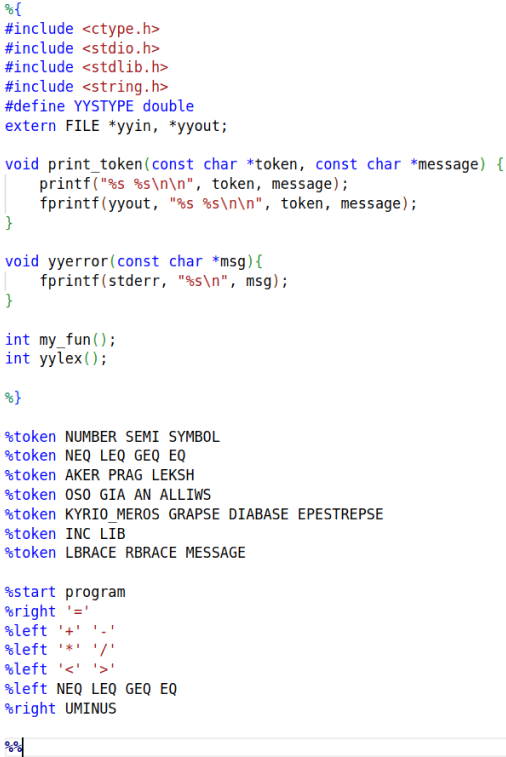
project.l:

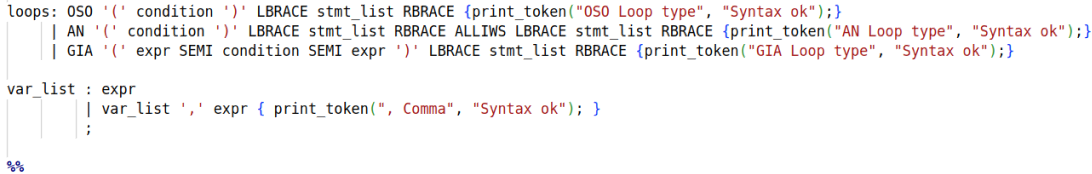
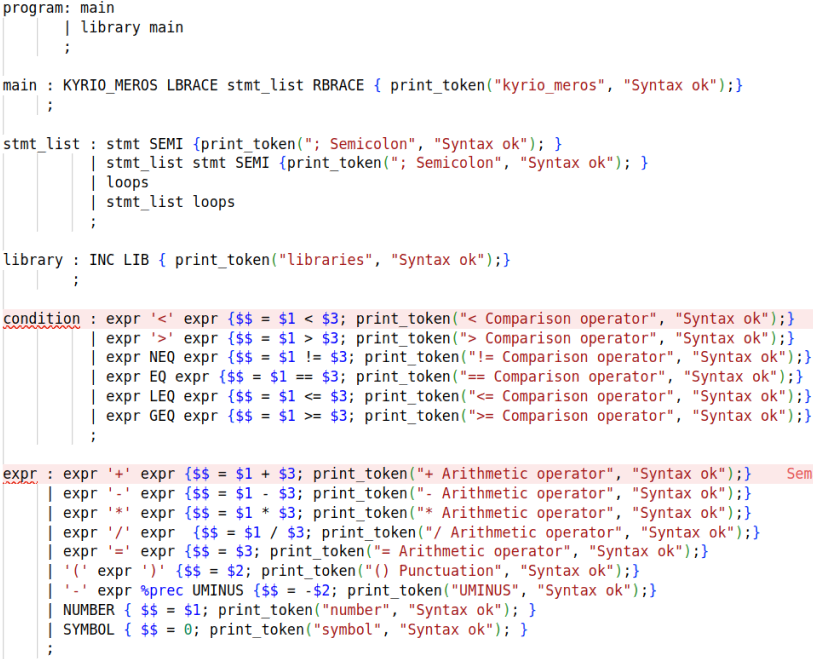
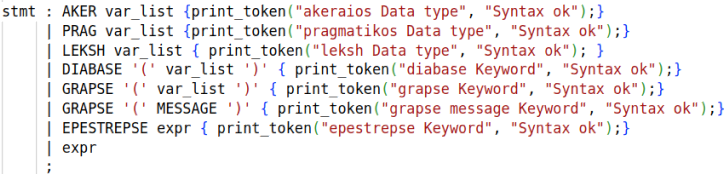


