# Υλοποίηση μεταγλωττιστή στην γλώσσα προγραμματισμού Cimple

Δημήτρης Βαγενάς ΑΜ: 2941

Βελτίωση και ολοκλήρωση της περσινής εργασίας με τον Κυριάκο Παπαδόπουλο ΑΜ: 3055

## Σκοπός και περιεχόμενα

Σκοπός αυτού του project είναι η δημιουργία ενός μεταφραστή μίας προγραμματιστικής γλώσσας που λέγεται Cimple. Η Cimple είναι μια μικρή, εκπαιδευτική, γλώσσα προγραμματισμού. Θυμίζει τη γλώσσα C από την οποία και αντλεί ιδέες και δομές, αλλά είναι αρκετά πιο μικρή, τόσο στις υποστηριζόμενες δομές, όσο φυσικά και σε προγραμματιστικές δυνατότητες για να εξυπηρετηθούν οι στόχοι του μαθήματος, δηλαδή η εξοικείωση των λειτουργειών του μεταγλωττιστή. Οι κανόνες και το λεξιλόγιο της γλώσσας Cimple περιγράφονται παρακάτω αναλυτικά.

## Το αλφάβητο της Cimple

**Το αλφάβητο** της γλώσσας Cimple αποτελείται από:

• τα μικρά και κεφαλαία γράμματα της λατινικής αλφαβήτου (A,…,Z και a,…,z),
• τα αριθμητικά ψηφία (0,…,9),
• τα σύμβολα των αριθμητικών πράξεων (+, -, *, /),
• τους τελεστές συσχέτισης (<, >, =, <=, >=, <>)
• το σύμβολο ανάθεσης (:=)
• τους διαχωριστές (;, “,”, :)
• τα σύμβολα ομαδοποίησης ([, ], (, ) , { , })
• του τερματισμού του προγράμματος (.)
• και διαχωρισμού σχολίων (#)

Τα σύμβολα [ , ] χρησιμοποιούνται στις λογικές παραστάσεις όπως τα σύμβολα ( , ) στις αριθμητικές παραστάσεις.

## Οι δεσμευμένες λέξεις

**Οι δεσμευμένες λέξεις της** γλώσσας Cimple είναι:
Program, declare, if, else, while, switchcase, forcase, incase, case, default, not, and, or, function, procedure, call, return, in, inout, input, print

**Οι λέξεις αυτές** δεν μπορούν να χρησιμοποιηθούν ως μεταβλητές. Οι σταθερές της γλώσσας είναι ακέραιες σταθερές που αποτελούνται από προαιρετικό πρόσημο και από μία ακολουθία αριθμητικών ψηφίων. Οι ακέραιες σταθερές πρέπει να έχουν τιμές από $-(2^{32}-1)$ έως $2^{32}-1$.

**Τα αναγνωριστικά της γλώσσας** είναι συμβολοσειρές που αποτελούνται από γράμματα και ψηφία, αρχίζοντας όμως από γράμμα. Κάθε αναγνωριστικό αποτελείται από τριάντα το πολύ γράμματα. Αναγνωριστικά με περισσότερους από 30 χαρακτήρες θεωρούνται λανθασμένα.

**Οι λευκοί χαρακτήρες** (tab, space, return, carriage return) αγνοούνται και μπορούν να χρησιμοποιηθούν με οποιονδήποτε τρόπο χωρίς να επηρεάζεται η λειτουργία του μεταγλωττιστή, αρκεί βέβαια, να μην βρίσκονται μέσα σε δεσμευμένες λέξεις, αναγνωριστικά, σταθερές .

Το ίδιο ισχύει και για τα **σχόλια**, τα οποία πρέπει να βρίσκονται ανάμεσα στα σύμβολα #.

## Μορφή προγράμματος

Κάθε πρόγραμμα ξεκινάει με τη λέξη κλειδί program. Στη συνέχεια ακολουθεί ένα αναγνωριστικό (όνομα) για το πρόγραμμα αυτό και τα τρία βασικά μπλοκ του προγράμματος: οι δηλώσεις μεταβλητών (declarations), οι συναρτήσεις και διαδικασίες (subprograms), οι οποίες μπορούν και να είναι φωλιασμένες μεταξύ τους, και οι εντολές του κυρίως προγράμματος (statements). Η δομή ενός προγράμματος Cimple φαίνεται παρακάτω.

```
program id
block
.
```

## Τύποι και δηλώσεις μεταβλητών

Ο μοναδικός τύπος δεδομένων που υποστηρίζει η Cimple είναι οι ακέραιοι αριθμοί. Η δήλωση γίνεται με την εντολή declare. Ακολουθούν τα ονόματα των αναγνωριστικών χωρίς καμία άλλη δήλωση, αφού γνωρίζουμε ότι πρόκειται για ακέραιες μεταβλητές και χωρίς να είναι αναγκαίο να βρίσκονται στην ίδια γραμμή. Οι μεταβλητές χωρίζονται μεταξύ τους με κόμματα. Το τέλος της δήλωσης αναγνωρίζεται με το ελληνικό ερωτηματικό. Επιτρέπεται να έχουμε περισσότερες των μία συνεχόμενες χρήσεις της declare.

## Τελεστές και εκφράσεις

Η προτεραιότητα των τελεστών από τη μεγαλύτερη στη μικρότερη είναι:
- Πολλαπλασιαστικοί: *, /
- Προσθετικοί: +, -
- Σχεσιακοί: =, <, >, <>, <=, >=
- Λογικοί: not
- Λογική σύζευξη: and
- Λογική διάζευξη: or

## Δομές της γλώσσας

### Εκχώρηση

ID := expression

Χρησιμοποιείται για την ανάθεση της τιμής μιας μεταβλητής ή μιας σταθεράς , ή μιας έκφρασης σε μία μεταβλητή.

### Απόφαση if

```
if (condition)
        statements1
[ else
        statements2 ]
```

Η εντολή απόφασης if εκτιμά εάν ισχύει η συνθήκη *condition* και εάν πράγματι ισχύει, τότε εκτελούνται οι εντολές *statements*1 που το ακολουθούν. Το else δεν αποτελεί υποχρεωτικό τμήμα
της εντολής και γι’ αυτό βρίσκεται σε αγκύλη. Οι εντολές *statements*2 που ακολουθούν το else εκτελούνται εάν η συνθήκη *condition* δεν ισχύει.

