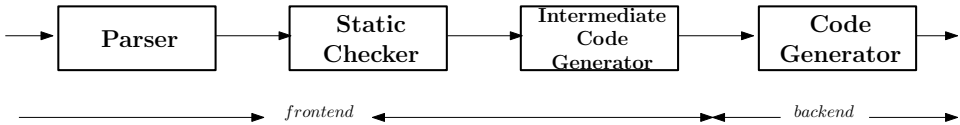
Μεταγλωττιστές

Παραγωγή Ενδιάμεσου Κώδικα

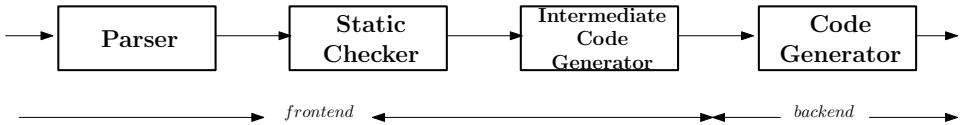
Δημήτρης Μιχαήλ

Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο

Παραγωγή Ενδιάμεσου Κώδικα



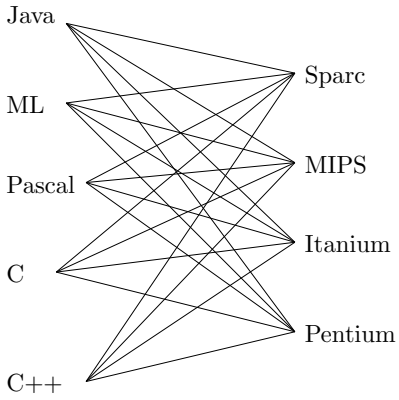
Παραγωγή Ενδιάμεσου Κώδικα

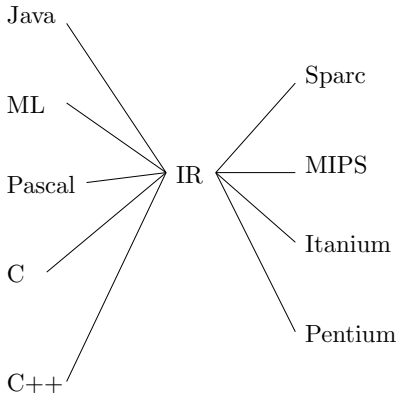


Η φάση της παραγωγής ενδιάμεσου κώδικα ουσιαστικά αποτελεί των συνδετικό κρίκο μεταξύ των φάσεων της ανάλυσης και της σύνθεσης του μεταγλωττιστή.

Ανάγκη Παραγωγής Ενδιάμεσου Κώδικα

Γιατί δεν παράγουμε απευθείας κώδικα μηχανής;

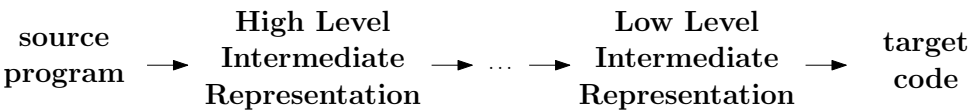




Εαν έχουμε N γλώσσες και M μηχανές, πρέπει να γράψουμε $N + M$ υλοποιήσεις αντί για $N \times M$ υλοποιήσεις.

Πολλές Ενδιάμεσες Μορφές

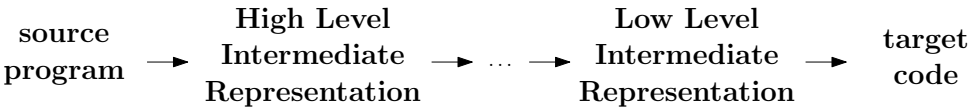
Συνήθως ένας μεταγλωττιστής παράγει πολλές ενδιάμεσες μορφές.



- οι υψηλού επιπέδου αναπαραστάσεις είναι πιο κοντά στην αρχική γλώσσα
- οι χαμηλού επιπέδου πιο κοντά στην γλώσσα μηχανής

Πολλές Ενδιάμεσες Μορφές

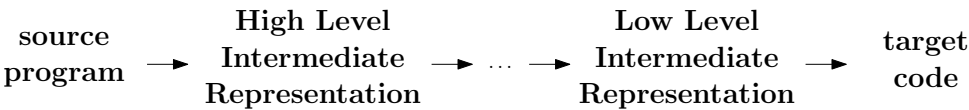
Συνήθως ένας μεταγλωττιστής παράγει πολλές ενδιάμεσες μορφές.



