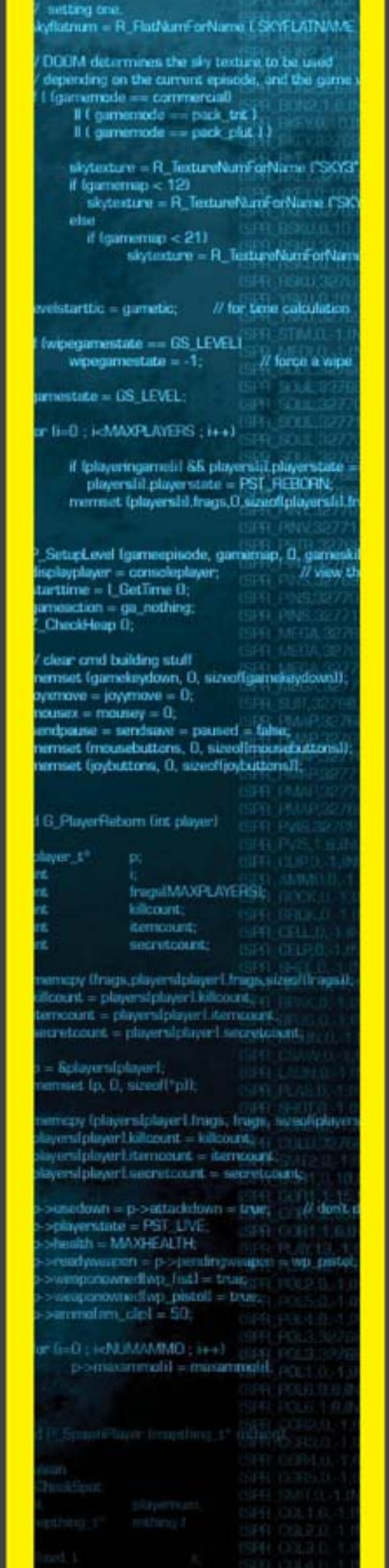
Κατασκευή περιβάλλοντος προγραμματισμού, μετάφρασης και εκτέλεσης προγραμμάτων για την ψευδογλώσσα "ΓΛΩΣΣΑ"



ΤΕΙ ΣΕΡΡΩΝ

Μπουζίκας Δημήτριος

επιβλέπων καθηγητής
Βοζίκης Χρήστος

Υπεύθυνη Δήλωση : Βεβαιώνω ότι είμαι συγγραφέας αυτής της πτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην πτυχιακή εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης βεβαιώνω ότι αυτή η πτυχιακή εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τις απαιτήσεις του προγράμματος σπουδών του Τμήματος Πληροφορικής & Επικοινωνιών του Τ.Ε.Ι. Σερρών.

Περίληψη

*Στο μάθημα της Γ΄ Λυκείου (Τεχνολογικής κατεύθυνσης) «Ανάπτυξη Εφαρμογών σε Προγραμματιστικό Περιβάλλον» διδάσκεται η ψευδογλώσσα «**ΓΛΩΣΣΑ**», μια ελληνική γλώσσα προγραμματισμού η οποία αποτελεί μια Ελληνική έκδοση της γλώσσας Pascal.*

Για την υλοποίηση της εφαρμογής χρειάστηκε να ενοποιηθούν 4 πράγματα. Ο συντάκτης (editor), ο αναλυτής του κώδικα (precompiler), ο Compiler που μετατρέπει τις εντολές που είναι σε «ΓΛΩΣΣΑ» σε εντολές Pascal και τέλος ο Compiler της Pascal όπου θα δημιουργήσει το τελικό εκτελέσιμο αρχείο ώστε να εμφανιστούν τα αποτελέσματα στην οθόνη.

Πιο συγκεκριμένα με την έναρξη της εκτέλεσης του προγράμματος, λαμβάνεται ο κώδικας που περιέχεται μέσα στον συντάκτη και αποστέλλεται σε μορφή συμβολοσειράς (string) στον CodeParser ο οποίος αναλαμβάνει να το κάνει λίστα για εύκολη διαχείριση. Έπειτα από εκεί ο κώδικας περνάει για λογική, συντακτική και σημασιολογική ανάλυση όπου θα διαπιστωθεί η ορθότητα με βάση τους κανόνες της «ΓΛΩΣΣΑΣ». Αν δεν βρεθεί κανένα σφάλμα, ο κώδικας περνάει στον Compiler για την διαδικασία της αντικατάστασης των εντολών «ΓΛΩΣΣΑΣ» με αντίστοιχες εντολές Pascal. Αφού ολοκληρωθεί και αυτό το βήμα αποθηκεύεται ο κώδικας με τις εντολές Pascal σε ένα προσωρινό αρχείο (files/pascal.pas) όπου καλείται ο Compiler της Pascal και μεταγλωττίζει το αρχείο ώστε να παραχθεί το εκτελέσιμο (files/pascal.exe) όπου στην συνέχεια θα καλεστεί, ώστε να εμφανιστεί η εφαρμογή που δημιουργήθηκε, στον τελικό χρήστη.

Summary

At lesson "Developing Applications in a programming environment" of Third Class at High School (Technological Direction) is teaching a pseudo language named "GLWSSA", a Greek programming Language which is a Greek version of Pascal.

In order to implemented this application it has to merge 4 different sections. One is the editor, which the code is writed into, next is the code analyzer which is like a precompiler, third is the Compiler for all required transactions between GLL and Pascal commands, and last is the Pascal Compiler which will create the final executable file for the results that should be displayed.

In a closer look, when program runs, code from editor will be exported as a string and sent it to CodeParser which is responsible to made this code easy accessible (vector of command line objects). After that, CodeParser sending each line for logic, syntax and semantic analyzing processes, to found out, if the code is logically and syntactically correct associated with correct meaning which is semantically acceptable. If no error found, GLL Compiler begins transactions of "GLWSSA" commands with Pascal commands. After this step completed, Pascal code will be saved in a temporary file (files/pascal.pas) which is called by the Pascal Compiler to build and make the executable file (files/pascal.exe), and finally call it to show the application at user.

Περιεχόμενα

Εισαγωγή 7

Πτυχές της «ΓΛΩΣΣΑΣ» 8

Το αλφάβητο 8

Το λεξιλόγιο 8

Το συντακτικό (syntax) 8

Η σημασιολογία (Semantics) 9

Εισαγωγή στην «ΓΛΩΣΣΑ» 10

Το Αλφάβητο 10

Το συντακτικό 10

Σταθερές 10

Μεταβλητές 11

Εντολή εκχώρησης 12

Εντολές εισόδου – εξόδου 12

Εντολές Προγράμματος 13

Εντολές / Δομές Επιλογής 13

Εντολές / Δομές Επανάληψης 15

Πίνακες 17

Εισαγωγή στην GLL 18

Εισαγωγή 18

Η Ροή του Compiler 18

Ανάλυση 20

CodeParser 20

RuleController 21

Command 21

ProgramCommand 22
DataTypeCommand 24
SelectionCommand 26
RepetitiveCommand 27
SyntaxRule 28
Εκτέλεση 38
Επίλογος 40

Εισαγωγή

Το «GLL» είναι μία εφαρμογή η οποία δέχεται κώδικα γραμμένο σε ΓΛΩΣΣΑ, τον οποίο τον ελέγχει για λογικά λάθη, έπειτα κάνει συντακτική και σημασιολογική ανάλυση γραμμή – γραμμή και αφού περάσει χωρίς κανένα σφάλμα τον μετατρέπει σε κώδικα Pascal και με την χρήση του Compiler της δημιουργεί το εκτελέσιμο και έπειτα το εμφανίζει στον χρήστη.

Η ανάπτυξη του προγράμματος έχει γίνει σε περιβάλλον C++Builder 6 και όλες οι αναλύσεις, το "parsing" το ταίριασμα των εντολών της «ΓΛΩΣΣΑΣ» έχει γίνει με την χρήση Regular Expressions και συγκεκριμένα με την χρήση της βιβλιοθήκης Deelx.

Πτυχές της «ΓΛΩΣΣΑΣ»

Το αλφάβητο

Αλφάβητο μίας γλώσσας καλείται το σύνολο των στοιχείων που χρησιμοποιείται από τη γλώσσα. Είναι το λεγόμενο primitive constructs που σημαίνει όλα τα στοιχεία που απαρτίζουν μία γλώσσα προγραμματισμού (νούμερα, χαρακτήρες, λέξεις και τελεστές/σύμβολα).

Το λεξιλόγιο

Το λεξιλόγιο αποτελείται από ένα υποσύνολο όλων των ακολουθιών που δημιουργούνται από τα στοιχεία του αλφαβήτου, τις λέξεις που είναι δεκτές από την γλώσσα. Για παράδειγμα η ακολουθία των γραμμάτων ΠΡΟΓΡΑΜΜΑ είναι δεκτή αφού αποτελεί εντολή, αλλά η ακολουθία ΠΡΟΓΡΑΜΑ δεν αποτελεί εντολή της γλώσσας, άρα δεν είναι δεκτή.

Το συντακτικό (syntax)

Η γραμματική αποτελείται από το τυπικό ή τυπολογικό (accidence) και το συντακτικό (syntax).

Τυπικό είναι το σύνολο των κανόνων που ορίζει τις μορφές με τις οποίες μία εντολή είναι αποδεκτή. Για παράδειγμα οι εντολές ΑΡΧΗ, Αρχή, αρχη είναι δεκτές, ενώ η λέξη αρχι δεν είναι αποδεκτή.

Συντακτικό είναι το σύνολο των κανόνων που καθορίζει ποιες ακολουθίες χαρακτήρων και τελεστών είναι σωστά κατανεμημένες. Για παράδειγμα η έκφραση: 3.2 + 3.2 είναι συντακτικά αποδεκτή σύμφωνα με τους κανόνες της γλώσσας ενώ η έκφραση: Πρόγραμμα αρχή τέλος δεν είναι συντακτικά αποδεκτή, για τον λόγω ότι οι εντολές δεν είναι σωστά κατανεμημένες σύμφωνα με τους κανόνες της γλώσσας.