## Επανάληψη while

while (condition)
    statements
Η εντολή επανάληψης while επαναλαμβάνει τις εντολές *statements*, όσο η συνθήκη *condition* ισχύει. Αν την πρώτη φορά που θα αποτιμηθεί η *condition* το αποτέλεσμα της αποτίμησης είναι ψευδές, τότε οι *statements* δεν εκτελούνται ποτέ.

## Επιλογή switchcase

switchcase
    (case (condition) statements )$*$
    default statements
Η δομή switchcase ελέγχει τις *condition* που βρίσκονται μετά τα case. Μόλις μία από αυτές βρεθεί αληθής, τότε εκτελούνται οι αντίστοιχες *statements*1 (που ακολουθούν το *condition*). Μετά
ο έλεγχος μεταβαίνει έξω από την switchcase. Αν, κατά το πέρασμα, καμία από τις case δεν ισχύσει, τότε ο έλεγχος μεταβαίνει στην default και εκτελούνται οι *statements*2. Στη συνέχεια ο έλεγχος μεταβαίνει έξω από την switchcase.

## Επανάληψη forcase

forcase
    (case (condition) statements )$*$
    default statements
    Η δομή επανάληψης forcase ελέγχει τις *condition* που βρίσκονται μετά τα case. Μόλις μία από αυτές βρεθεί αληθής, τότε εκτελούνται οι αντίστοιχες *statements*1 (που ακολουθούν το *condition*). Μετά ο έλεγχος μεταβαίνει στην αρχή της forcase. Αν καμία από τις case δεν ισχύει, τότε

ο έλεγχος μεταβαίνει στη default και εκτελούνται οι *statements*2. Στη συνέχεια ο έλεγχος μεταβαίνει έξω από την forcase.

### Επανάληψη incase

incase
    (case (condition) statements )$*$

Η δομή επανάληψης incase ελέγχει τις *condition* που βρίσκονται μετά τα case, εξετάζοντας τες κατά σειρά. Για κάθε μία για τις οποίες η αντίστοιχη *condition* ισχύει εκτελούνται οι *statements*
που ακολουθούν το *condition*. Θα εξεταστούν με τη σειρά όλες οι *condition* και θα εκτελεστούν όλες οι *statements* των οποίων οι *condition* ισχύουν. Αφότου εξεταστούν όλες οι case, ο έλεγχος
μεταβαίνει έξω από τη δομή incase, εάν καμία από τις *statements* δεν έχει εκτελεστεί, ή μεταβαίνει στην αρχή της incase, έαν έστω και μία από τις *statements* έχει εκτελεστεί.

### Επιστροφή τιμής συνάρτησης

return (expression)

Χρησιμοποιείται μέσα σε συναρτήσεις για να επιστραφεί το αποτέλεσμα της συνάρτησης, το οποίο είναι το αποτέλεσμα της αποτίμησης του *expression*.

### Έξοδος δεδομένων

print (expression)

Εμφανίζει στην οθόνη το αποτέλεσμα της αποτίμησης του *expression*.

## Είσοδος δεδομένων

input (ID)

Ζητάει από τον χρήστη να δώσει μία τιμή μέσα από το πληκτρολόγιο. Η τιμή που θα δώσει θα μεταφερθεί στην μεταβλητή *ID*.

## Κλήση διαδικασίας

call functionName(actualParameters)

Καλεί μία διαδικασία.

## Συναρτήσεις και διαδικασίες

Η Cimple υποστηρίζει συναρτήσεις και διαδικασίες. Για τις συναρτήσεις η σύνταξη είναι:

```
function ID (formalparlist)
{
	block
}
```

ενώ για τις διαδικασίες:

```
procedure ID (formalparlist)
{
	block
}
```

Η formalPars είναι η λίστα των τυπικών παραμέτρων. Οι συναρτήσεις και οι διαδικασίες μπορούν να φωλιάσουν η μία μέσα στην άλλη. Οι κανόνες εμβέλειας ακολουθούν τους κανόνες της PASCAL. Η επιστροφή της τιμής μιας συνάρτησης γίνεται με την return. Η κλήση μιας συνάρτησης, γίνεται μέσα από τις αριθμητικές παραστάσεις σαν τελούμενο. Π.χ.: D = a + f(in x) όπου *f* η συνάρτηση και *x* παράμετρος που περνάει με τιμή. Η κλήση μιας διαδικασίας γίνεται με την call. Π.χ.: call

f(inout x) όπου f η διαδικασία και x παράμετρος που περνάει με αναφορά.

## Μετάδοση παραμέτρων

Η Cimple υποστηρίζει δύο τρόπους μετάδοσης παραμέτρων:

• **με τιμή**: Δηλώνεται με τη λεκτική μονάδα in. Αλλαγές στην τιμή της δεν επιστρέφονται στο πρόγραμμα που κάλεσε τη συνάρτηση

• **με αναφορά**: Δηλώνεται με τη λεκτική μονάδα inout. Κάθε αλλαγή στη τιμή της μεταφέρεται αμέσως στο πρόγραμμα που κάλεσε τη συνάρτηση Στην κλήση μίας συνάρτησης οι πραγματικοί παράμετροι συντάσσονται μετά από τις λέξεις κλειδιά in και inout, ανάλογα με το αν περνούν με τιμή ή αναφορά.

## Κανόνες εμβέλειας

**Καθολικές** ονομάζονται οι μεταβλητές που δηλώνονται στο κυρίως πρόγραμμα και είναι προσβάσιμες σε όλους. **Τοπικές** είναι οι μεταβλητές που δηλώνονται σε μία συνάρτηση ή διαδικασία και είναι προσβάσιμες μόνο μέσα από τη συγκεκριμένη συνάρτηση ή διαδικασία. Κάθε συνάρτηση ή διαδικασία, εκτός των τοπικών μεταβλητών, των παραμέτρων της και των καθολικών μεταβλητών, έχει επίσης πρόσβαση και στις μεταβλητές που έχουν δηλωθεί σε συναρτήσεις ή διαδικασίες προγόνους ή και σαν παραμέτρους αυτών.