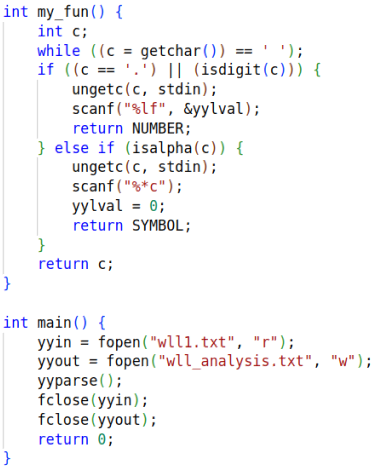


Αυτός ο κώδικας Flex ορίζει τους κανόνες για την αναγνώριση των λεκτικών μονάδων (είναι case sensitive) σε ένα πρόγραμμα γραμμένο στη GPL. Οι κανόνες περιλαμβάνουν κανονικές εκφράσεις για αναγνώριση αριθμητικών, συμβόλων, λέξεων-κλειδιών, και άλλων στοιχείων της γλώσσας. Κάθε φορά που αναγνωρίζεται ένα λεκτικό, εκτελείται μια συνάρτηση που τυπώνει το λεκτικό και στη συνέχεια το επιστρέφει στον συντακτικό αναλυτή. Επιπλέον, υπάρχει ένα μηχανισμός για αναγνώριση σχολίων που αγνοούνται κατά την ανάλυση. Αυτός ο κώδικας είναι το πρώτο βήμα στη διαδικασία ανάλυσης ενός προγράμματος.

project.y:







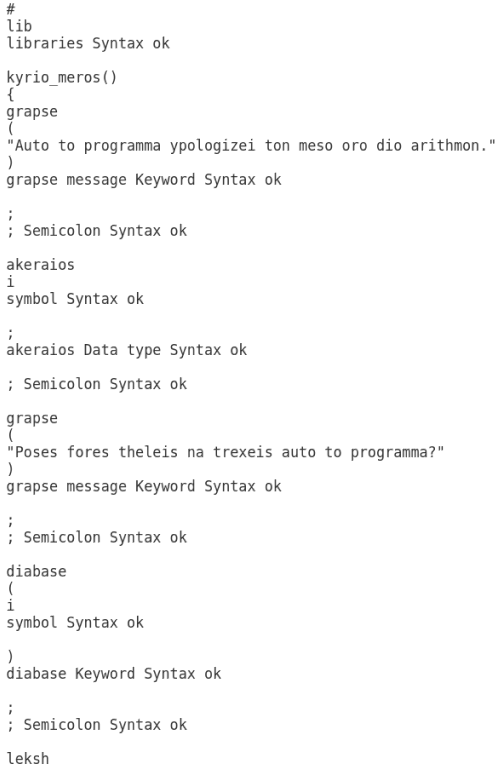
Αυτός ο κώδικας Bison ορίζει τους κανόνες για τη συντακτική ανάλυση του προγράμματος γραμμένο στη GPL. Οι κανόνες αυτοί καθορίζουν πώς πρέπει να σχηματιστούν οι δομές του προγράμματος, όπως η κύρια συνάρτηση, οι βιβλιοθήκες, οι δηλώσεις μεταβλητών, οι εντολές εκχώρησης, οι συνθήκες σύγκρισης και οι δομές επανάληψης. Κάθε φορά που αναγνωρίζεται μια δομή, εκτελείται μια συνάρτηση που τυπώνει τον τύπο της δομής και ένα μήνυμα επιβεβαίωσης (print\_token() ). Επίσης, υπάρχει και μια βοηθητική συνάρτηση my\_fun() για την αναγνώριση των διαφορετικών τύπων των λεκτικών μονάδων, αριθμούς (number) και συμβολοσειρές (symbol). Αυτός ο κώδικας είναι υπεύθυνος για την κατανόηση της δομής του προγράμματος και τη δημιουργία μιας αναπαράστασης του στο τερματικό του περιβάλλοντος του VS Code.