Η γνώση του συντακτικού επιτρέπει την δημιουργία σωστών προτάσεων στις φυσικές γλώσσες ενώ στις γλώσσες προγραμματισμού τη δημιουργία σωστών εντολών.

Η σημασιολογία (Semantics)

Η σημασιολογία (Semantics) είναι το σύνολο των κανόνων που καθορίζει το νόημα των λέξεων και κατά επέκταση των εκφράσεων και προτάσεων που χρησιμοποιούνται σε μία γλώσσα.

Για παράδειγμα η πρόταση «Εγώ είμαστε ψηλός» έχει την σύνταξη: <ουσιαστικό> <ρήμα> <ουσιαστικό>, η οποία είναι συντακτικά αποδεκτή, όμως σημασιολογικά δεν είναι αποδεκτή αφού η λέξη «Εγώ» αναφέρεται σε ενικό αριθμό ενώ η λέξη «είμαστε» αναφέρεται σε πληθυντικό.

Οπότε σημασιολογικό σφάλμα μπορεί να εμφανιστεί σε μία ακολουθία εντολών που είναι συντακτικά αποδεκτή αλλά έχει λάθος νόημα. Στην γλώσσα για παράδειγμα η εντολή :

Μεταβλητή <- 2.3 / 'Δημήτρης' είναι συντακτικά αποδεκτή αφού ακολουθεί την φόρμα <variable> <- <literal><operator><literal>, όμως σημασιολογικά δεν είναι σωστή γιατί δεν μπορείς να διαιρέσεις έναν αριθμό με χαρακτήρα.

Στις γλώσσες προγραμματισμού οι οποίες είναι τεχνητές γλώσσες, ο δημιουργός της γλώσσας αποφασίζει τη σημασιολογία των λέξεων της γλώσσας.

Εισαγωγή στην «ΓΛΩΣΣΑ»

Το Αλφάβητο

Γράμματα

Κεφαλαία ελληνικού αλφαβήτου (Α-Ω)

Πεζά ελληνικού αλφαβήτου (α-ω)

Κεφαλαία λατινικού αλφαβήτου (A-Z)

Πεζά λατινικού αλφαβήτου (a-z)

Ψηφία

0 – 9

Ειδικοί χαρακτήρες

%+ - * / = ( ) . , ‘ ! : & κενός χαρακτήρας ^

Το συντακτικό

Σταθερές

Σύνταξη

```
ΣΤΑΘΕΡΕΣ
      Όνομα_1 = σταθερά_τιμή_1
      Όνομα_2 = σταθερά_τιμή_2
      .
      .
```

```
.
Όνομα_ν = σταθερά_τιμή_ν
```

Παράδειγμα

```
ΣΤΑΘΕΡΕΣ
    ΠΙ = 3.14
    ΦΠΑ = 0.23
    ΟΝΟΜΑ = 'Δημήτρης'
```

Τα ονόματα των σταθερών μπορούν να αποτελούνται από γράμματα πεζά ή κεφαλαία του ελληνικού (α-ωΑ-Ω) ή του λατινικού (a-zA-Z) αλφαβήτου, ψηφία (0–9) καθώς και τον χαρακτήρα κάτω παύλα (underscore _), ενώ πρέπει να ξεκινούν υποχρεωτικά με γράμμα. Επίσης δεν επιτρέπεται η χρησιμοποίηση δεσμευμένων λέξεων για την δήλωση σταθερών, όπως Πρόγραμμα, αρχή κτλ.

Αποδεκτές σταθερές: Όνομα, Α100, ΦΠΑ

Μη αποδεκτές: 100Α, 1ΦΠΑ

Μεταβλητές

Σύνταξη

```
ΜΕΤΑΒΛΗΤΕΣ
    τύπος_1 : λίστα_μεταβλητών_1
    τύπος_2 : λίστα_μεταβλητών_2
    .
    .
    .
    τύπος_ν : λίστα_μεταβλητών_ν
```

Παράδειγμα

```
ΜΕΤΑΒΛΗΤΕΣ
    ΠΡΑΓΜΑΤΙΚΕΣ : Σύνολο, Α
    ΑΚΕΡΑΙΕΣ : Τιμή, Ν
    ΧΑΡΑΚΤΗΡΕΣ : Όνομα
    ΛΟΓΙΚΕΣ : Έλεγχος
```

Όπως αναφερθήκαμε πριν για τα ονόματα των σταθερών και ποια ακολουθία χαρακτήρων είναι αποδεκτή, το ίδιο ισχύει και για τις μεταβλητές.

Εντολή εκχώρησης

Σύνταξη

```
Όνομα_μεταβλητής <- έκφραση
```

Παράδειγμα

```
Α <- 132
ΜΗΝΑΣ <- 'Ιανουάριος'
ΕΜΒΑΔΟΝ <- Α * Β
```

Στις εντολές εκχώρησης μεγάλη προσοχή πρέπει να δίνεται στον τύπο που προσπαθούμε να εκχωρήσουμε. Για παράδειγμα στην μεταβλητή ΜΗΝΑΣ δεν μπορούμε να βάλουμε νούμερο η μια μαθηματική έκφραση.

Εντολές εισόδου – εξόδου

Σύνταξη

```
ΔΙΑΒΑΣΕ λίστα_μεταβλητών
```

Παράδειγμα

```
ΔΙΑΒΑΣΕ Ποσότητα, Τιμή
```

Η εκτέλεση της εντολής οδηγεί στην είσοδο τιμών από το πληκτρολόγιο και την εκχώρηση τους στις μεταβλητές που αναφέρονται.

Σύνταξη

```
ΓΡΑΨΕ λίστα_στοιχείων
```

Παράδειγμα

```
ΓΡΑΨΕ 'Η τετραγωνική ρίζα του',Α ,' είναι: ',ΡΙΖΑ
```

Χρησιμοποιείται για την εμφάνιση σταθερών τιμών καθώς και των τιμών των μεταβλητών που αναφέρονται στη λίστα.

Εντολές Προγράμματος

Σύνταξη

```
ΠΡΟΓΡΑΜΜΑ τίτλος_προγραμματος
```

Παράδειγμα

```
ΠΡΟΓΡΑΜΜΑ Κόστος_Υπολογιστών
```

Η εντολή ΠΡΟΓΡΑΜΜΑ είναι η πρώτη εντολή κάθε προγράμματος ακολουθούμενη από τον τίτλο του προγράμματος.

Σύνταξη

```
ΑΡΧΗ
```

Παράδειγμα

```
ΑΡΧΗ
```

Η εντολή ΑΡΧΗ είναι η εντολή έναρξης των προγραμμάτων, και σε καμία περίπτωση δεν ακολουθεί δίπλα της άλλη εντολή ή οποιοσδήποτε χαρακτήρας εκτός του τέλους γραμμής.

Σύνταξη

```
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Παράδειγμα

```
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Η εντολή ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ είναι η εντολή τερματισμού του προγράμματος, και σε καμία περίπτωση δεν ακολουθεί δίπλα της άλλη εντολή ή οποιοσδήποτε χαρακτήρας εκτός του τέλους γραμμής.

Εντολές / Δομές Επιλογής

Σύνταξη

```
ΑΝ συνθήκη_1 ΚΑΙ συνθήκη_2 ΤΟΤΕ
      Εντολή_1
      Εντολή_2
      …
      Εντολή_ν
ΑΛΛΙΩΣ_ΑΝ συνθήκη_3 ΤΟΤΕ
      Εντολή_1
      Εντολή_2
      …
      Εντολή_ν
ΑΛΛΙΩΣ
      Εντολή_1
      Εντολή_2
      …
      Εντολή_ν
ΤΕΛΟΣ_ΑΝ
```

Παράδειγμα

```
ΑΝ αριθμός > 0 ΤΟΤΕ
      ΓΡΑΨΕ 'Ο αριθμός είναι θετικός'
ΑΛΛΙΩΣ_ΑΝ αριθμός < 0 ΤΟΤΕ
      ΓΡΑΨΕ 'Ο αριθμός είναι αρνητικός'
ΑΛΛΙΩΣ
      ΓΡΑΨΕ 'Ο αριθμός είναι 0'
ΤΕΛΟΣ_ΑΝ
```

Αν η συνθήκη ισχύει, τότε εκτελούνται οι εντολές που βρίσκονται μεταξύ των λέξεων ΤΟΤΕ και ΑΛΛΙΩΣ, διαφορετικά εκτελούνται οι εντολές μεταξύ ΑΛΛΙΩΣ και ΤΕΛΟΣ_ΑΝ. Η εκτέλεση του προγράμματος συνεχίζεται με την εντολή που ακολουθεί τη δήλωση ΤΕΛΟΣ_ΑΝ.

Σύνταξη

```
ΕΠΙΛΕΞΕ
      ΠΕΡΙΠΤΩΣΗ έκφραση_1
            Εντολές_1
      ΠΕΡΙΠΤΩΣΗ έκφραση_2
            Εντολές_2
```

```
        ……..
        ΠΕΡΙΠΤΩΣΗ ΑΛΛΙΩΣ

                Εντολές_αλλιώς
    ΤΕΛΟΣ_ΕΠΙΛΟΓΩΝ
```

Παράδειγμα

```
    ΔΙΑΒΑΣΕ αριθμός
    ΕΠΙΛΕΞΕ αριθμός
        ΠΕΡΙΠΤΩΣΗ 0
            ΓΡΑΨΕ 'Μηδέν'
        ΠΕΡΙΠΤΩΣΗ 1,3,5,7,9
            ΓΡΑΨΕ 'Ο αριθμός είναι μονός'
        ΠΕΡΙΠΤΩΣΗ 2,4,6,8
            ΓΡΑΨΕ 'Ο αριθμός είναι ζυγός'
        ΠΕΡΙΠΤΩΣΗ ΑΛΛΙΩΣ
            ΓΡΑΨΕ 'Άλλος αριθμός'
    ΤΕΛΟΣ_ΕΠΙΛΟΓΩΝ
```

Υπολογίζεται η τιμή της έκφρασης και εκτελούνται οι εντολές που ανήκουν στην αντίστοιχη περίπτωση τιμών. Αν η τιμή της έκφρασης δεν αντιστοιχεί σε καμία περίπτωση, τότε εκτελούνται οι εντολές αλλιώς.