Τα συντακτικά δέντρα είναι υψηλού επιπέδου, αναπαριστούν την φυσική ιεραρχική δομή ενός προγράμματος και είναι πολύ βολικά για την πραγματοποίηση εργασιών όπως ο έλεγχος τύπων.

Πολλές Ενδιάμεσες Μορφές

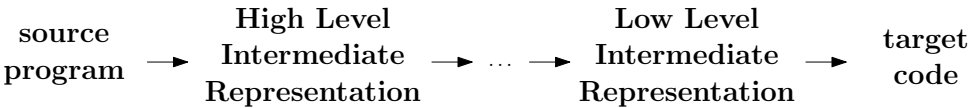
Συνήθως ένας μεταγλωττιστής παράγει πολλές ενδιάμεσες μορφές.



Μια χαμηλού επιπέδου αναπαράσταση είναι πιο βολική για εργασίες που σχετίζονται με την αρχιτεκτονική της μηχανής που μας ενδιαφέρει όπως η ανάθεση καταχωρητών και η επιλογή εντολών.

Πολλές Ενδιάμεσες Μορφές

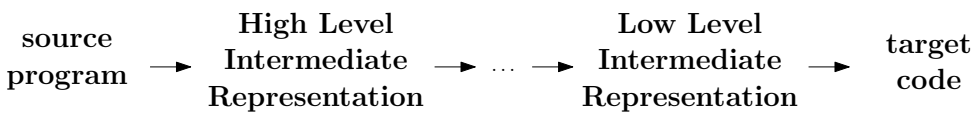
Συνήθως ένας μεταγλωττιστής παράγει πολλές ενδιάμεσες μορφές.



Η επιλογή των ενδιάμεσων μορφών διαφέρει από μεταγλωττιστή σε μεταγλωττιστή και εξαρτάται πολλές φορές από τα χαρακτηριστικά που θέλουμε να έχουμε.

Πολλές Ενδιάμεσες Μορφές

Συνήθως ένας μεταγλωττιστής παράγει πολλές ενδιάμεσες μορφές.



Η ενδιάμεση μορφή μπορεί να είναι ακόμη και μια άλλη γλώσσα.

Οι πρώτοι μεταγλωττιστές για την γλώσσα C++ είχαν ως ενδιάμεση μορφή γλώσσα C.

Ενδιάμεσες Μορφές

Υπάρχουν πολλών ειδών ενδιάμεσες μορφές.

1. αφηρημένα συντακτικά δέντρα (abstract syntax trees)
2. δέντρα ενδιάμεσου κώδικα (intermediate code trees)
3. κατευθυνόμενα ακυκλικά γραφήματα (directed acyclic graphs)
4. κώδικας 3-διευθύνσεων

Αφηρημένα Συντακτικά Δέντρα

Είδαμε ήδη πως ο συντακτικός αναλυτής μπορεί να κατασκευάσει ένα αφηρημένο συντακτικό δέντρο.

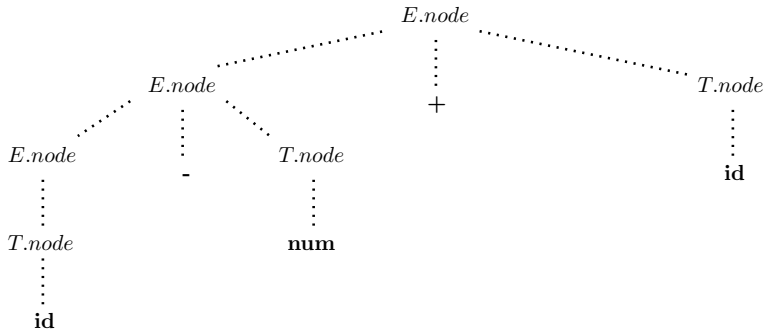
π.χ

κανόνας	σημασιολογική ρουτίνα
$E \to E_1 + T$	$E.node = new\ Node('+', E_1.node, T.node)$
$E \to E_1 - T$	$E.node = new\ Node('-', E_1.node, T.node)$
$E \to T$	$E.node = T.node$
$T \to (E)$	$T.node = E.node$
$T \to id$	$T.node = new\ Leaf(id, id.name)$
$T \to num$	$T.node = new\ Leaf(num, num.yylval)$

Αυτή η ενδιάμεση μορφή είναι πολύ κοντά στην αρχική γλώσσα και δεν βοηθάει άμεσα για την παραγωγή κώδικα.