Ισχύει ο δημοφιλής κανόνας ότι, αν δύο (ή περισσότερες) μεταβλητές ή παράμετροι έχουν το ίδιο όνομα και έχουν δηλωθεί σε διαφορετικό επίπεδο φωλιάσματος, τότε οι τοπικές μεταβλητές και παράμετροι υπερκαλύπτουν τις μεταβλητές και παραμέτρους των

προγόνων, οι οποίες με τη σειρά τους υπερκαλύπτουν τις καθολικές μεταβλητές.

Μία συνάρτηση ή διαδικασία έχει δικαίωμα να καλέσει τον εαυτό της και όποια συνάρτηση βρίσκεται στο ίδιο επίπεδο φωλιάσματος με αυτήν, της οποίας η δήλωση προηγείται στον κώδικα.

Ακολουθούν κάποια παραδείγματα, τα οποία χρησιμοποιήθηκαν κατά την υλοποίηση του μεταγλωττιστή ώστε να μπορούμε να διαπιστώσουμε την ορθή ή όχι λειτουργία των επιμέρους σταδίων του μεταγλωττιστή της γλώσσας C.

**ΠΑΡΑΔΕΙΓΜΑ 1 :**

```
program primes
{
   # declarations for main #
   declare i;
   function isPrime(in x)
   {
      # declarations for isPrime #
      declare i;
      function divides(in x, in y)
      {
         # body of divides #
         if (y = (y/x)*x)
         return (1);
         else
         return (0);
      }
      # body of isPrime #
      i:=2;
      while (i<x)
      {
         if (divides(in i, in x)=1)
         return(0);;
         i := i + 1
      };
```

```
        return(1)
    }
    # body of main #
    i := 2;
    while (i<=30)
        if (isPrime(in i)=1)
            print(i);;
}.
```

**ΠΑΡΑΔΕΙΓΜΑ 2 :**

```
program factorial
{
# declarations #
    declare x;
    declare i,fact;
    # main #

    input(x);
    fact:=1;
    i:=1;
    while (i<=x)
    {
        fact:=fact*i;
        i:=i+1;
    };
    print(fact);
}.
```

**ΠΑΡΑΔΕΙΓΜΑ 3 :**

```
program cimplefinal

{

        declare a,b,c;
        function fun(in a, inout b)
        {
                b := a+1;
                c := 4;
                return(b);
        }
        a := 1;
        c := fun(in a, inout b);
        print(c);
        print(b);
}.
```

**Η υλοποίηση ενός μεταγλωττιστή** αποτελείται από κάποια συγκεκριμένα στάδια τα οποία έχουν μορφή αλυσίδας. Δηλαδή η σωστή λειτουργία ενός σταδίου αποτελεί προϋπόθεση για την δημιουργία του επόμενου σταδίου.

## ΛΕΚΤΙΚΗ ΑΝΑΛΥΣΗ

Το πρώτο στάδιο ενός μεταγλωττιστή είναι η λεκτική ανάλυση. Η λεκτική ανάλυση αναλαμβάνει να μετατρέψει τον πηγαίο κώδικα σε σύμβολα (tokens). Κάθε token αποτελείται από τρία στοιχεία, τον τύπο, την τιμή και την γραμμή που είναι γραμμένος. Η λεκτική ανάλυση δέχεται ως είσοδο έναν-έναν τους μεμονωμένους χαρακτήρες του πηγαίου κώδικα και παράγει σύμβολα (tokens). Μια λεκτική μονάδα (token) είναι μια ακολουθία από χαρακτήρες που έχουν καταχωρηθεί ανάλογα με τους κανόνες ως κάποιο σύμβολο.

Στον λεκτικό αναλυτή της γλώσσας Cimple οι λευκοί χαρακτήρες (tab, space, return, carriage return) αγνοούνται και μπορούν να χρησιμοποιηθούν με οποιονδήποτε τρόπο χωρίς να επηρεάζεται η λειτουργία του μεταγλωττιστή, αρκεί βέβαια, να μην βρίσκονται μέσα σε

δεσμευμένες λέξεις, αναγνωριστικά, σταθερές. Η διαδικασία σχηματισμού λεκτικών μονάδων από ένα ρεύμα εισόδου χαρακτήρων ονομάζεται tokenization και ο λεκτικός αναλυτής κατηγοριοποιεί τις μονάδες ανάλογα με τον τύπο συμβόλου τους. Αν ο λεκτικός αναλυτής βρει κάποια μη επιτρεπτή λεκτική μονάδα, αναφέρει σφάλμα. Ο λεκτικός αναλυτής αφαιρεί τα σχόλια και συσχετίζει κάθε λεκτική μονάδα με την γραμμή στην οποία βρίσκεται στο αρχείο Cimple. Για παράδειγμα είναι ευθύνη του λεκτικού αναλυτή να εντοπίσει σφάλμα σε περίπτωση που τα αναγνωριστικά της γλώσσας δεν αποτελούνται μόνο από γράμματα και ψηφία ή το μέγεθός τους ξεπερνάει τους 30 χαρακτήρες. Επίσης είναι ευθύνη του λεκτικού αναλυτή να ανιχνεύσει τη λανθασμένη χρήση σχολίων.

## ΣΥΝΤΑΚΤΙΚΗ ΑΝΑΛΥΣΗ

Το δεύτερο στάδιο ενός μεταγλωττιστή είναι η συντακτική ανάλυση.

Η συντακτική ανάλυση υλοποιεί κάποιες βασικές λειτουργίες :

- Ελέγχει ότι τα tokens που παίρνει από το Λεκτικό Αναλυτή υπακούν στις συντακτικές προδιαγραφές της γλώσσας
- Υλοποιεί τη συντακτική ανάλυση μιας συγκεκριμένης γλώσσας
- Ελέγχει αν το πρόγραμμα είναι σύμφωνο με τη γραμματική της γλώσσας που υλοποιεί (αν ανήκει στη συγκεκριμένη γλώσσα)
- Σε περίπτωση συντακτικού λάθους ενημερώνει το χρήστη με μήνυμα λάθους
- Σχηματίζει και κάποια δεντρική μορφή για τις επόμενες φάσεις

Στον συντακτικό αναλυτή για κάθε έναν από τους κανόνες της γραμματικής, φτιάχνουμε και μία αντίστοιχη συνάρτηση.