Εντολές / Δομές Επανάληψης

Σύνταξη

```
    ΟΣΟ συνθήκη ΕΠΑΝΕΛΑΒΕ
        Εντολή_1
        Εντολή_2
        …
        Εντολή_ν
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
```

Παράδειγμα

```
    Άθροισμα <- 0
    ΟΣΟ Άθροισμα < 1000 ΕΠΑΝΕΛΑΒΕ
        ΔΙΑΒΑΣΕ Α
```

```
        Άθροισμα <- Άθροισμα + Α
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
```

Ελέγχεται η συνθήκη και αν είναι Αληθής, εκτελούνται οι εντολές που βρίσκονται ανάμεσα στις ΟΣΟ_ΕΠΑΝΕΛΑΒΕ και ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ. Στη συνέχεια ελέγχεται πάλι η συνθήκη και αν ισχύει, εκτελούνται πάλι οι ίδιες εντολές. Όταν η λογική έκφραση γίνει Ψευδής, τότε σταματάει η επανάληψη και εκτελείται η εντολή μετά το ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ.

Σύνταξη

```
ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
        Εντολή_1
        Εντολή_2
        …
        Εντολή_ν
ΜΕΧΡΙΣ_ΟΤΟΥ λογική_έκφραση
```

Παράδειγμα

```
ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
        ΔΙΑΒΑΣΕ Α
        Άθροισμα <- Άθροισμα + Α
ΜΕΧΡΙΣ_ΟΤΟΥ Άθροισμα >= 1000
```

Εκτελούνται οι εντολές μεταξύ των ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ και ΜΕΧΡΙΣ_ΟΤΟΥ. Στη συνέχεια ελέγχεται η λογική έκφραση και αν δεν ισχύει(είναι ψευδής), τότε οι εντολές που βρίσκονται ανάμεσα στις ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ και ΜΕΧΡΙΣ_ΟΤΟΥ, εκτελούνται πάλι. Ελέγχεται ξανά η λογική έκφραση και αν δεν ισχύει, επαναλαμβάνεται η εκτέλεση των ίδιων εντολών.

Σύνταξη

```
ΓΙΑ μεταβλητή ΑΠΟ τιμή1 ΜΕΧΡΙ τιμή2 ΜΕ ΒΗΜΑ βήμα
        Εντολή_1
        Εντολή_2
        …
        Εντολή_ν
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
```

Παράδειγμα

```
ΓΙΑ Αριθμό ΑΠΟ 1 ΜΕΧΡΙ 100 ΜΕ ΒΗΜΑ 2
      Άθροισμα <- Άθροισμα + Αριθμό
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
```

Οι εντολές του βρόχου εκτελούνται για όλες τις τιμές της μεταβλητής από την αρχική τιμή μέχρι την τελική τιμή, αυξανόμενες με την τιμή του βήματος. Αν το βήμα είναι ίσο με 1, τότε παραλείπεται.

Πίνακες

Δήλωση Μονοδιάστατου

```
ΜΕΤΑΒΛΗΤΕΣ
      ΠΡΑΓΜΑΤΙΚΕΣ : θερμοκρασία[30]
```

Δήλωση Πολυδιάστατου

```
ΜΕΤΑΒΛΗΤΕΣ
      ΠΡΑΓΜΑΤΙΚΕΣ : θερμοκρασία[30,10]
```

Ανάγνωση/Εγγραφή σε Μονοδιάστατο πίνακα

```
ΔΙΑΒΑΣΕ θερμοκρασία[i]
ΓΡΑΨΕ 'Η θερμοκρασία είναι: ',θερμοκρασία[i]
```

Ανάγνωση/Εγγραφή σε Πολυδιάστατο πίνακα

```
ΔΙΑΒΑΣΕ θερμοκρασία[i,j]
ΓΡΑΨΕ 'Η θερμοκρασία είναι: ',θερμοκρασία[i,j]
```

Η χρήση πινάκων είναι ένας βολικός τρόπος για τη διαχείριση πολλών δεδομένων ιδίου τύπου.

Εισαγωγή στην GLL

Εισαγωγή

Ο Compiler της GLL είναι χτισμένος ολοκληρωτικά με Κλάσεις και έχει γίνει και χρήση του γνωστού σχεδιαστικού μοτίβου MVC. Με αυτόν τον τρόπο μπορούμε πολύ εύκολα να διαχωρίσουμε το interface (views) από τους Controllers και τα Models.

Η Ροή του Compiler

Η γενική ροή του προγράμματος είναι:

- Φορτώνεται σε ένα string όλος ο κώδικας του Memo που γίνεται η συγγραφή του κώδικα
- Έπειτα ο **CodeParser** ένας Controller κάνει όλη την δουλειά ώστε να φέρει τον κώδικα σε μορφή εύκολα και γρήγορα επεξεργάσιμη,
- Συνδέεται με όλους τους άλλους Controllers που είναι υπεύθυνοι για την λογική και την συντακτική ανάλυση στέλνοντας τους κάθε φορά μία γραμμή κώδικα,
- Αν επιστραφεί λάθος από κάποιον Controller τότε σταματάει την εκτέλεση και επιστρέφει στο Front End (interface) όπου γίνεται η εμφάνιση των σφαλμάτων.
- Αν δεν βρεθεί σφάλμα τότε συνδέεται με τον Compiler ο οποίος με την σειρά του αναλαμβάνει τα transactions μεταξύ των εντολών ΓΛΩΣΣΑΣ σε εντολές Pascal.
- Με την ολοκλήρωση και του παραπάνω βήματος αποθηκεύεται το αρχείο με τις εντολές Pascal σε ένα προσωρινό αρχείο και εκτελείται το βήμα του Linker όπου καλείται ο Compiler της Pascal ώστε να δημιουργηθεί το εκτελέσιμο.

- Αφού δημιουργηθεί το εκτελέσιμο καλείται και εμφανίζεται στον χρήστη.

Ανάλυση

CodeParser

Είναι αυτός που κάνει το "parsing" του κώδικα και «συνδέεται» άμεσα με όλους τους Controllers και τα Views. Κατά βάση θεωρείται Controller γιατί συνδέεται άμεσα με Models και Views. Αποτελείται από δύο κύριες μεθόδους:

`Source *CodeParser::parseSourceCode(int totalLines, string unparsedCode)`

Αυτή η function παίρνει την συμβολοσειρά unparsedCode η οποία περιέχει όλο τον κώδικα σε μορφή string, και γνωρίζοντας τις συνολικές γραμμές (totalLines) «παρσάρεται» το string. Στην φάση αυτή γίνεται εξαγωγή της κάθε γραμμής κώδικα ξεχωριστά όπου σε συνεργασία με την Κλάση Tool καθαρίζεται η γραμμή κώδικα από περιττά κενά, tabs, comments κλπ και έπειτα γίνεται εκχώρηση ως αντικείμενο τύπου Command στο splittedCommands, ένα vector από Commands το οποίο εισάγεται ως παράμετρος στον Constructor στης Κλάσης SourceCode. Το αντικείμενο αυτό είναι η επιστρεφόμενη τιμή της συνάρτησης αυτής.

Με λίγα λόγια η παραπάνω συνάρτηση παίρνει ως όρισμα κώδικα ως string και τον εξάγει σε μορφή αντικειμένου όπου μέσα του περιέχεται μια λίστα (vector) από αντικείμενα τύπου Command.

`bool CodeParser::raiseFalses(SourceCode *sourceCode)`

Η συνάρτηση που είναι υπεύθυνη να κάνει ανάγνωση όλο το vector<Command> και για κάθε commandLine να καλέσει τους Controllers που είναι υπεύθυνοι για την Λογική, Συντακτική και σημασιολογική ανάλυση.

Επιστρέφει bool τιμές, true σε περίπτωση που βρεθούν σφάλματα όπου και σταματάει την εκτέλεση του προγράμματος. Σε αντίθετη περίπτωση επιστρέφει false οπότε έχει διαβάσει όλο το vector.

`void CodeParser::runCompiler(SourceCode *sourceCode)`

Αφού δεν έχει επιστραφεί κάποιο σφάλμα καλείται η παραπάνω συνάρτηση για να γίνει η εναλλαγή κώδικα ΓΛΩΣΣΑΣ σε κώδικα Pascal. Στην πραγματικότητα αυτή η συνάρτηση λειτουργεί ως Controller της Κλάσης Compiler.

RuleController

Μία κλάση ορίου η οποία συνδέεται με κλάσης οντότητας LogicRule, SyntaxRule & SemanticRule. Στην ουσία είναι η κύρια κλάση που «καλεί» τις κατάλληλες συναρτήσεις για την λογική, συντακτική και σημασιολογική ανάλυση του κώδικα.

`bool RuleController::logicRules(int line, string codeLine)`

Συνάρτηση υπεύθυνη να καλεί τις διαδικασίες για την λογική ανάλυση του κώδικα. Καλείται για κάθε νέα γραμμή κώδικα (codeLine).