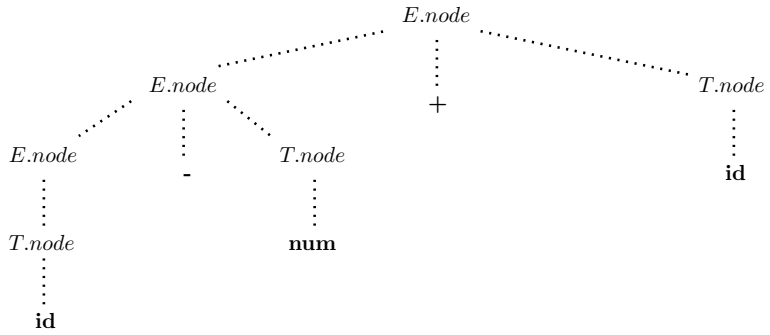
Αφηρημένα Συντακτικά Δέντρα

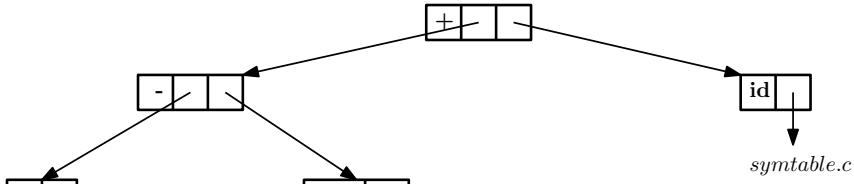
Η έκφραση $a - 4 + c$



Αφηρημένα Συντακτικά Δέντρα

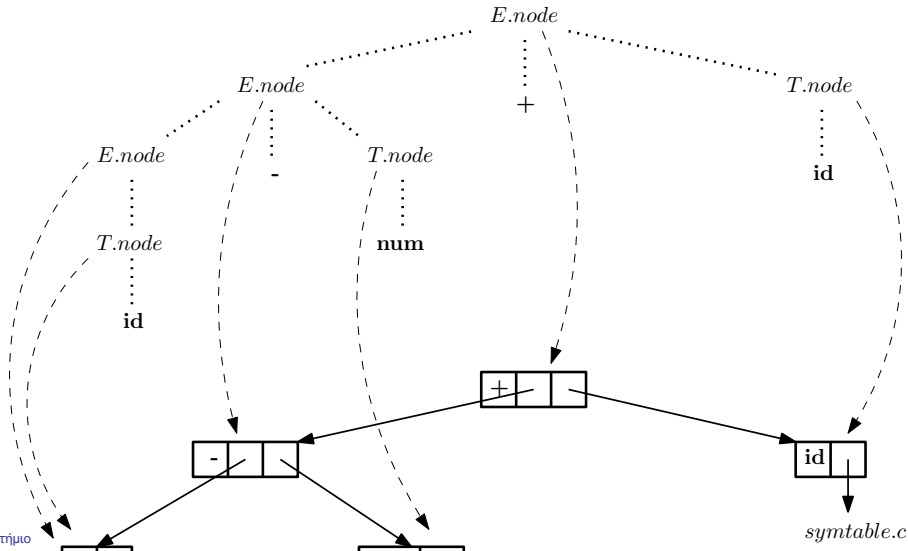
Η έκφραση $a - 4 + c$





Αφηρημένα Συντακτικά Δέντρα

Η έκφραση $a - 4 + c$



Κατευθυνόμενα Ακυκλικά Γραφήματα

Η αναπαράσταση με κατευθυνόμενα ακυκλικά γραφήματα ακολουθεί την ίδια λογική με τα αφηρημένα συντακτικά δέντρα. Η μόνη διαφορά είναι πως ένας κόμβος μπορεί να έχει πάνω από έναν γονέα.

Αυτή η αναπαράσταση είναι πιο οικονομική από τα αφηρημένα συντακτικά δέντρα και ταυτόχρονα περιέχει σημαντική πληροφορία που μπορεί να χρησιμοποιήσει ο μεταγλωττιστής για να παράγει καλύτερο κώδικα.