Όταν συναντάμε μη τερματικό σύμβολο καλούμε το αντίστοιχο υπό-πρόγραμμα.

Όταν συναντάμε τερματικό σύμβολο, τότε:

- εάν και ο λεκτικός αναλυτής επιστρέφει λεκτική μονάδα που αντιστοιχεί στο τερματικό αυτό σύμβολο έχουμε αναγνωρίσει επιτυχώς τη λεκτική μονάδα
- αντίθετα εάν ο λεκτικός αναλυτής δεν επιστρέψει τη λεκτική μονάδα που περιμένει ο συντακτικός αναλυτής, έχουμε λάθος και εμφανίζεται το αντίστοιχο σφάλμα στην οθόνη

Όταν αναγνωριστεί και η τελευταία λεκτική μονάδα του πηγαίου προγράμματος, τότε η συντακτική ανάλυση έχει στεφτεί με επιτυχία.

Ο συντακτικός αναλυτής αναδρομικής κατάβασης αποτελείται από συναρτήσεις, μια για κάθε μη τερματικό σύμβολο, που καλούν η μια την άλλη

Ο συντακτικός αναλυτής αναδρομικής κατάβασης υλοποιεί μια συνάρτηση για κάθε κανόνα της γραμματικής.

Η κάθε συνάρτηση:

- Για τα μη τερματικά σύμβολα του κανόνα καλεί την αντίστοιχη συνάρτηση.
- Διαβάζει τα τερματικά σύμβολα του κανόνα από το λεκτικό αναλυτή.
- Αν διαβάσει ένα τερματικό σύμβολο που δεν αντιστοιχεί στον κανόνα επιστρέφει ή τερματίζει με σφάλμα.
- Αν έχει να επιλέξει ανάμεσα σε διαφορετικές παραγωγές για έναν κανόνα, διαβάζει το επόμενο token και αποφασίζει ανάλογα με αυτό.

Η δεντρική μορφή είναι κατασκευασμένη καθοδικά από πάνω προς τα κάτω (top-down), ξεκινώντας από τη ρίζα και προχωρώντας προς τα φύλλα ακολουθώντας μια αναδρομική κατάβαση.

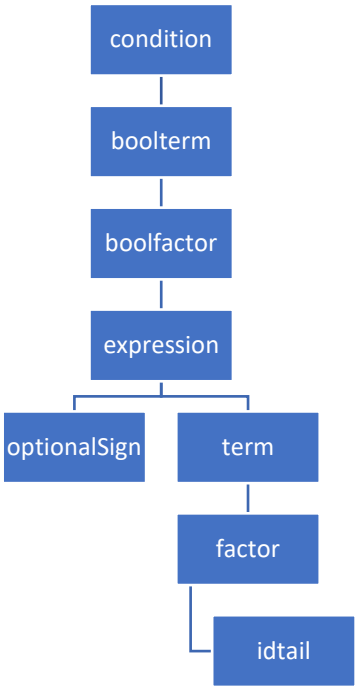
Η ρίζα του δένδρου είναι το αρχικό σύμβολο-token program με το οποίο πρέπει να ξεκινάει ένα πρόγραμμα σε γλώσσα Cimple.

Στο δεύτερο επίπεδο βρίσκεται σύμβολο-token block το οποίο διαχειρίζεται όλων των ειδών υποσυνόλων του κώδικα (declarations, subprograms, statements). Έχει τρία παιδιά την declarations την subprograms και την statements που βρίσκονται στο τρίτο επίπεδο.

Η declarations διαχειρίζεται την δήλωση μεταβλητών στην αρχή του προγράμματος. Η subprograms διαχειρίζεται όλες τις συναρτήσεις του προγράμματος. Η statements διαχειρίζεται όλα τα περιεχόμενα του κυρίου προγράμματος αλλά και των συναρτήσεων.

Στην συνέχεια παρουσιάζεται το πλήρες δένδρο που αποτελεί τον συντακτικό αναλυτή.

# ΑΠΕΙΚΟΝΙΣΗ ΔΕΝΔΡΙΚΗΣ ΜΟΡΦΗΣ ΤΟΥ ΣΥΝΤΑΚΤΙΚΟΥ ΑΝΑΛΥΤΗ


Η συνέχεια της condition

# ΕΝΔΙΑΜΕΣΟΣ ΚΩΔΙΚΑΣ

Το τρίτο στάδιο ενός μεταγλωττιστή είναι η παραγωγή του ενδιάμεσου κώδικα. Για κάθε δομή της γλώσσας προσδιορίζεται ο αντίστοιχος ενδιάμεσος κώδικας.

Ο ενδιάμεσος κώδικας είναι ένας τρόπος παράστασης του προγράμματος κοντά σε υπάρχουσες αρχιτεκτονικές υπολογιστών που παράγεται από το συντακτικό δένδρο.

Οι λόγοι που το συντακτικό δένδρο παράγει ενδιάμεσο κώδικα και όχι κατευθείαν κώδικα μηχανής είναι οι παρακάτω:

- Η μετάφραση σε κώδικα μηχανής από το συντακτικό δένδρο είναι πολύ πιο δύσκολη απ' ότι από ενδιάμεσο κώδικα
- το τμήμα του μεταγλωττιστή που περιλαμβάνει μέχρι και τη δημιουργία του ενδιάμεσου κώδικα είναι ανεξάρτητο από την αρχιτεκτονική του τελικού επεξεργαστή και μπορεί να επαναχρησιμοποιηθεί για την υλοποίηση μεταγλωττιστών για διαφορετικές αρχιτεκτονικές
- ορισμένες βελτιστοποιήσεις υλοποιούνται ευκολότερα σε ενδιάμεσο κώδικα.

Ο ενδιάμεσος κώδικας είναι ένα σύνολο από τετράδες

- ένας τελεστής
- τρία τελούμενα

Οι τετράδες είναι αριθμημένες. Κάθε τετράδα έχει μπροστά της έναν μοναδικό αριθμό που τη χαρακτηρίζει. Μόλις τελειώσει η εκτέλεση μίας τετράδας εκτελείται η τετράδα που έχει τον αμέσως μεγαλύτερο αριθμό, εκτός εάν η τετράδα που μόλις εκτελέστηκε υποδείξει κάτι διαφορετικό.