`bool RuleController::syntaxRules(int line, string codeLine)`

Ίσως η πιο βασική συνάρτηση του προγράμματος διότι είναι αυτή που δρομολογεί το είδος της εντολής που «παρσάρετε» και σύμφωνα με αυτό καλεί τις κατάλληλες διαδικασίες για συντακτική ανάλυση και ορθότητα του κώδικα. Και αυτή καλείται επίσης για κάθε νέα γραμμή κώδικα (codeLine). Στην ουσία είναι συνάρτηση route γιατί δρομολογεί την εκτέλεση με βάση το ποια είναι η εντολή που περιέχεται μέσα στο codeLine εκείνη την στιγμή.

Command

Η κλάση Command όπου περιέχει την βασική δομή και τα attributes που χρειάζεται μία γραμμή κώδικα ώστε να δηλωθεί. Από την συγκεκριμένη κλάση κληρονομούν όλοι οι τύποι εντολών που θα αναλύσουμε παρακάτω.

ProgramCommand

Τυχαίνει το όνομα να παραπέμπει στην εντολή πρόγραμμα, αλλά είναι εξυπηρετεί όλες τις εντολές προγράμματος. Η γενική ιδέα που ακολουθήθηκε σε όλη την εργασία είναι η χρησιμοποίηση κλάσεων για σετ εντολών.

```
01:   bool ProgramCommand::isProgramCommand(string codeLine){
02:         char charCodeLine[255];
03:         strcpy(charCodeLine,codeLine.c_str());
04:
05:         CRegexpT <char> regexp(programCmd);
06:         MatchResult result = regexp.Match(charCodeLine);
07:         if(result.IsMatched())
08:               return true;
09:
10:         return false;
11:   }
```

Ανάλυση

Στην γραμμή 03 αντιγράφουμε την γραμμή κώδικα μία μεταβλητή τύπου char που έχουμε δηλώσει στην γραμμή 02. Αυτό γίνεται γιατί η βιβλιοθήκη deelx ζητάει μεταβλητή τύπου const char. Έπειτα στην γραμμή 05 γίνεται η αρχικοποίηση του regular expression που βρίσκεται στην μεταβλητή programCmd. Στην γραμμή 06 με βάση το regular expression που δημιουργήθηκε γίνεται το match και τα αποτελέσματα πάνε στο αντικείμενο result. Το οποίο χρησιμοποιείται στην γραμμή 07 για να επαληθευτεί αν βρέθηκε το pattern μέσα στο charCodeLine.

Με την χρήση regural expression οπότε επιστρέφει true σε περίπτωση που η εντολή που είναι στην αρχή μέσα στο codeLine είναι η εντολή πρόγραμμα.

Αναφορά στο regex που χρησιμοποιήθηκε:

```
char programCmd[] = "^(π|Π)(ρ|Ρ)(ο|ό|Ο|Ό)(γ|Γ)(ρ|Ρ)(α|Α)(μ|Μ){2}(α|Α)(\n| |$)";
```

^ : αρχή γραμμής

(π|Π)(ρ|Ρ) κ.ο.κ : π ή Π, ρ ή Ρ κ.ο.κ – το ότι είναι δίπλα δίπλα σημαίνει ότι μεταξύ τους δεν μπορεί να βρεθεί άλλος χαρακτήρας.

{2} : επανάληψη 2 φορές την προηγούμενη καν. έκφραση δηλ. ισοδύναμο με **(μ|Μ)(μ|Μ)**

(\n| |$)** : \n αλλαγή γραμμής, κενός χαρακτήρας (στην c είναι το κενό) , **$ τέλος γραμμής

Στην ουσία η παραπάνω κανονική έκφραση θα ταιριάσει με όλους τους συνδυασμούς που δίνουν την λέξη πρόγραμμα (πχ προγραμμα, ΠΡΟΓΡΑΜΜΑ, Πρόγραμμα κ.ο.κ.) και μετά να ακολουθεί κενό και οποιοσδήποτε χαρακτήρας ή αλλαγή γραμμής ή τέλος γραμμής.

Ομοίως έχουμε τις συναρτήσεις για τις εντολές ΑΡΧΗ & ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ αντίστοιχα :

`bool ProgramCommand::isStartCommand(string codeLine) *`

`bool ProgramCommand::isEndCommand(string codeLine) *`

`string ProgramCommand::convertProgramCommand(string codeLine)`

Η παραπάνω εντολή χρησιμοποιείται για τα transactions μεταξύ των δύο γλωσσών. Αφού κάνει match την εντολή μέσα στο codeLine την αντικαθιστά με την εντολή PROGRAM.

Ομοίως έχουμε τις συναρτήσεις για τις εντολές ΑΡΧΗ & ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ αντίστοιχα :

`string ProgramCommand::convertStartCommand(string codeLine) *`

`string ProgramCommand::convertEndCommand(string codeLine) *`

* Για λόγους συντομίας και επειδή έχουν παρόμοιες λειτουργίες η ανάλυση παραλείπεται.

DataTypeCommand

Ανάλυση

Περιλαμβάνει όλες τις εντολές που είναι για την δήλωση τύπων δεδομένων. Επίσης σ' αυτήν ανήκει και δήλωση σταθερών με την τιμή τους.

Έχει τις συναρτήσεις που ανιχνεύουν το αν η εντολή είναι μία από τις: ΣΤΑΘΕΡΕΣ, ΜΕΤΑΒΛΗΤΕΣ, ΑΚΕΡΕΑΙΕΣ, ΠΡΑΓΜΑΤΙΚΕΣ οι οποίες δουλεύουν όμοια με τις εντολές is της ProgramCommand. Υπάρχει ακόμα μία συνάρτηση που ανιχνεύει την ύπαρξη ανάθεσης τιμής σε όνομα σταθεράς, και θα αναλύσω το regular expression που είναι:

```
char setConstVar[] = "^[a-zA-Zα-ωA-ΩΆΈΉΊΌΎΏάέήίόύώς_0-9]+( )?=( )?[a-zA-Zα-ωA-ΩΆΈΉΊΌΎΏάέήίόύώς_0-9\'\"\.]*(\n| |$)";
```

^ : αρχή γραμμής

[a-zA-Z….]+ : ότι περιέχεται μέσα στα brackets θα το κάνει match, οτιδήποτε άλλο όχι. Το + δηλώνει ότι η προηγούμενη κανονική έκφραση θα βρεθεί μία φορά τουλάχιστον.

( )?=( )? : μία ή καμία φορά το κενό, μετά ακολουθεί =, και μετά πάλι το ίδιο

[a-zA-Zα-ωA-ΩΆΈΉΊΌΎΏάέήίόύώς_0-9\'\"\.]*: αυτό θα κάνει match με αριθμούς γράμματα και τα μονά και διπλά αυτάκια και την τελεία, καμία ή περισσότερες φορές.

Στην ουσία η παραπάνω κανονική έκφραση θα ταιριάσει με όλους τους συνδυασμούς :

Φπα = 0.18, Τεστ = 'δοκιμή' , Name = 'Dimitris'

Αλλά και με εσφαλμένες δηλώσεις όπως:

Test = 45 wrong, Όνομα = Δημήτρης

Οι οποίες θα αναλυθούν συντακτικά παρακάτω και θα τυπωθεί το σφάλμα.

Στην συνέχεια της συγκεκριμένης κλάσης έχουμε τα transaction:

"^(σ|Σ)(τ|Τ)(α|Α)(θ|Θ)(ε|Ε)(ρ|Ρ)(ε|έ|Έ|Ε)(σ|ς|Σ)" αντικαθίσταται με το CONST

"^(μ|Μ)(ε|Ε)(τ|Τ)(α|Α)(β|Β)(λ|Λ)(η|Η)(τ|Τ)(ε|έ|Έ|Ε)(σ|ς|Σ)" με το VAR

"^(α|Α)(κ|Κ)(ε|έ|Έ|Ε)(ρ|Ρ)(α|Α)(ι|Ι)(ε|Ε)(σ|ς|Σ)" αντικαθιστάται με το LONGINT

"^(π|Π)(ρ|Ρ)(α|Α)(γ|Γ)(μ|Μ)(α|Α)(τ|Τ)(ι|Ι)(κ|Κ)(ε|έ|Έ|Ε)(σ|ς|Σ)" με το REAL

και "^(**χ|Χ)(α|Α)(ρ|Ρ)(α|Α)(κ|Κ)(τ|Τ)(η|ή|Ή|Η)(ρ|Ρ)(ε|Ε)(σ|ς|Σ)"** με το CHAR

IOCommand

Ανάλυση

Περιλαμβάνει όλες τις εντολές που είναι εντολές εισόδου – εξόδου :

isReadCommand αν είναι η εντολή ΔΙΑΒΑΣΕ

isWriteCommand αν είναι η εντολή ΓΡΑΨΕ

isNotWriteLnCommand αυτή επιστρέφει πότε η εντολή ΓΡΑΨΕ δεν είναι με ορίσματα, δηλ αν κάνει match σημαίνει ότι είναι καθαρή ΓΡΑΨΕ που τυπώνει ένα μήνυμα μόνο.

isAssignmentCommand η εντολή εκχώρησης (Μεταβλητή <- τιμή)

Και οι συναρτήσεις μετατροπής:

convertReadCommand , convertWriteCommand , convertWriteLnCommand, convertWriteLnVariables , convertVariables , addRestRequired και addAssignRestRequired είναι οι απαραίτητες συναρτήσεις για την μετατροπή όλων των εντολών εισόδου – εξόδου σε γλώσσα Pascal.

```
ΓΡΑΨΕ 'Μήνυμα χωρίς ορίσματα'          WRITE(' Μήνυμα χωρίς ορίσματα');
ΓΡΑΨΕ 'Μήνυμα με όρισμα',όρισμα        WRITELN(' Μήνυμα με όρισμα',orisma);
ΔΙΑΒΑΣΕ Τιμή_μονάδος                   READLN(Timh_monados);
Κόστος <- Ποσότητα*Τιμή_μονάδος        Kostos := Posothta*Timh_monados;
```