Κατευθυνόμενα Ακυκλικά Γραφήματα

Η έκφραση

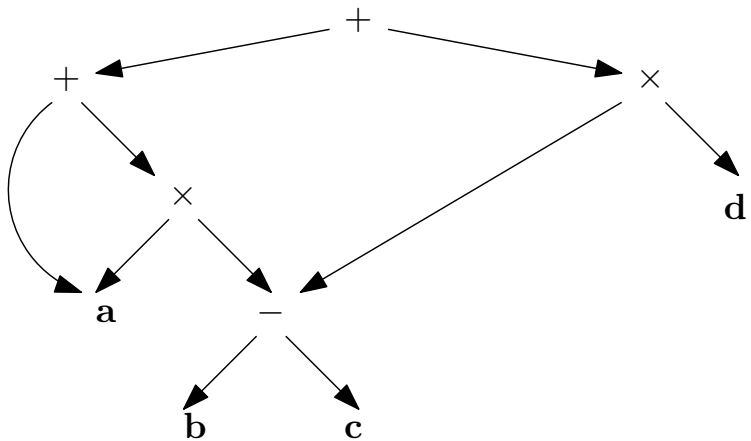
$$a + a \times (b - c) + (b - c) \times d$$

Κατευθυνόμενα Ακυκλικά Γραφήματα

Η έκφραση

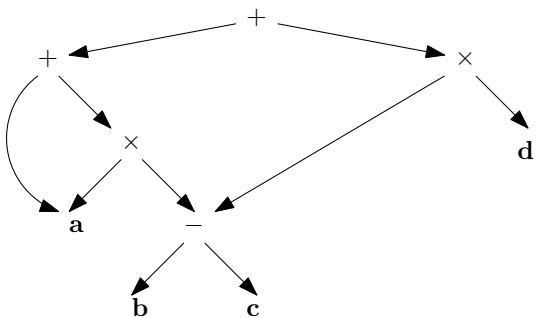
$$a + a \times (b - c) + (b - c) \times d$$

μεταφράζεται σε



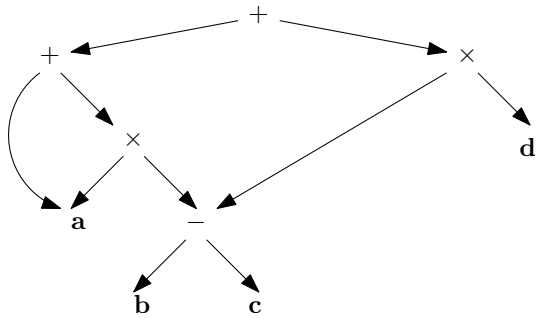
Κατευθυνόμενα Ακυκλικά Γραφήματα

$$a + a \times (b - c) + (b - c) \times d$$



- Και οι δύο εμφανίσεις της έκφρασης $b - c$ αναπαρίστανται με έναν κόμβο.
- Παρόλο που τα b και c εμφανίζονται δύο φορές, υπάρχουν μόνο μια φορά μέσα στην αναπαράσταση.

Κατευθυνόμενα Ακυκλικά Γραφήματα

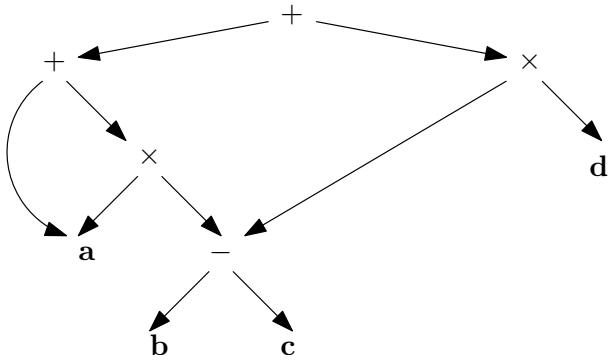


Τα μεταφραστικά σχήματα που παράγουν αφηρημένα συντακτικά δέντρα μπορούν με μικρές αλλαγές να παράξουν και κατευθυνόμενα ακυκλικά γραφήματα.

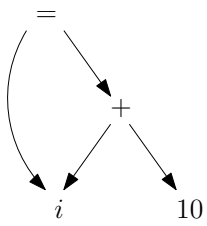
Πριν δημιουργήσουμε ένα καινούριο κόμβο, κοιτάμε μήπως υπάρχει ήδη.