Για την παραγωγή του ενδιάμεσου κώδικα δημιουργούμε κάποιες νέες ρουτίνες για να επιτευχθεί το επιθυμητό αποτέλεσμα. Οι ρουτίνες αυτές καλούνται σε κατάλληλα σημεία των ρουτίνων του συντακτικού αναλυτή για την παραγωγή του ενδιάμεσου κώδικα. Οι ρουτίνες αυτές είναι οι εξής :

nextquad(), genquad(op, x, y, z), newtemp(), emptylist(), makelist(x), merge(list1, list2), backpatch(list,z)

Η φάση του ενδιάμεσου κώδικα τερματίζει με την παραγωγή:

- Ενός αρχείου μορφής .int όπου αποθηκεύεται ο ενδιάμεσος κώδικας που υλοποιείται στη ρουτίνα productIntFile και
- Ενός αρχείου μορφής .c από την μετατροπή του ενδιάμεσου κώδικα σε κώδικα C, αν δεν περιέχεται στο πηγαίο πρόγραμμα υπό-πρόγραμμα, που υλοποιείται στη ρουτίνα productCFile

## ΣΗΜΑΣΙΟΛΟΓΙΚΗ ΑΝΑΛΥΣΗ

Το τέταρτο στάδιο ενός μεταγλωττιστή είναι η σημασιολογική ανάλυση. Η σημασιολογική ανάλυση επιτελεί τις εξής βασικές λειτουργίες:

- Μια μεταβλητή Χ να έχει δηλωθεί πριν από την κλήση της
- Μια μεταβλητή Χ μπορεί να έχει ίδιο όνομα με άλλες μεταβλητές Χ και πρέπει να αναγνωρίζει πότε αναφερόμαστε σε ποια μεταβλητή, δηλαδή να αναγνωρίζει την εμβέλεια των μεταβλητών
- Κάθε αναγνωριστικό να έχει δηλωθεί μόνο μια φορά σε κάθε εμβέλεια
- Να γνωρίζει πόσες παραμέτρους δέχεται σαν είσοδο ένα υπό-πρόγραμμα. Οι τύποι των παραμέτρων με τις οποίες καλούνται οι συναρτήσεις είναι ακριβώς αυτοί με τις οποίες έχουν δηλωθεί και με τη σωστή σειρά
- Κάθε συνάρτηση πρέπει να έχει μέσα της τουλάχιστον ένα return, δεν πρέπει να υπάρχει return έξω από συνάρτηση
- Μέσα σε μια διαδικασία και στο κυρίως πρόγραμμα δεν επιτρέπεται να υπάρχει return

Η σημασιολογική ανάλυση αφορά τον στατικό έλεγχο του προγράμματος. Ο έλεγχος της σημασιολογικής ορθότητας και η δημιουργία κώδικα χρειάζεται να έχει γνώση των αναγνωριστικών τα οποία εμφανίζονται στο πρόγραμμα και στον τρόπο με τον οποίο τα αναγνωριστικά αυτά χρησιμοποιούνται μέσα στο πρόγραμμα.

Οι αναγκαίες αυτές πληροφορίες συλλέγονται και αποθηκεύονται στον πίνακα συμβόλων κατά την ολοκλήρωση μετάφρασης ενός block (scope).

## ΠΙΝΑΚΑΣ ΣΥΜΒΟΛΩΝ

Ο πίνακας συμβόλων συμβάλει στην επιτυχημένη υλοποίηση της σημασιολογικής ανάλυσης και προετοιμάζει το έδαφος για την παραγωγή του τελικού κώδικα. Το εγγράφημα δραστηριοποίησης του πίνακα συμβόλων έχει ως εξής:

- Δημιουργείται για κάθε συνάρτηση από αυτήν που την καλεί
- Όταν αρχίζει η εκτέλεση της συνάρτησης ο δείκτης στοίβας μεταφέρεται στην αρχή του εγγραφήματος δραστηριοποίησης
- Περιέχει πληροφορίες που χρησιμεύουν για την εκτέλεση και τον τερματισμό της συνάρτησης καθώς και πληροφορίες που σχετίζονται με τις μεταβλητές που χρησιμοποιεί
- Όταν τερματίζεται η συνάρτηση ο χώρος που καταλαμβάνει το εγγράφημα δραστηριοποίησης επιστρέφεται στο σύστημα

Όταν και αν ολοκληρωθεί η μετάφραση επιτυχώς θα δημιουργηθεί και ο ολοκληρωμένος πίνακας συμβόλων και θα τυπωθεί στο αρχείο κειμένου *cimplefile*.symb.txt

~~(Στην συνέχεια η υλοποίηση του μεταφραστή περιλαμβάνει και ένα τελικό στάδιο, το στάδιο του Τελικού Κώδικα το οποίο λόγο κακής διαχείρισης του χρόνου μας σε συνδυασμό με φόρτο διάφορων εργασιών δεν προλάβαμε να το υλοποιήσουμε. Μελλοντικά όταν υλοποιηθεί το προγραμματιστικό κομμάτι του Τελικού Κώδικα, τότε θα ενημερωθεί καταλλήλως και η αναφορά.)~~

## ΤΕΛΙΚΟΣ ΚΩΔΙΚΑΣ

Η τελευταία φάση της παραγωγής κώδικα είναι η παραγωγή του τελικού κώδικα. Ο τελικός κώδικας προκύπτει από τον ενδιάμεσο κώδικα με τη βοήθεια του πίνακα συμβόλων. Συγκεκριμένα, από κάθε εντολή ενδιάμεσου κώδικα προκύπτει μία σειρά εντολών τελικού κώδικα, η οποία για να παραχθεί ανακτά πληροφορίες από τον πίνακα συμβόλων.

Ο τελικός κώδικας είναι εντολές σε γλώσσα μηχανής (assembly code) και για αυτόν τον μεταφραστή παράγεται κώδικας για τον επεξεργαστή RISC-V.