SelectionCommand

Ανάλυση

Περιλαμβάνει όλες τις εντολές που είναι εντολές / δομές επιλογής :

isIfCommand αν είναι η εντολή αν

isElseCommand αν είναι η εντολή αλλιώς

isElseIfCommand αν είναι η εντολής αλλιώς_αν

isEndIfCommand αν είναι η εντολή τέλος_αν

isSwitchCommand η εντολή επίλεξε

isCaseCommand η εντολή περίπτωση και η περίπτωση αλλιώς

isEndCaseCommand η εντολή τελος_επιλογών

Και οι συναρτήσεις μετατροπής:

convertIfCommand , convertElseCommand, convertElseIfCommand και convertEndIfCommand σε συνδυασμό μετατρέπουν με 2 – 4 βήματα έναν ολόκληρο βρόχο της εντολής ΑΝ, ανεξαρτήτως τις εντολές που υπάρχουν ενδιάμεσα.

```
ΑΝ Ποσότητα < 5 ΤΟΤΕ                   IF Posothta < 5 THEN BEGIN
      Γραψε 'Μήνυμα1'                        WRITE('Μήνυμα1');
ΑΛΛΙΩΣ                                 END ELSE BEGIN
```

```
      Γραψε 'Μήνυμα2'
ΤΕΛΟΣ_ΑΝ
```

```
      WRITE('Μήνυμα2');
END;
```

Ομοίως οι συναρτήσεις: **convertSwitchCommand , convertCaseCommand και convertEndCaseCommand** μετατρέπουν βρόχους της δομής επιλογής ΕΠΙΛΕΞΕ.

```
ΕΠΙΛΕΞΕ Ποσότητα
      ΠΕΡΙΠΤΩΣΗ 1,2,3
            Γραψε 'Μήνυμα1'
      ΠΕΡΙΠΤΩΣΗ ΑΛΛΙΩΣ
            Γραψε 'Μήνυμα2'
ΤΕΛΟΣ_ΠΕΡΙΠΤΩΣΕΩΝ
```

```
CASE Posothta OF
      1,2,3: BEGIN
            WRITE('Μήνυμα1');
END;  ELSE BEGIN
            WRITE('Μήνυμα2');
      END; END;
```

RepetitiveCommand

Ανάλυση

Περιλαμβάνει όλες τις εντολές που είναι εντολές / δομές επιλογής :

isWhileCommand αν είναι η εντολή **όσο**

isEndRepeatCommand αν είναι η εντολή **τελος_επανάληψης**

isStartRepeatCommand αν είναι η εντολής **αρχή_επανάληψης**

isUntilCommand αν είναι η εντολή **μέχρις_ότου**

isForCommand η εντολή **για**

Και οι συναρτήσεις μετατροπής:

convertWhileCommand, convertEndRepeatCommand, convertStartRepeatCommand , convertUntilCommand και convertForCommand που σε συνδυασμό μετατρέπουν όλους τους τύπους επαναληπτικών διαδικασιών σε κώδικα Pascal.

Η εντολή για…από..μεχρι – τελος_επανάληψης:

```
ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 10
```

```
FOR i:=1 TO 10 DO
```

```
  ΓΙΑ j ΑΠΟ 1 ΜΕΧΡΙ 30
   ΓΡΑΨΕ 'Θερμοκρασία: ',i
    ΔΙΑΒΑΣΕ Θερμοκρασία[i,j]
  ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
```

```
  FOR j:=1 TO 10
    WRITELN(' Θερμοκρασία:', i,'',j);
    READLN(Uermokrasia[i,j]);
  END.
END;
```

Η εντολή όσο…επανέλαβε – τελος_επανάληψης:

```
i <- 1
ΟΣΟ i < 10 ΕΠΑΝΕΛΑΒΕ
  ΓΡΑΨΕ 'Η τιμή του i: ', i
  i <- i + 1
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
```

```
i := 1;
WHILE i < 10 DO BEGIN
  WRITELN('Η τιμή του i: ', i);
  i := i + 1;
END;
```

Η εντολή αρχή_επανάληψης – μέχρις_ότου:

```
i <- 1
ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
  ΓΡΑΨΕ 'Η τιμή του i: ', i
  i <- i + 1
ΜΕΧΡΙΣ_ΟΤΟΥ i > 10
```

```
i := 1;
REPEAT
  WRITELN('Η τιμή του i: ', i);
  i := i + 1;
UNTIL i > 10;
```

SyntaxRule

Ίσως η ποιο βασική Κλάση του προγράμματος. Και αυτό γιατί στην συγκεκριμένη κλάση γίνεται όλη η συντακτική ανάλυση του κώδικα. Χωρίς της οποίας δεν μπορεί να παραχθεί το εκτελέσιμο, αφού πρέπει πρώτα να διαπιστωθεί η ορθότητα του κώδικα ώστε αφού γίνει το transaction μεταξύ ΓΛΩΣΣΑΣ και Pascal να καλεστεί ο Compiler της και να παραχθεί το εκτελέσιμο.

Για λόγους συντομίας και αποφυγής άσκοπης επαναλαμβανόμενης ανάλυσης, και λόγω του μεγάλου όγκου συναρτήσεων που διαθέτει η εν λόγω Κλάση, θα

παραλειφθούν οι περισσότερες συναρτήσεις ώστε να επικεντρωθούμε στις πιο σημαντικές.

bool progCmdFirstRule(string codeLine, int line)

Ελέγχει πριν τσεκάρει με regular exp την γραμμή κώδικα αν έχουν βρεθεί εντολές πριν από αυτήν, αν ναι εμφανίζει το κατάλληλο σφάλμα.

bool progCmdSecondRule(string codeLine, int line)

Αρχικά βλέπει αν έχει «παρσαριστεί» ήδη η εντολή πρόγραμμα οπότε υπάρχει σφάλμα. Στην συνέχεια αν ταιριάξει ακριβώς με την εντολή πρόγραμμα και το τίτλο προγράμματος προχωράει σε νέο έλεγχο του τίτλου προγράμματος ώστε να εξασφαλιστεί ότι δεν είναι δεσμευμένη λέξη ή κάποια μη επιτρεπόμενη. Αν δεν ταιριάξει ακριβώς τότε προσπαθεί να βρει τι λείπει ή τι είναι επιπλέον δηλωμένο και τυπώνει το ανάλογο τύπο λάθους.

Με παρόμοια λογική λειτουργούν οι συναρτήσεις: **startCmdFirstRule, startCmdSecondRule, endCmdFirstRule, endCmdSecondRule, constCmdFirstRule, constCmdSecondRule, setConstVarFirstRule, varCmdFirstRule, varCmdSecondRule.**

```
bool setConstVarSecondRule(string codeLine, int line)
/**
  * Αν κάνει match τότε έχει 3 cases να τσεκάρει
  * 1. Να τσεκάρει αν το όνομα της σταθεράς είναι αποδεκτό
  * 2. Να τσεκάρει αν η σύνταξη είναι σωστή
  * 3. Έπειτα να δει αν το όνομα της σταθεράς έχει ήδη χρησιμοποιηθεί
  * 4. Τέλος αν περάσει όλους τους ελέγχους τοποθετείτε το όνομα της σταθεράς
  *    στο vector με τις σταθερές σύμφωνα με τον τύπο που έχει η τιμή που τις
  *    ανατίθενται.
  */
```

Σε περίπτωση που δεν ταιριάξει με την σωστή συντακτική δομή για κάθε επιπλέον λέξη ή εντολή που βρίσκεται μετά την δήλωση της σταθεράς θα τυπωθεί ένα σφάλμα.

```
bool intCmdSecondRule(string codeLine, int line);
bool floatCmdSecondRule(string codeLine, int line);
bool charCmdSecondRule(string codeLine, int line);
```

Οι τρεις παραπάνω συναρτήσεις δουλεύουν με τον ίδιο τρόπο. Αναλύουν συντακτικά την δήλωση των μεταβλητών ανά τύπο.

Θα αναλύσουμε το regular expression που χρησιμοποιήθηκε:

```
char validSyntax[] ="^(α|A)(κ|K)(ε|έ|Έ|E)(ρ|P)(α|A)(ι|I)(ε|E)(σ|ς|Σ)(
)?:( )?[a-zA-Zα-ωA-ΩΆΈΉΊΌΎΏάέήίόύώς_0-9]+(\\[[0-9,]+\\])*((( )?,(
)?[a-zA-Zα-ωA-ΩΆΈΉΊΌΎΏάέήίόύώς_0-9]+(\\[[0-9,]+\\])*)+$|$)";
```

Ανάλυση

^(α|A)(κ|K)(ε|έ|Έ|E)(ρ|P)(α|A)(ι|I)(ε|E)(σ|ς|Σ) : Ταιριάζει με την εντολή ακέραιες

( )?:( )? : Ταιριάζει με 0 ή 1 κενό : 0 ή 1 κενό

[a-zA-Zα-ωA-ΩΆΈΉΊΌΎΏάέήίόύώς_0-9]+ : όνομα μεταβλητής με 1 ή περισσότερα από αυτά

(\\[[0-9,]+\\])* : Αυτό είναι για τους πίνακες, το **\\[** σημαίνει ότι θα κάνει ακριβώς match τον χαρακτήρα [. Μετά είναι το [0-9,]+ που σημαίνει αριθμός και κόμμα μπορεί να υπάρχει μία ή περισσότερες φορές και **\\]** κλείνει η αγκύλη. Το * σημαίνει να ταιριάξει είτε υπάρχει τέτοια ακολουθία είτε όχι.