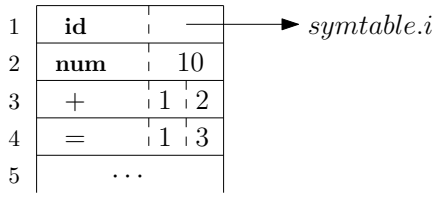
Κατευθυνόμενα Ακυκλικά Γραφήματα

$p_1 = \mathit{Leaf}(\mathit{id}, \mathit{entry} - a)$
$p_2 = \mathit{Leaf}(\mathit{id}, \mathit{entry} - a) = p_1$
$p_3 = \mathit{Leaf}(\mathit{id}, \mathit{entry} - b)$
$p_4 = \mathit{Leaf}(\mathit{id}, \mathit{entry} - c)$
$p_5 = \mathit{Node}('-', p_3, p_4)$
$p_6 = \mathit{Node}('\times', p_1, p_5)$
$p_7 = \mathit{Node}('+', p_1, p_6)$
$p_8 = \mathit{Leaf}(\mathit{id}, \mathit{entry} - b) = p_3$
$p_9 = \mathit{Leaf}(\mathit{id}, \mathit{entry} - c) = p_4$
$p_{10} = \mathit{Node}('-', p_3, p_4) = p_5$
$p_{11} = \mathit{Leaf}(\mathit{id}, \mathit{entry} - d)$
$p_{12} = \mathit{Node}('\times', p_5, p_{11})$
$p_{13} = \mathit{Node}('+', p_7, p_{12})$



Πολλές φορές οι κόμβοι ενός DAG αποθηκεύονται σε έναν πίνακα με εγγραφές.





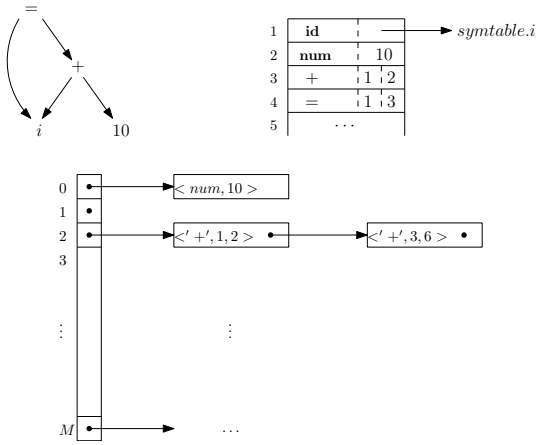
Πριν κατασκευάσουμε έναν κόμβο για $< op, l, r >$ κοιτάμε άμα υπάρχει ήδη στον πίνακα και σε ποια θέση. Άμα δεν υπάρχει τον προσθέτουμε ως τελευταίο στον πίνακα.

Για μεγαλύτερη ταχύτητα χρησιμοποιούμε έναν πίνακα κατακερματισμού.

Κατευθυνόμενα Ακυκλικά Γραφήματα

Η μέθοδος Τιμής-Αριθμού (Value-Number Method)

Για μεγαλύτερη ταχύτητα χρησιμοποιούμε έναν πίνακα κατακερματισμού για να αποθηκεύσουμε τις εγγραφές.



Χρησιμοποιούμε π.χ κατακερματισμό με αλυσίδες (hashing with chaining) και υπολογίζοντας μια συνάρτηση $h(op, l, r)$ βρίσκουμε σε ποια λίστα να αποθηκεύσουμε τον κόμβο.

Άσκηση

Φτιάξτε ένα κατευθυνόμενο ακυκλικό γράφημα για την έκφραση

$$((x+y)-((x+y)*(x-y)))+((x+y)*(x-y))$$

Δέντρα Ενδιάμεσου Κώδικα

Παρόμοια με τα αφηρημένα συντακτικά δέντρα είναι και τα ”δέντρα ενδιάμεσου κώδικα”.

Μεταφράζουμε ένα αφηρημένο συντακτικό δέντρο σε ένα δέντρο χαμηλότερου επιπέδου. Τα δέντρα αυτά περιέχουν κόμβους που βρίσκονται πιο κοντά στην μηχανή.

Δέντρα Ενδιάμεσου Κώδικα

Παρόμοια με τα αφηρημένα συντακτικά δέντρα είναι και τα ”δέντρα ενδιάμεσου κώδικα”.