Τέλος, με την ολοκλήρωση της μετάφρασης δημιουργείτε το αρχείο με τον κώδικα assembly με την ονομασία *cimplefile*.asm.

## ΕΠΙΛΟΓΟΣ

Σε αυτό το σημείο, ο μεταγλωττιστής της γλώσσας Cimple έχει υλοποιηθεί. Καταφέραμε να δημιουργήσουμε έναν μεταγλωττιστή για μία απλή γλώσσα που τυπικά δεν υπάρχει, με αποτέλεσμα αν κάποιος ακολουθήσει το συντακτικό και τους κανόνες της να μπορέσει να υλοποιήσει και να εκτελέσει απλά προγράμματα και να πάρει τα αποτελέσματά τους.

Ακολουθεί το γνωστό documentation με την επεξήγηση κάθε συνάρτησης του προγράμματος.

## ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΟ ΚΟΜΜΑΤΙ

Αρχικά, **η συνάρτηση lex** αποτελεί τον λεκτικό αναλυτή. Ο λεκτικός αναλυτής όποτε καλείται διαβάζει από το αρχείο του προγράμματος Cimple και είτε επιστρέφει το επόμενο διαβασμένο στοιχείο είτε εκτυπώνει την ύπαρξη λεκτικού λάθους και τερματίζει το πρόγραμμα. Αρχικά έχουμε ορίσει στον lex κάποιες λίστες με τα βασικά στοιχεία της γλώσσας Cimple ώστε να προσθέτουμε στο στοιχείο που επιστρέφουμε τα κατάλληλα χαρακτηριστικά.

Το στοιχείο που επιστρέφει ο λεκτικός αναλυτής είναι ένα αντικείμενο με την ονομασία Token το οποίο έχει τα πεδία tokenType, tokenString, lineNo. Όπου tokenType είναι το token της λεκτικής μονάδας (π.χ. 'addtk' σε περίπτωση που έχουμε να κάνουμε με '+' ή '-'), tokenString είναι το ίδιο το αλφαριθμητικό που διαβάστηκε (πχ 'program' ή '+' ) και lineNo είναι η γραμμή από την οποία διαβάστηκε.

Σε περίπτωση που το επόμενο διαβασμένο στοιχείο είναι κάποιος κενός χαρακτήρας ή σχόλια του προγράμματος, τα παραβλέπουμε και επιστρέφουμε το επόμενο στοιχειό μετά από αυτά.

Όποτε καλείται ο lex για να επιστρέψει ένα αντικείμενο token γίνεται και η λεκτική ανάλυση ταυτόχρονα για το συγκεκριμένο αντικείμενο.

**Η συνάρτηση program** αποτελεί την αρχή του μεταγλωτιστή. Παίρνει τον ορισμό και το όνομα του προγράμματος, καλεί την συνάρτηση block και περιμένει να κλείσει το πρόγραμμα. Επίσης εδώ δημιουργείται το βασικό επίπεδο του πίνακα συμβόλων.

**Η συνάρτηση block** ελέγχει την ορθή δομή του κύριου προγράμματος και των υπό-προγραμμάτων, δηλαδή ακολουθείται η δομή declarations, subprograms και blockstatements. Επίσης κάθε φορά που ολοκληρώνεται ένα block πριν αφαιρεθεί ένα επίπεδο από τον πίνακα συμβόλων, καταγράφεται στο αρχείο *cimplefile*.symb.txt οι οντότητες αυτού του επιπέδου και προστίθεται ο τελικός κώδικας στο αρχείο *cimplefile*.asm.

**Η συνάρτηση declarations** ελέγχει την συντακτική και την λεξική ανάλυση των declarations.

**Η συνάρτηση varlist** ελέγχει την συντακτική και την λεκτική ανάλυση των μεταβλητών που ορίζονται στα declarations.

**Η συνάρτηση subprograms** ενεργοποιείται μόλις λάβουμε token που προσδιορίζει ότι έχουμε function ή procedure.

**Η συνάρτηση subprograms** καλείται από την subprograms. Διαβάζει το όνομα της function ή της procedure και καλεί την formalparlist για να ελέγξει τις παραμέτρους .Στην συνέχεια καλεί την block για να γίνει ανάλυση των περιεχομένων της function ή της procedure και στο τέλος ελέγχει αν κλείνει σωστά.

**Η συνάρτηση formalparlist** καλείται από την subprograms για να αναλύσει λεξικά και συντακτικά τις παραμέτρους. Καλεί την formalparitem για να αναλύσει μία μία κάθε παράμετρο.

**Η συνάρτηση formalparitem** αναλύει μία μία τις παραμέτρους όσο αναφορά την λεκτική και την συντακτικής τους ανάλυση.

**Η συνάρτηση blockstatements** καλείται από την μπλοκ μετά την declaration και την subprograms και με την σειρά της καλή την statement.

**Η συνάρτηση statements** καλείται από άλλα statements που περιέχουν κι άλλες εντολές (ifStat, elsepart, whileStat, forcaseStat, incaseStat, switchcaseStat).

**Η συνάρτηση statement** ανάλογα με το token statement που λαμβάνει ανακατευθύνει τη ροή της συντακτικής ανάλυσης στην αντίστοιχη συνάρτηση.

**Η συνάρτηση assignStat** καλείται από την statements στην περίπτωση που έχουμε assignment statement . Κρατάει σε μια μεταβλητή το όνομα του assignment statement για την δημιουργία του ενδιάμεσου κώδικα και αφού κάνει τους απαραίτητους συντακτικούς ελέγχους καλεί την expression.

**Η συνάρτηση ifStat** καλείται από την statements στην περίπτωση που έχουμε if statement. Αυτή με την σειρά της καλεί την condition και κρατάει τις απαραίτητες θέσεις για τα jump που θα πρέπει να ενσωματωθούν στον ενδιάμεσο κώδικα σε περίπτωση που η αποτίμησης της έκφρασης είναι True και σε περίπτωση που η αποτίμησης της έκφρασης είναι False. Τέλος καλεί την statements ώστε να διαβαστούν τα περιεχόμενα της if και καλεί την συνάρτηση elsepart.