((( )?,( )?[a-zA-Zα-ωA-ΩΆΈΉΊΌΎΏάέήίόύώς_0-9]+(\\[[0-9,]+\\])*)+$|$)

Αυτό δηλώνει ότι μετά από την πρώτη μεταβλητή μπορεί να κάνει match αυτό

( )?,( )?[a-zA-Zα-ωA-ΩΆΈΉΊΌΎΏάέήίόύώς_0-9]+(\\[[0-9,]+\\])*)+$ ή να μην υπάρχει τίποτε άλλο, αλλά να είναι τέλος γραμμής $. Αν δεν είναι τέλος γραμμής θα πρέπει να ταιριάζει το ίδιο με το προηγούμενο που αναλύσαμε με την διαφορά ότι πριν από κάθε μεταβλητή πρέπει απαραίτητα να προηγείται κόμμα -> **( )?,( )?** συνοδευόμενη ή και μη από κενά. Το παραπάνω μπορεί να ταιριάξει όσες φορές χρειάζεται.

Ομοίως υπάρχουν οι αντίστοιχες κανονικές εκφράσεις για τους χαρακτήρες και για τους πραγματικούς αριθμούς.

bool readCmdSecondRule(string codeLine, int line)

Εδώ γίνεται ο έλεγχος της εντολής «ΔΙΑΒΑΣΕ» . Η εντολή αυτή ακολουθείτε πάντα από μία ή περισσότερες μεταβλητές ή πίνακες. Επομένως η κανονική έκφραση έχει μία

βασική μορφή και μετά τέλος γραμμής ή συνεχίζει με κόμμα και έπειτα μεταβλητή όσες φορές χρειάζεται.

Αρχικά ελέγχεται αν είναι συντακτικά αποδεκτή η γραμμή κώδικα. Έπειτα και εφόσον δεν υπάρχει συντακτικό σφάλμα, ελέγχονται όλες οι μεταβλητές μία προς μία αν έχουν δηλωθεί. Σε περίπτωση που βρεθεί στην λίστα μεταβλητών της εντολής ΔΙΑΒΑΣΕ μεταβλητή η οποία είναι δηλωμένη ως σταθερά τότε εμφανίζεται συντακτικό σφάλμα ότι δεν επιτρέπεται η χρήση της σταθεράς ως μεταβλητή. Αν δεν δηλωθεί η μεταβλητή τότε εμφανίζεται μήνυμα μην αναγνωρίσιμης μεταβλητής. Σε περίπτωση που η μεταβλητή είναι πίνακας ελέγχεται κάθε μετρητής που περιέχεται μέσα στις αγκύλες, για το αν έχει δηλωθεί ως μεταβλητή και αν είναι αποδεκτού τύπου. Αν δεν είναι αποδεκτού τύπου (ακέραιος) τότε εμφανίζεται σφάλμα «μην αποδεκτός τύπος μεταβλητής ως μετρητής στον πίνακα…».

Η κανονική έκφραση που χρησιμοποιήθηκε σ' αυτήν την συνάρτηση για την εγκυρότητα της σύνταξης είναι παρόμοια με την παραπάνω με την διαφορά ότι για το ταίριασμα των πινάκων χρησιμοποιεί αντί για: **(\\[[0-9,]+\\])*** την έκφραση: **(\\[[a-zA-Z,]+\\])*** και αυτό γιατί η εντολή ΔΙΑΒΑΣΕ δέχεται τιμές τις οποίες τις καταχωρεί μέσα στους πίνακες. Οπότε ο πίνακας πρέπει να έχει μέσα στις αγκύλες τον μετρητή του και όχι κάποιο νούμερο. Για παράδειγμα το σωστό είναι «ΔΙΑΒΑΣΕ Τιμές[i]» και όχι το «ΔΙΑΒΑΣΕ Τιμές[20]».

bool writeCmdSecondRule(string codeLine, int line)

Είναι η συνάρτηση για την συντακτική ανάλυση της εντολής ΓΡΑΨΕ χωρίς να ακολουθούν μεταβλητές/πίνακες. Η συγκεκριμένη εκτελείται μόνο για τις εντολές όπως «ΓΡΑΨΕ 'Μήνυμα'». Γι' αυτό τον λόγω εξηγείται ότι η κανονική έκφραση του validSyntax τελειώνει με το $, που σημαίνει ότι μετά το «μήνυμα» είναι το τέλος γραμμής.

bool writeLnCmdSecondRule(string codeLine, int line)

Βασική συνάρτηση για την ανάλυση των εντολών ΓΡΑΨΕ με μηνύματα και λίστες μεταβλητών ή πινάκων. Επειδή η σύνταξη αυτής της εντολής είναι σε μορφή:

<ΓΡΑΨΕ>
'message1',var1,'message2',var2,'message3',var3.....,'messageN',varN$

Γνωρίζοντας την κανονική έκφραση που ταιριάζει με κάθε διαφορετική οντότητα (εντολή, μήνυμα, μεταβλητή κ.λπ) δηλαδή

(γ|Γ)(ρ|P)(α|ά|Ά|A)(ψ|Ψ)(ε|E) : για την εντολή ΓΡΑΨΕ, γράψε κλπ.

(\'[a-zA-Zα-ωA-ΩΆΈΉΊΌΎΏάέήίόύώς0-9 \.\;\?\!\@\#\$\%\^\&*\(\)\:\,\-\=]+\'): για κάθε μήνυμα μέσα σε μονά αυτάκια 'Μήνυμα' και

([a-zA-Zα-ωA-ΩΆΈΉΊΌΎΏάέήίόύώς_0-9]+(\\[[a-zA-Zα-ωA-ΩΆΈΉΊΌΎΏάέήίόύώς_0-9,]+\\])*) : για κάθε μεταβλητή ή πίνακα Τιμή[i], Κόστος_1.

πρέπει να δημιουργηθεί μία κανονική έκφραση η οποία οσαδήποτε και αν είναι τα μηνύματα και οι μεταβλητές να μπορεί να ταιριάζει όταν είναι σωστή η σύνταξη της εντολής. Οπότε βήμα-βήμα δημιουργήθηκε ένα regex που να ταιριάζει με μήνυμα – μεταβλητή :

CASE 1: <γράψε> 'A message followed by variable',variable$

έπειτα επίσης να μπορεί να ταιριάξει και με μήνυμα – μεταβλητή – μήνυμα :

CASE 2: <γράψε> 'Msg1 before variable',variable,'Msg2 after variable'$

αλλά και με μήνυμα – μεταβλητή – μήνυμα – μεταβλητή :

CASE 3: <γράψε> 'Msg1 before var1',var1,'Msg2 after var1 & before var2',var2$

και τέλος με την πολλαπλή εμφάνιση μηνυμάτων και μεταβλητών. Σύμφωνα με όλα τα παραπάνω η κανονική έκφραση ταιριάζει με όλες τις πιθανές περιπτώσεις. Προστέθηκε επίσης στην έκφραση το **{0, }** που σημαίνει η κανονική έκφραση να ταιριάξει όσες φορές είναι εφικτό, αλλά και 0 αν δεν μπορεί καθόλου (το πρώτο case).

Όπως και στην «ΔΙΑΒΑΣΕ» έτσι και σ' αυτήν, κάθε μεταβλητή/πίνακας ελέγχεται για το αν έχει δηλωθεί και αν πληροί όλους τους κανόνες της ΓΛΩΣΣΑΣ.

bool assignCommandRule(string codeLine, int line);

Αυτή η συνάρτηση αναλύει συντακτικά τις εντολές εκχώρησης.

Ο πρώτος έλεγχος που γίνεται στην συνάρτηση αυτή είναι ο τύπος της μεταβλητής στην οποία γίνεται η εκχώρηση τιμής. Εμφάνιση σφάλματος γίνεται αν δεν είναι μεταβλητή αλλά σταθερά, όπου δεν επιτρέπεται εκχώρηση τιμής σε σταθερά. Επίσης ελέγχεται αν υπάρχει η μεταβλητή.

Έπειτα ελέγχεται αν πρόκειται για εκχώρηση μίας τιμής σε μεταβλητή (Κόστος <- 0, Τιμή[i] <- 10) ή εκχώρηση αποτελέσματος μαθηματικής πράξης μεταξύ

μεταβλητών/αριθμών/πινάκων (Κόστος <- Τιμή + 10). Και στις δύο περιπτώσεις γίνεται πρώτα έλεγχος για μην αναγνωρίσιμες μεταβλητές και για σημασιολογικά σφάλματα όπως ασυμφωνία τύπων. Αν για παράδειγμα προσπαθεί να γίνει εκχώρηση ενός χαρακτήρα σε μεταβλητή τύπου ακεραίου.

Εκφράσεις:

"^[a-zA-Zα-ωA-ΩΆΈΉΊΌΎΏάέήίόύώς_0-9]+(\\[[a-zA-Z,]+\\])*" : ταιριάζει με μεταβλητές και πίνακες.

"( )?<-( )?": ταιριάζει με τον τελεστή ανάθεσης (<-)

Αφού χρησιμοποιηθούν οι παραπάνω εκφράσεις για την μεταβλητή που γίνεται η ανάθεση έπειτα ακολουθούν οι εκφράσεις για τις τιμές/μεταβλητές.

(^([0-9])+$) ή : ταιριάζει μόνο τους αριθμούς ακεραίους

(^[0-9])+(\\.)?([0-9])+$) ή : ταιριάζει δεκαδικούς αριθμούς

(^\'([a-zA-Zα-ωA-ΩΆΈΉΊΌΎΏάέήίόύώς0-9])*\'$) ή : ταιριάζει χαρακτήρες

(^[a-zA-Zα-ωA-ΩΆΈΉΊΌΎΏάέήίόύώς_0-9]+(\\[[a-zA-Z,]+\\])*$): μεταβλητές/πίνακες

bool ifCmdFirstRule(string codeLine, int line)

Συνάρτηση για την ανάλυση της εντολής ΑΝ συνθήκη ΤΟΤΕ αλλά όπως επίσης και για την ΑΝ συνθήκη1 ΚΑΙ συνθήκη2 ΤΟΤΕ. Η διαδικασία σε αυτήν την συνάρτηση πάει ως εξής:

Σε πρώτη φάση και εφόσον είναι συντακτικά σωστή όλη η γραμμή κώδικα, ελέγχονται όλοι οι τελεστές σύγκρισης των συνθηκών. Σε περίπτωση που δεν είναι σωστός ο τελεστής σύγκρισης δημιουργείτε σφάλμα μη αναγνωρίσουμε τελεστή σύγκρισης.