Χρησιμοποιούμε κόμβους που περιγράφουν κλασικές λειτουργίες ενός υπολογιστή όπως

1. MEM(e) τα περιεχόμενα την μνήμης στην διεύθυνση *e*
2. CALL(f,l) κλήση της συνάρτηση *f* με λίστα παραμέτρων *l*
3. MOVE(TEMP t, e) υπολογισμός την έκφρασης *e* και μετακίνηση στην προσωρινή μεταβλητή *t*
4. JUMP(e) μετακίνηση ελέγχου στην διεύθυνση *e*
5. SEQ(s1,s2) παράθεση δύο προτάσεων $s1$ και $s2$
6. κ.τ.λ

Δεν θα μελετήσουμε περαιτέρω αυτή την αναπαράσταση.

Κώδικας Τριών Διευθύνσεων

three-address code

Στο κώδικα τριών διευθύνσεων επιτρέπεται η χρήση μόνο ενός τελεστή στο δεξιό μέρος μιας εντολής.

Μια έκφραση όπως

$$x + y * z$$

μεταφράζεται σε μια σειρά από εντολές

$$t_1 = y * z$$
$$t_2 = x + t_1$$

όπου t_1 και t_2 είναι προσωρινά ονόματα που δημιουργούνται από τον μεταγλωττιστή.

Κώδικας Τριών Διευθύνσεων

three-address code

Το ξεδίπλωμα τον πολύπλοκων εκφράσεων και των εντολών μέσω κώδικα τριών διευθύνσεων βοηθάει

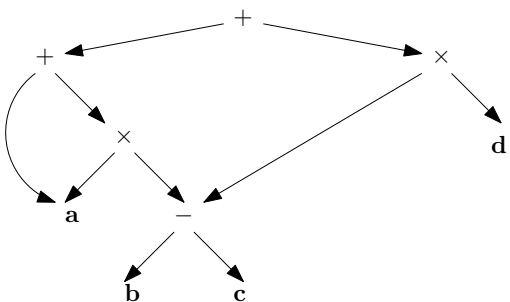
- στην παραγωγή τελικού κώδικα και
- στην βελτιστοποίηση του τελικού κώδικα.

Η χρήση των ονομάτων για τα ενδιάμεσα αποτελέσματα επιτρέπει την εύκολη εναλλαγή της σειράς των εντολών.

Κώδικας Τριών Διευθύνσεων

three-address code

Ο κώδικας τριών διευθύνσεων είναι μια γραμμική αναπαράσταση ενός συντακτικού δέντρου ή κατευθυνόμενου ακυκλικού γραφήματος.



$$t_1 = b - c$$
$$t_2 = a * t_1$$
$$t_3 = a + t_2$$
$$t_4 = t_1 * d$$
$$t_5 = t_3 + t_4$$

Κώδικας Τριών Διευθύνσεων

three-address code

Ο κώδικας τριών διευθύνσεων βασίζεται επάνω σε δύο βασικές έννοιες:

1. διευθύνσεις
2. εντολές

Σε αντικειμενοστραφής όρους θα λέγαμε πως αυτές οι δύο έννοιες αντιστοιχούν σε κλάσεις και οι συγκεκριμένου τύπου διευθύνσεις ή εντολές αντιστοιχούν σε υποκλάσεις.

Διευθύνσεις

- **ένα όνομα**
  Για ευκολία επιτρέπουμε σε ονόματα του πηγαίου κώδικα να εμφανιστούν ως διευθύνσεις στον κώδικα 3-διευθύνσεων.

  Σε μια υλοποίηση κρατάμε ένα δείκτη στην αντίστοιχη εγγραφή του πίνακα συμβόλων που αφορά το όνομα.

- **μία σταθερά**

- **προσωρινά ονόματα (temporaries)**
  Ο μεταγλωττιστής παράγει προσωρινά ονόματα, κάθε φορά που χρειάζεται να κρατήσει κάποιο ενδιάμεσο αποτέλεσμα. Τα ονόματα αυτά αποθηκεύονται κανονικά στον πίνακα συμβόλων.

  Στην παραγωγή τελικού κώδικα τα προσωρινά ονόματα υλοποιούνται με την χρήση των καταχωρητών.

Εντολές

Ένα πρόγραμμα σε κώδικα 3-διευθύνσεων περιέχει διάφορες εντολές που θα δούμε παρακάτω. Εντολές που αλλάζουν τον έλεγχο του προγράμματος χρησιμοποιούν και ετικέτες που υποδηλώνουν την θέση μιας εντολής στο πρόγραμμα.