**Η συνάρτηση elsepart** καλείται από την ifStat με σκοπό να διαβάσει else σε περίπτωση που αυτό υπάρχει και να προσθέσει ενδιάμεσο κώδικα για αυτό το σημείο.

**Η συνάρτηση whileStat** καλείται από την statements στην περίπτωση που έχουμε while statement. Στην συνάρτηση αυτή καλούνται κάποιες αναγκαίες συναρτήσεις που θα αναλυθούν παρακάτω για την δημιουργία του ενδιάμεσου κώδικα και καλείται και η συνάρτηση statements για να διαβαστούν και να αναλυθούν οι γραμμές κώδικα που περιέχονται στην while.

**Η συνάρτηση switchcaseStat** καλείται από την statements στην περίπτωση που έχουμε switch statement. Η συνάρτηση αυτή για κάθε case που υπάρχει καλεί διαδοχικά πρώτα την condition και στην συνέχεια την statements ώστε να αναλυθούν/ει οι συνθήκες/η που έχει κάθε case και στην συνέχεια το εσωτερικό κάθε case. Παράλληλα δημιουργείτε και το αντίστοιχο κομμάτι ενδιάμεσου κώδικα. Τέλος ελέγχεται η ύπαρξη του default και καλείται η statements για την ανάλυση των περιεχομένων του default.

**Η συνάρτηση forcaseStat** καλείται από την statements στην περίπτωση που έχουμε forcase statement. Η συνάρτηση αυτή για κάθε case που υπάρχει καλεί διαδοχικά πρώτα την condition και στην συνέχεια την statements ώστε να αναλυθούν/ει οι συνθήκες/η που έχει κάθε case και στην συνέχεια το εσωτερικό κάθε case. Παράλληλα δημιουργείτε και το αντίστοιχο κομμάτι ενδιάμεσου κώδικα. Τέλος ελέγχεται η ύπαρξη του default και καλείται η statements για την ανάλυση των περιεχομένων του default.

**Η συνάρτηση incaseStat** καλείται από την statements στην περίπτωση που έχουμε incase statement. Η συνάρτηση αυτή για κάθε case που υπάρχει καλεί διαδοχικά πρώτα την condition και στην συνέχεια την

statements ώστε να αναλυθούν/ει οι συνθήκες/η που έχει κάθε case και στην συνέχεια το εσωτερικό κάθε case. Επίσης δημιουργείται και ένα βοηθητικό flag και οι αντίστοιχες εντολές ενδιάμεσου κώδικα για να γίνει η κατάλληλη ροή του προγράμματος στην μέσα στην incase στην περίπτωση κάποιου “true condition”. Παράλληλα δημιουργείτε και το αντίστοιχο κομμάτι ενδιάμεσου κώδικα.

**Η συνάρτηση returnStat** καλείται από την statements στην περίπτωση που έχουμε return statement. Αυτή με την σειρά της καλεί την expression για να αναλυθεί το επιστρεφόμενο. Παράλληλα γίνεται συντακτική ανάλυση και συγγραφή του απαραίτητου ενδιάμεσου κώδικα.

**Η συνάρτηση callStat** καλείται από την statements στην περίπτωση που έχουμε call statement. Αυτή με την σειρά της καλεί την actualparlist για την ανάλυση των παραμέτρων που θα περαστούν στην κλήση της συνάρτησης. Παράλληλα γίνεται συντακτική ανάλυση και συγγραφή του απαραίτητου ενδιάμεσου κώδικα.

**Η συνάρτηση printStat** καλείται από την statements στην περίπτωση που έχουμε print statement. Αυτή με την σειρά της καλεί την expression για να αναλυθούν τα περιεχόμενα του print. Παράλληλα γίνεται συντακτική ανάλυση και συγγραφή του απαραίτητου ενδιάμεσου κώδικα.

**Η συνάρτηση inputStat** καλείται από την statements στην περίπτωση που έχουμε input statement. Αυτή με την σειρά της ελέγχει τα επόμενα token που επιστρέφει ο lex για να κάνει την συντακτική ανάλυση και παράλληλα γράφει τον απαραίτητο ενδιάμεσο κώδικα.

**Η συνάρτηση actualparlist** είναι μια συνάρτηση που καλείται για να διαχειρίζεται τις παραμέτρους στις περιπτώσεις που έχουμε ορισμό ή κλήση συνάρτησης. Αυτή με την σειρά της καλεί την actualparitem. Παράλληλα κάνει και συντακτική για ότι token λαμβάνει από τον lex.

**Η συνάρτηση actualparitem** καλείται από την actualparlist για να διαχειριστεί τις εκφράσεις που μπήκαν ως παράμετροι. Με την σειρά της καλεί την expression. Παράλληλα γίνεται συντακτική ανάλυση και συγγραφή του απαραίτητου ενδιάμεσου κώδικα.

**Η συνάρτηση condition** καλείται για να διαχειριστεί όλες τις λογικές εκφράσεις που υπάρχουν σε μια λογική έκφραση. Καλεί την boolterm για

κάθε λογική έκφραση που υπάρχει. Επιπλέον διαχειρίζεται τις περιπτώσεις ύπαρξης or. Παράλληλα γίνεται συντακτική ανάλυση και συγγραφή του απαραίτητου ενδιάμεσου κώδικα.

**Η συνάρτηση boolterm** καλείται από την condition για την ανάλυση των εσωτερικών λογικών εκφράσεων. Επιπλέον διαχειρίζεται τις περιπτώσεις ύπαρξης and. Με την σειρά της καλεί την boolfactor.

**Η συνάρτηση boolfactor** καλείται από την boolterm για να ελέγξει περιπτώσεις που έχουμε not ή νέα ύπο έκφραση όπου σε αυτές τις περιπτωσεις αυτή καλεί την condition ή αν έχουμε απλή έκφραση όπου στην περίπτωση αυτή καλεί την expression. Παράλληλα γίνεται συντακτική ανάλυση και συγγραφή του απαραίτητου ενδιάμεσου κώδικα.

**Η συνάρτηση expression** καλείται στις περιπτώσεις που έχουμε αριθμητική έκφραση. Αυτή αρχικά καλεί την optionalSign και στην συνέχεια ενώνει το αποτέλεσμα που θα λάβει με αυτό από την κλήση της term. Έπειτα για κάθε ύπο αριθμητική έκφραση(ύπαρξη ‘+’ ή ‘-‘) καλεί την term. Παράλληλα γίνεται συντακτική ανάλυση και συγγραφή του απαραίτητου ενδιάμεσου κώδικα.