Έπειτα με την χρήση της κ.ε :

((α|ά|Ά|A)(ν|N))|((κ|K)(α|A)(ι|I))|((τ|T)(ο|ό|Ό|O)(τ|T)(ε|E))

αφαιρούνται οι εντολές που υπάρχουν στην γραμμή και μένουν μόνο οι συνθήκες όπου με χρήση νέας κ.ε εξάγονται σε μια λίστας όπου κάθε λίστα έχει 2 string μέσα. Στο ένα έχει το αριστερό μέρος της συνθήκης και στο άλλο το δεξιό. Οπότε μετά για κάθε συνθήκη και αφού υπάρχουν σε μορφή λίστας γίνεται ο περεταίρω έλεγχος για τον αν είναι δηλωμένες μεταβλητές, τον τύπο που γίνεται η σύγκριση κλπ.

bool elseCmdFirstRule(string codeLine, int line)

Εκτελείται όταν στο codeLine υπάρχει η εντολή ΑΛΛΙΩΣ. Κάνει 2 ελέγχους. Αν καλεστεί και δεν έχει περαστεί (παρσαριστεί) η εντολή ΑΝ τότε υπάρχει σφάλμα ή αν οι συνολικές ΑΝ που είναι ανοιχτές είναι ίσες με τις ΑΝ που έχουν κλείσει τότε σημαίνει πως η εντολή ΑΛΛΙΩΣ δεν είναι μέσα σε εντολή ΑΝ, οπότε δημιουργείτε νέο σφάλμα. Σε αντίθετη περίπτωση και αφού δεν ισχύει τίποτα από τα παραπάνω κάνε ένα τελευταίο έλεγχο για το αν έχει ήδη περάσει από εντολή αλλιώς στην ίδια ΑΝ, που επίσης είναι λάθος.

bool elseCmdSecondRule(string codeLine, int line);

Σαν συνέχεια της προηγούμενης αν ταιριάζει ακριβώς με την εντολή ΑΛΛΙΩΣ και δεν υπάρχει κάτι άλλο δίπλα της συνεχίζεται η εκτέλεση κανονικά. Αν υπάρχει οτιδήποτε δημιουργείται συντακτικό σφάλμα. Ο έλεγχος για τη εξακρίβωση αν υπάρχει κάτι δίπλα από την εντολή γίνεται με την κ.ε : **.*** που σημαίνει οποιοσδήποτε χαρακτήρας εκτός του χαρακτήρα αλλαγής γραμμής.

bool elseIfCmdFirstRule(string codeLine, int line);

Γίνονται οι ίδιοι έλεγχοι όπως και στην περίπτωση της εντολής ΑΛΛΙΩΣ.

bool elseIfCmdSecondRule(string codeLine, int line);

Αυτή η συνάρτηση έχει ακριβώς την ίδια λογική και υλοποίηση με την συνάρτηση ifCmdFirstRule για τον λόγω ότι ισχύουν ακριβώς τα ίδια όσο αναφορά τους κανόνες σύνταξης και σημασιολογίας. Η μόνη διαφορά που υπάρχει είναι στην αρχική εντολή που είναι αντί της ΑΝ η ΑΛΛΙΩΣ_ΑΝ.

bool endIfCmdFirstRule(string codeLine, int line)

Εκτελείται όταν στο codeLine υπάρχει η εντολή ΤΕΛΟΣ_ΑΝ Ελέγχει αν δεν έχει δηλωθεί εντολή ΑΝ οπότε και εμφανίζει σχετικό σφάλμα ή αν οι εντολές ΑΝ που ανοίχτηκαν είναι ίσες με τις εντολές ΤΕΛΟΣ_ΑΝ (οι αν που έκλεισαν δηλ.) που σημαίνει ότι δεν υπάρχει εντολή ΑΝ ανοιχτή, πράγμα το οποίο δημιουργεί σφάλμα.

bool endIfCmdSecondRule(string codeLine, int line)

Σαν συνέχεια της προηγούμενης αν ταιριάζει ακριβώς με την εντολή ΤΕΛΟΣ_ΑΝ και δεν υπάρχει κάτι άλλο δίπλα της συνεχίζεται η εκτέλεση κανονικά. Αν υπάρχει οτιδήποτε δημιουργείται συντακτικό σφάλμα. Παρόμοια υλοποίηση με την εντολή ΑΛΛΙΩΣ.

bool switchCmdFirstRule(string codeLine, int line)

Εκτελείται όταν στο codeLine περιέχεται η εντολή ΕΠΙΛΕΞΕ. Αρχικά ελέγχεται για την ορθότητα της σύνταξης και στην συνέχεια για τον αν έχει δηλωθεί η μεταβλητή/σταθερά που έρχεται μετά την εντολή ως παράμετρος επιλογής.

bool caseCmdFirstRule(string codeLine, int line)

Εκτελείται όταν βρεθεί η εντολή ΠΕΡΙΠΤΩΣΗ ή ΠΕΡΙΠΤΩΣΗ ΑΛΛΙΩΣ ως συνέχεια της εντολής Δομής επιλογής ΕΠΙΛΕΞΕ. Αρχικά ελέγχεται αν έχει περαστεί εντολή ΕΠΙΛΕΞΕ. Αν δεν ισχύει το παραπάνω ελέγχει αν είναι η εντολή ΠΕΡΙΠΤΩΣΗ ΑΛΛΙΩΣ και επιπλέον αν έχει ήδη περαστεί αυτή η εντολή. Που σημαίνει ότι στην ίδια δομή επιλογής υπάρχουν περισσότερες από μία ΠΕΡΙΠΤΩΣΗ ΑΛΛΙΩΣ εντολές. Σε περίπτωση που δεν ισχύει τίποτα από τα παραπάνω, που σημαίνει ότι η εντολή είναι η ΠΕΡΙΠΤΩΣΗ, ελέγχει αν έχει περάσει ήδη η εντολή ΠΕΡΙΠΤΩΣΗ ΑΛΛΙΩΣ. Αν ισχύει δημιουργείται σφάλμα ότι έχει γίνεται χρήση της εντολής ΠΕΡΙΠΤΩΣΗ ενώ έχει ήδη δηλωθεί η εντολή ΠΕΡΙΠΤΩΣΗ ΑΛΛΙΩΣ.

bool caseCmdSecondRule(string codeLine, int line)

Εκτελείται μετά από τον έλεγχο της προτεραιότητας των εντολών και κάνει συντακτική και σημασιολογική ανάλυση για τις εντολές «ΠΕΡΙΠΤΩΣΗ λίστα-τιμών» αλλά και για την εντολή «ΠΕΡΙΠΤΩΣΗ ΑΛΛΙΩΣ». Σε περίπτωση που είναι η πρώτη «σπάει» την λίστα τιμών με βάση το κόμμα (,) και για κάθε μία τιμή ελέγχει αν είναι εντολές, αν είναι ίδιου τύπου με την μεταβλητή που έχει δοθεί ως παράμετρος επιλογής στην εντολή ΕΠΙΛΕΞΕ και αναλόγως δημιουργεί τα ανάλογα σφάλματα.

bool endCaseCmdFirstRule(string codeLine, int line)

Η εντολή ΤΕΛΟΣ_ΕΠΙΛΟΓΩΝ. Ελέγχει αν δεν έχει δηλωθεί καθόλου εντολή ΕΠΙΛΕΞΕ εμφανίζοντας το σχετικό σφάλμα. Γεγονός το οποίο σημαίνει ότι προσπαθεί να κλείσει μία δομή επιλογής (ΕΠΙΛΕΞΕ) που δεν έχει καν ανοιχτεί.

bool endCaseCmdSecondRule(string codeLine, int line)

Ως συνέχεια της firstRule ελέγχει αν υπάρχει άγνωστη εντολή δίπλα από την εντολή ΤΕΛΟΣ_ΕΠΙΛΟΓΩΝ. Αλλιώς ταιριάζει ακριβώς που σημαίνει ότι είναι συντακτικά σωστή.

bool whileCmdFirstRule(string codeLine, int line)

Εκτελείται για την εντολή ΟΣΟ συνθήκη ΕΠΑΝΕΛΑΒΕ. Αρχικά ελέγχεται αν είναι συντακτικά αποδεκτή η γραμμή κώδικα, αν είναι τότε για κάθε συνθήκη εξάγονται σε μία λίστα οι τελεστές και ελέγχονται αν είναι αποδεκτή. Αν δεν είναι δημιουργείται

λογικό σφάλμα. Έπειτα αφαιρούνται με την χρήση κ.ε οι εντολές ΟΣΟ και ΕΠΑΝΕΛΑΒΕ από το codeLine ούτως ώστε να μείνουν μόνο οι συνθήκες για συντακτική και σημασιολογική ανάλυση. Γίνεται ένα «σπάσιμο» των συνθηκών με βάση τον τελεστή σύγκρισης ώστε να εξαχθούν οι μεταβλητές του που χρησιμοποιούνται στην κάθε συνθήκη. Εκεί γίνεται για κάθε μεταβλητή έλεγχος για την ύπαρξη των μεταβλητών και την σωστή σύγκριση μεταξύ τους ως προς τον τύπο τους.

bool endWhileCmdFirstRule(string codeLine, int line)