1. **εντολή ανάθεσης** της μορφής

$$x = y \; op \; z$$

όπου *op* είναι ένας δυαδικός αριθμητικός ή λογικός τελεστής και τα *x*, *y* και *z* είναι διευθύνσεις.

π.χ

$$t_1 = b - c$$
$$t_r = 3 * d$$

2. **εντολή ανάθεσης** της μορφής

$$x = op\ y$$

όπου *op* είναι ένας μοναδιαίος τελεστής. Συνήθης τελεστές είναι το μείον, η λογική άρνηση και η μετατροπή τύπων, π.χ η μετατροπή ενός ακεραίου σε αριθμό κινητής υποδιαστολής.

π.χ

$$t_1 = -b$$
$$t_2 = !d$$
$$a = (double)f$$

Εντολές

3. **εντολή αντιγραφής** της μορφής

$$x = y$$

π.χ

$$t_1 = a$$
$$t_2 = 10$$

4. **εντολή διακλάδωσης χωρίς συνθήκη** της μορφής

$$goto\ L$$

όπου L είναι μια ετικέτα. Η εντολή 3-διευθύνσεων με ετικέτα L θα εκτελεστεί στην συνέχεια.

π.χ

```
L: a = a + 1
   b = a
   goto L
```

Εντολές

5 **εντολή διακλάδωσης** της μορφής

if x goto L

ή

ifFalse x goto L

που ανάλογα με την τιμή του *x* εκτελούν την εντολή με ετικέτα *L* ή αφήνουν την ροή του προγράμματος να συνεχιστεί κανονικά.

π.χ

```
   a = 0
L: a = a + 1
   e = a > 99
   ifFalse e goto L
```

Εντολές

6 **εντολή διακλάδωσης** της μορφής

if x relop y goto L

που χρησιμοποιούν έναν σχεσιακό τελεστή (<,==,>=, κ.τ.λ.) στα *x* και *y* και ανάλογα με την τιμή εκτελούν την εντολή με ετικέτα *L* ή αφήνουν την ροή του προγράμματος να συνεχιστεί κανονικά.

π.χ

```
   a = 0
   b = 99
L: a = a + 1
   if a > b goto L
```

Εντολές

7 **εντολές για την υλοποίηση συναρτήσεων** της μορφής

param x	για παραμέτρους
call p, n	κλήση μεθόδου *p* με *n* παραμέτρους
y = *call p, n*	κλήση συνάρτησης *p* με *n* παραμέτρους και ανάθεση του αποτελέσματος στην *y*
return y	επιστροφή τιμής

π.χ

```
param x1
param x2
param x3
t1 = call add, 3
```

Εντολές

8 **εντολές αντιγραφής με διευθυνσιοδότηση** της μορφής

$$x = y[i]$$
$$x[i] = y$$

η εντολή $x = y[i]$ θέτει στο x την τιμή που βρίσκεται στην μνήμη i θέσεις μετά την διεύθυνση του y

η εντολή $x[i] = y$ θέτει τα περιεχόμενα της μνήμης i θέσεις μετά το x στην τιμή y

Εντολές

9 **αναθέσεις διευθύνσεων και δεικτών** της μορφής

$$x = \&y$$
$$x = *y$$
$$*x = y$$

Η εντολή $x = \&y$ θέτει ως r-value στο x την θέση (l-value) του y.
Πιθανώς το y είναι ένα όνομα (ακόμη και προσωρινό) που είναι l-value (π.χ A[i][j]) και το x είναι ένα όνομα δείκτη ή ένα προσωρινό όνομα.

Η εντολή $x = *y$ θέλει ως r-value του x τα περιεχόμενα που βρίσκονται στην θέση που δείχνει ένας δείκτης y ή ένα προσωρινό με r-value μια θέση.

Η $*x = y$ θέτει ως r-value του αντικειμένου που δείχνει ο x ως το r-value του y.

Παράδειγμα Κώδικα 3-Διευθύνσεων

Ο κώδικας

```
do {
    i = i + 1;
}
while (a[i] < v);
```

μεταφράζεται ανάλογα με τον τρόπο που αναπαριστούμε τις ετικέτες εντολών σε

```
L:  t1 = i + 1;
    i  = t1;
    t2 = i * 4;
    t3 = a [ t2 ];
    if t3 < v goto L
```

```
100: t1 = i + 1;
101: i  = t1;
102: t2 = i * 4;
103: t3 = a [ t2 ];
104: if t3 < v goto 100
```

αν υποθέσουμε πως τα αντικείμενα στον πίνακα χρειάζονται χώρο ίσο με 4 μονάδες.