**Η συνάρτηση term** καλεί την συνάρτηση factor και για κάθε ύπαρξη συμβόλου τύπου ‘mult’ καλεί την factor. Παράλληλα γίνεται συντακτική ανάλυση και συγγραφή του απαραίτητου ενδιάμεσου κώδικα.

**Η συνάρτηση factor** ελέγχει την ύπαρξη τριών περιπτώσεων. Πρώτων το επόμενο στοιχείο που θα λάβει από τον lex να είναι αριθμός. Δεύτερον το επόμενο στοιχείο που θα λάβει από τον lex να είναι παρένθεση, το οποίο σημαίνει ότι έχουμε να κάνουμε με υπό έκφραση οπότε καλεί την expression. Τρίτον το επόμενο στοιχείο που θα λάβει από τον lex να είναι μία άλλη μεταβλητή όπου στην περίπτωση αυτή καλεί την idtail. Παράλληλα γίνεται συντακτική ανάλυση και συγγραφή του απαραίτητου ενδιάμεσου κώδικα.

**Η συνάρτηση idtail** ελέγχει αν καλέστηκε για να αναγνωρίσει μία μεταβλητή και να επιστρέψει αυτό το γεγονός ή την κλήση μίας συνάρτησης όπου παράγει τον απαραίτητο ενδιάμεσο κώδικα.

**Η συνάρτηση optionalSign** καλείται στις περιπτώσεις που ένας αριθμός ή αριθμητική έκφραση έχει πρόσημο μπροστά . Παράλληλα γίνεται συντακτική ανάλυση και συγγραφή του απαραίτητου ενδιάμεσου κώδικα.

**Οι ακόλουθες συναρτήσεις χρησιμοποιήθηκαν για την συγγραφή του ενδιάμεσου κώδικα:**

**Η συνάρτηση nextquad** καλείται για να επιστρέψει τον αριθμό της επόμενης τετράδας που πρόκειται να παραχθεί.

**Η συνάρτηση genquad** καλείται για να δημιουργήσει και να επιστρέψει μία λίστα που θα περιέχει τον αριθμό γραμμής και μια λίστα με τα στοιχεία της τετράδας.

**Η συνάρτηση newtemp** καλείται για να δημιουργήσει και να επιστρέψει μία νέα μεταβλητή.

**Η συνάρτηση emptylist** καλείται για να δημιουργήσει και να επιστρέψει μία άδεια λίστα.

**Η συνάρτηση makelist** καλείται για να δημιουργήσει και να επιστρέψει μία λίστα που περιέχει μόνο το στοιχείο που πήρε σαν είσοδο η συνάρτηση.

**Η συνάρτηση merge** καλείται για να επιστρέψει την ένωση των δύο λιστών που πήρε σαν είσοδο.

**Η συνάρτηση backpatch** παίρνει σαν είσοδο ένα label μιας τετράδας (τύπου if ή jump) και ένα label και αναθέτει σε αυτή την τετράδα αυτό το label.

**Η συνάρτηση productIntFile** υπάρχει για την συγγραφή του ενδιάμεσου κώδικα σε ένα νέο αρχείο μορφής ".int".

**Οι ακόλουθες συναρτήσεις προστέθηκαν για την συγγραφή του σημασιολογικού κώδικα και του πίνακα συμβόλων:**

**Η συνάρτηση addScope** καλείται για να προσθέσει ένα ακόμα επίπεδο στον πίνακα συμβόλων

**Η συνάρτηση addOffset** διαπερνά το τελευταίο offset του συμβολικού πίνακα και για κάθε οντότητα που δεν είναι υποπρόγραμμα προσθέτει 4 στο offset υπολογίζοντας το offset της κάθε μεταβλητής

**Η συνάρτηση removeScope** καλείται για να διαγράψει το τελευταίο scope από τον πίνακα συμβόλων.

**Η συνάρτηση addEntity** καλείται για προσθέσει μία νέα οντότητα στο συμβολικό πίνακα στο τελευταίο scope.

**Η συνάρτηση addArgument** καλείται για να προσθέσει στο υποπρόγραμμα του προηγούμενου επιπέδου το είδος των ορισμάτων που παίρνει και κατ' επέκταση και το πλήθος των ορισμάτων.

**Η συνάρτηση searchEntity** καλείται για να βρει αν υπάρχει μία οντότητα και την επιστρέφει.

**Η συνάρτηση checkEntityExists** καλείται για να βρει αν μια οντότητα υπάρχει πριν χρησιμοποιηθεί.

**Η συνάρτηση checkArguments** καλείται για να δει αν τα ορίσματα της συνάρτησης που πάει να καλεστεί έχουν σωστό αριθμό και τύπο.

**Οι ακόλουθες συναρτήσεις προστέθηκαν για την συγγραφή του τελικού κώδικα:**

**Η συνάρτηση varInfo** είναι μία βοηθητική συνάρτηση που παίρνει το όνομα μίας οντότητας και επιστρέφει όλες τις πληροφορίες από τον συμβολικό πίνακα και το βάθος στο οποίο βρίσκεται

**Η συνάρτηση gnlvcode** είναι μία βοηθητική συνάρτηση που παράγει τον τελικό κώδικα στην περίπτωση που μία μεταβλητή που θέλουμε να φτάσουμε βρίσκεται σε ανώτερο scope από αυτό που την χρειαζόμαστε

**Η συνάρτηση loadvr** είναι μία βοηθητική συνάρτηση που φορτώνει μία μεταβλητή από το εγγράφημα δραστηριοποίησης

**Η συνάρτηση storerv** είναι μία βοηθητική συνάρτηση που αποθηκεύει μία μεταβλητή από το εγγράφημα δραστηριοποίησης

**Η συνάρτηση createAssembly** είναι η συνάρτηση που παράγει τον τελικό κώδικα. Διαβάζει μία-μία τις τετράδες από τον ενδιάμεσο κώδικα και τις μετατρέπει σε εντολές τελικού κώδικα.