Παρακάτω παρατίθενται 1 έγκυρο παράδειγμα για την κατανόηση της ανάλυσης, καθώς και 1 παράδειγμα με λάθη ως προς τη συντακτική δομή των λεκτικών μονάδων:

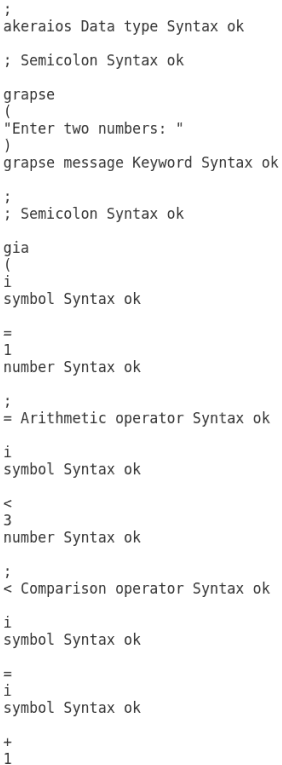
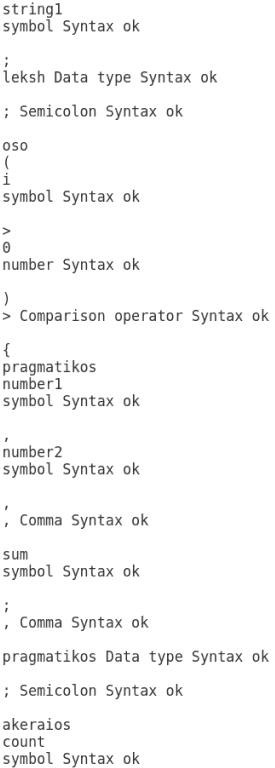
wll1.txt:



Αποτέλεσμα:

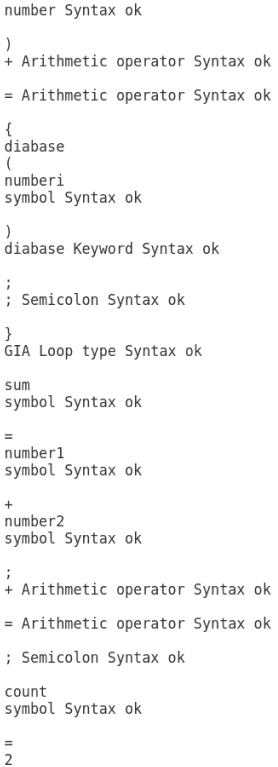
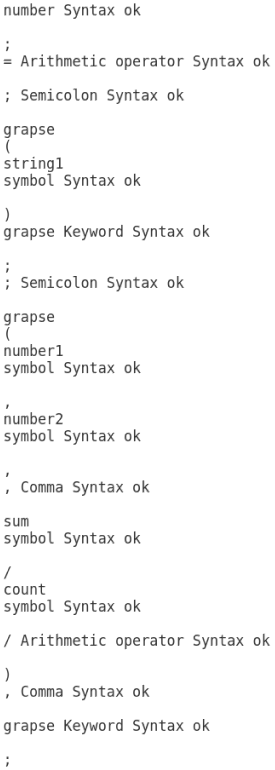