Εκτελείται για την εντολή ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ. Εδώ γίνεται αρχικά έλεγχος αν έχει περαστεί η εντολή ΟΣΟ ή η ΓΙΑ. Αυτό γίνεται επειδή η εντολή αυτή είναι καθορίζει το τέλος επανάληψης και της εντολής ΟΣΟ αλλά και της εντολής ΓΙΑ. Οπότε αν καμία από τις δύο δεν έχει περαστεί πριν την εκτέλεση της δημιουργείται συντακτικό σφάλμα για εσφαλμένη χρήση της εντολής.

bool endWhileCmdSecondRule(string codeLine, int line)

Ελέγχει συντακτικά την εντολή ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ για μη αναγνωρίσιμες εντολές ή λέξεις δηλωμένες δίπλα σ' αυτήν την εντολή. Ίδια υλοποίηση με τις εντολές ΤΕΛΟΣ_ΑΝ, ΤΕΛΟΣ_ΕΠΙΛΟΓΩΝ κλπ.

bool doStartCmdFirstRule(string codeLine, int line)

Η εντολή ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ, όπως είναι στην c η εντολή do….while. Η συνάρτηση αναλαμβάνει την συντακτική ανάλυση της εντολής όπου η υλοποίηση είναι παρόμοια με αυτήν των εντολών ΤΕΛΟΣ_ΑΝ, ΤΕΛΟΣ_ΕΠΙΛΟΓΩΝ, ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ.

bool untilCmdFirstRule(string codeLine, int line)

Εκτελείται όταν στο codeLine υπάρχει η εντολή ΜΕΧΡΙΣ_ΟΤΟΥ. Η συνάρτηση χρειάζεται να κάνει μόνο έναν έλεγχο ώστε να διαπιστωθεί αν δεν έχει περαστεί καθόλου εντολή ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ. Αυτό σημαίνει σφάλμα για εσφαλμένη χρήστης της εντολής.

bool untilCmdSecondRule(string codeLine, int line)

Η ΜΕΧΡΙΣ_ΟΤΟΥ έχει παρόμοια υλοποίηση με την εντολή ΟΣΟ γιατί και οι 2 περιέχουν συνθήκες. Οπότε γίνεται έλεγχος των συγκριτών, και έπειτα των συνθηκών και των μεταβλητών κ.ο.κ.

`bool forCmdFirstRule(string codeLine, int line)`

Η συνάρτηση που εκτελείται για την εντολή ΓΙΑ τιμή ΑΠΟ αρχή ΜΕΧΡΙ τέλος. Αρχικά γίνεται έλεγχος για την συντακτική ορθότητα της εντολής. Αφού περάσει, γίνεται διαχωρισμός της εντολής με βάση το κενό όπου το codeLine μετατρέπεται πλέον σε λίστα από strings. Πρακτικά το πρώτο στοιχείο του vector (list.at(0) θα είναι η εντολή ΓΙΑ. Στο δεύτερο περιέχεται η μεταβλητή που χρησιμοποιείται στην εντολή ως μετρητής. Το τρίτο στοιχείο είναι η εντολή ΑΠΟ και στο 4 είναι η τιμή εκκίνησης της δομής. Στο 5 και στο 6 βρίσκονται η εντολή ΜΕΧΡΙ και τιμή τερματισμού αντίστοιχα.

Σύμφωνα με την παραπάνω λίστα ξεκινάει μία διαδικασία ελέγχου για την μεταβλητή, την τιμή εκκίνησης και τερματισμού. Η μεταβλητή ελέγχεται για το αν έχει δηλωθεί και μάλιστα πρέπει να είναι απαραίτητα ακεραίου τύπου ειδάλλως δημιουργείται σημασιολογικό σφάλμα «μη επιτρεπόμενος τύπος μεταβλητής ή σταθεράς ως μετρητή στην εντολή ΓΙΑ». Έπειτα γίνεται έλεγχος της εντολής ΜΕ ΒΗΜΑ αν υπάρχει στο codeLine ως προς την ορθότητα της. Τέλος ελέγχονται οι τύποι των τιμών εκκίνησης και τερματισμού της εντολής ΓΙΑ ότι είναι ακέραιοι.

Εκτέλεση

Με την ολοκλήρωση της συντακτικής ανάλυσης του κώδικα και αφού δεν υπάρχει κανένα λογικό, συντακτικό ή σημασιολογικό σφάλμα, η εκτέλεση περνάει στην φάση της μετάβασης από κώδικα «ΓΛΩΣΣΑ» σε κώδικα Pascal. Αυτή η διαδικασία εκτελείται είτε ο χρήστης πατήσει το κουμπί Μεταγλώττιση είτε το κουμπί Εκτέλεση. Η διαφορά είναι ότι στην μεταγλώττιση η εκτέλεση σταματάει αφού ολοκληρωθεί η μετατροπή, χωρίς να συνεχίζει στην κλήση του Compiler της Pascal για την δημιουργία του εκτελέσιμου.

Αφού δεν βρεθεί κάποιο σφάλμα ο CodeParser καλεί μία δικιά του εσωτερική διαδικασία, την runCompiler.

```
void CodeParser::runCompiler(SourceCode *sourceCode)
```

Ξανά διαβάζει όλα τα vector που περιέχουν μέσα τις εντολές «ΓΛΩΣΣΑΣ» και για κάθε μία εντολή καλεί τον Compiler (GLL – έναν Controller στην ουσία) δίνοντας του την κάθε γραμμή κώδικα ως παράμετρο στην συνάρτηση preparePascalCommand. Με την σειρά του μετά ο Compiler και αναγνωρίζοντας ποια εντολή περιέχεται κάθε φορά στο codeLine, εκτελεί τις αντίστοιχες συναρτήσεις convert για κάθε τύπο εντολής.

Ως επιστρεφόμενη τιμή ο Compiler περιμένει την converted ποια εντολή σε μορφή string, την οποία με την σειρά του οφείλει να αποθηκεύσει στο pascalCmds ένα vector από strings που «κρατάει» όλες τις εντολές Pascal κάθε φορά που εκτελείται το πρόγραμμα.

Με το που ολοκληρωθεί η διαδικασία της μετατροπής και αφού έχει γεμίσει η λίστα με τις εντολές Pascal, όλη η εκτέλεση επιστρέφεται στην main. Εκεί διαβάζεται το vector με τις εντολές και δημιουργείται ένα προσωρινό string με όνομα finalPas όπου γεμίζει με τις γραμμές κώδικα Pascal. Γίνεται καθαρισμός των προσωρινών αρχείων (files/pascal.exe & files/pascal.o) από προηγούμενη εκτέλεση και αποθηκεύεται το string finalPos σε ένα προσωρινό αρχείο pascal.pas στον φάκελο files. Έπειτα με την βοήθεια του system καλούμε τον Compiler της Pascal δίνοντας του ως παράμετρο το αρχείο που θέλουμε να μεταγλωττίσει και ανακατεύθυνση τα errors σ’ ένα αρχείο files/error.log.

Αφού εκτελεστεί η παραπάνω εντολή με αυτές τις παραμέτρους έχει δημιουργηθεί μέσα στον φάκελο files ένα αρχείο pascal.o & το εκτελέσιμο pascal.exe. Οπότε με την χρήση και πάλι της system κάνουμε «κλήση» του εκτελέσιμου και εμφανίζονται τα αποτελέσματα σε DOS παράθυρο, όπως αν τρέχαμε το αρχείο με ένα Virtual Pascal Compiler.

Επίλογος

Για να κλείσω θέλω να πω λίγα πράγματα σχετικά με αυτή την εργασία.

Για να είμαι ειλικρινής όταν ανέλαβα την Πτυχιακή και γνωρίζοντας μερικά πράγματα για την «ΓΛΩΣΣΑ» θεώρησα ότι θα είναι εύκολη υπόθεση. Σε αντίθεση όμως με αυτά που νόμιζα ήταν μια πολύ δύσκολη εργασία με αρκετές δεκάδες ατελείωτες ώρες αδιεξόδων. Στην πραγματικότητα όμως αυτή η εργασία με βοήθησε στο να ξαναθυμηθώ την C++, αλλά και μου αύξησε την όρεξη να ασχοληθώ πιο εκτενέστερα με τους Compilers και το Parsing.

Με αφορμή λοιπόν αυτήν την Πτυχιακή ευελπιστώ να ασχοληθώ περισσότερο με την βελτίωση αυτής της εφαρμογής σε πολύ μεγαλύτερο επίπεδο, ώστε να κατασκευαστεί ένας κανονικός Compiler για την «ΓΛΩΣΣΑ», ή και ότι άλλο προκύψει!

Για όποιον θέλει και νομίζει ότι είναι ενδιαφέρον θέμα να ασχοληθεί, το project είναι under version στο :

svn://bouzikas.com/home/bouzikas/myrepo/ptyxiaki

Βιβλιογραφία

- **Μεταγλωττιστές Γλωσσών προγραμματισμού: Θεωρία & Πράξη**

  (Κωνσταντίνος Ε. Λάζος, Παναγιώτης Θ. Κατσαρός, Ζαφείρης Καραίσκος)

- **DEELX Regular Expression Laboratory**

  (Official Regular expression Engine at http://www.regexlab.com/deelx/)

- **Cplusplus Official Website with C++ Documentation**

  (http://www.cplusplus.com/doc/)

- **Free Pascal Online Documentation**

  (http://www.freepascal.org/ - Also Downloaded Free Compiler Release: 2.4.2 Intel/i386)

- **Οπτικός Προγραμματισμός**

  (Σημειώσεις Θεωρίας Δρ. Σπύρος Καζαρλής)

- **Ανάπτυξη Εφαρμογών σε Προγραμματιστικό Περιβάλλον**

  (Αθηνά Βακάλη, Ηλίας Γιαννόπουλος, Νέστωρ Ιωαννίδης, Χρήστος Κοιλιάς)

Μπουζίκας Δημήτριος

Σιατίστης 38, Κορδελιό
Θεσσαλονίκη

+30 6939749011

dimimpou@gmail.com