1



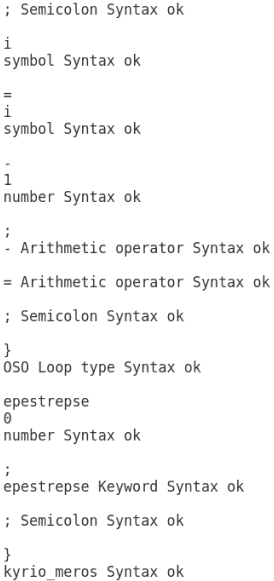
2

3



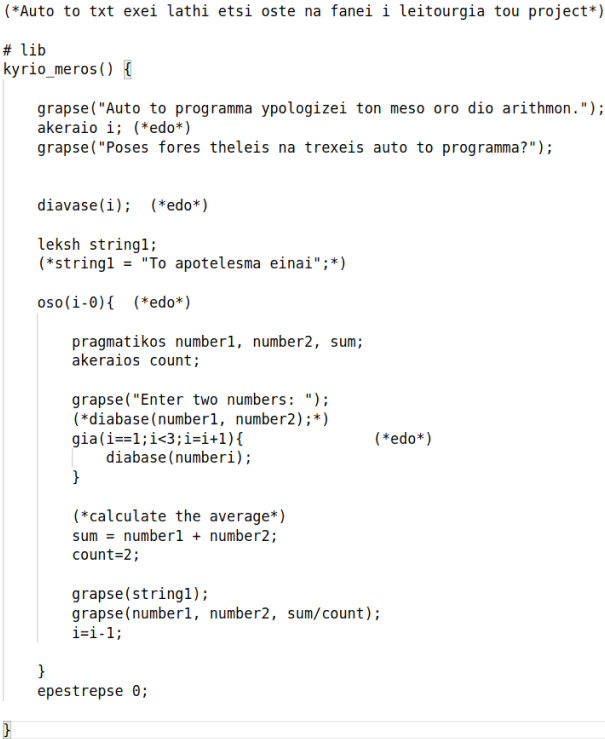
5

4

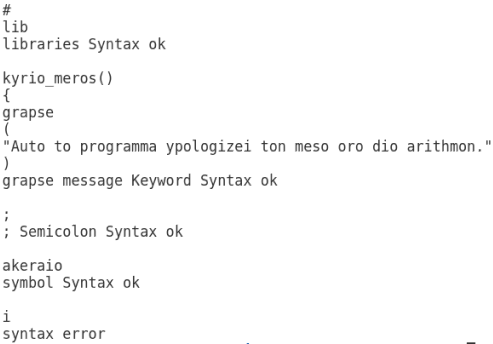


6

wll4.txt (λανθασμένο):



Αποτέλεσμα:



Στον κώδικα υπάρχουν πολλά λάθη, όμως η συντακτική ανάλυση θα ανιχνεύσει το πρώτο σφάλμα που θα δει και θα σταματήσει. Το λάθος εδώ είναι ότι ο κώδικας προσπαθεί να δηλώσει μια μεταβλητή i ως ακέραια, αλλά υπάρχει ορθογραφικό λάθος και το εκλαμβάνει ως σύμβολο, με αποτέλεσμα να μην αναγνωρίζει αυτήν την συντακτική δομή και να σταματάει.

# Συμπεράσματα

Συνολικά, η ανάπτυξη του αναλυτή για τη GPL αποτελεί μια επιτυχημένη διαδικασία. Μέσω της χρήσης εργαλείων όπως το Flex και το Bison, καταφέραμε να δημιουργήσουμε ένα εργαλείο που αναγνωρίζει και επεξεργάζεται τον κώδικα στη γλώσσα μας. Με αυτή τη διαδικασία, αποκτήσαμε εμπειρία και κατανόηση σχετικά με την ανάπτυξη γλωσσών προγραμματισμού και τη χρήση αναλυτικών εργαλείων.

# Μελλοντικές επεκτάσεις

Στο μέλλον, μπορούμε να εξετάσουμε τη δυνατότητα επέκτασης του αναλυτή για την υποστήριξη περισσότερων χαρακτηριστικών της γλώσσας, όπως η προσθήκη νέων τύπων δεδομένων ή η βελτίωση της ανίχνευσης σφαλμάτων, καθώς ο κώδικας είναι εύκολα επεκτάσιμος. Επιπλέον, μπορούμε να εξετάσουμε τη δυνατότητα ενσωμάτωσης εργαλείων για την ανάλυση σημασιολογίας, προσθέτοντας έτσι δυνατότητες όπως ο έλεγχος τύπων και η βελτιστοποίηση του κώδικα.

# Βιβλιογραφία

|  |  |
| --- | --- |
| [1] | GitHub Repository  <https://github.com/Helen1Z/lexical-and-syntax-analyzer> |
| [2] | Compiler Construction using Flex and Bison  <https://dlsiis.fi.upm.es/traductores/Software/Flex-Bison.pdf> |
| [3] | Flex – Bison Compiler  <https://github.com/nooyooj/flex-bison-compiler> |
| [4] | Introducing Flex and Bison  <https://www.oreilly.com/library/view/flex-bison/9780596805418/ch01.html> |