- Η επιλογή των λειτουργιών που μπορούν να γίνουν άμεσα στην ενδιάμεση μορφή είναι σημαντικό θέμα στην σχεδίαση των μεταγλωττιστών.

Επιλογή Επιτρεπτών Λειτουργιών

Ενδιάμεση Μορφή

(i) Η επιλογή των λειτουργιών που μπορούν να γίνουν άμεσα στην ενδιάμεση μορφή είναι σημαντικό θέμα στην σχεδίαση των μεταγλωττιστών.

(ii) Σίγουρα πρέπει να μπορούν να υλοποιήσουν όλα τα χαρακτηριστικά της αρχικής γλώσσας.

Επιλογή Επιτρεπτών Λειτουργιών

Ενδιάμεση Μορφή

(i) Η επιλογή των λειτουργιών που μπορούν να γίνουν άμεσα στην ενδιάμεση μορφή είναι σημαντικό θέμα στην σχεδίαση των μεταγλωττιστών.

(ii) Σίγουρα πρέπει να μπορούν να υλοποιήσουν όλα τα χαρακτηριστικά της αρχικής γλώσσας.

(iii) Λειτουργίες που είναι πιο κοντά στις εντολές της μηχανής υλοποιούνται πιο εύκολα στην φάση της παραγωγής τελικού κώδικα.

Επιλογή Επιτρεπτών Λειτουργιών

Ενδιάμεση Μορφή

(i) Η επιλογή των λειτουργιών που μπορούν να γίνουν άμεσα στην ενδιάμεση μορφή είναι σημαντικό θέμα στην σχεδίαση των μεταγλωττιστών.

(ii) Σίγουρα πρέπει να μπορούν να υλοποιήσουν όλα τα χαρακτηριστικά της αρχικής γλώσσας.

(iii) Λειτουργίες που είναι πιο κοντά στις εντολές της μηχανής υλοποιούνται πιο εύκολα στην φάση της παραγωγής τελικού κώδικα.

(iv) Εαν όμως παράγουμε πολλές εντολές για την υλοποίηση μιας λειτουργίας που η μηχανή έχει άμεσα διαθέσιμη, οι επόμενες φάσης της βελτιστοποίησης και της παραγωγής τελικού κώδικα θα πρέπει να δουλέψουν σκληρά.

Υλοποίηση Κώδικα 3-Διευθύνσεων

Μπορούμε να υλοποιήσουμε κώδικα 3-Διευθύνσεων με πολλούς τρόπους όπως:

- Τετράδες (Quadruples)
- Τριάδες (Triples)
- Έμμεσες Τριάδες (Indirect triples)

Οι υλοποιήσεις αυτές αναπαριστούν τις εντολές ως αντικείμενα ή εγγραφές με τα απαραίτητα πεδία.

Τετράδες
Quadruples

Μια τετράδα (Quadruple ή απλά ”quad”) έχει τέσσερα πεδία

1. *op*
2. arg_1
3. arg_2
4. *result*

Το πεδίο *op* περιέχει έναν εσωτερικό κωδικό για την λειτουργία.

Τετράδες
Quadruples

Μια τετράδα (Quadruple ή απλά ”quad”) έχει τέσσερα πεδία

1. *op*
2. arg_1
3. arg_2
4. *result*

Το πεδίο *op* περιέχει έναν εσωτερικό κωδικό για την λειτουργία.

Για να εκφράσουμε την λειτουργία

$$x = y + z$$

χρησιμοποιούμε τον κωδικό ’+’ στο πεδίο *op*, *y* στο πεδίο arg_1, *z* στο πεδίο arg_2 και *x* στο *result*.

```
+, y, z, x
```

Τετράδες
Quadruples

Εντολές που έχουν μόνο ένα όρισμα, δεν χρησιμοποιούν το arg_2.

(i) Η εντολή $x = y$ γράφεται ως

```
=, y, -, x
```

Τετράδες
Quadruples

Εντολές που έχουν μόνο ένα όρισμα, δεν χρησιμοποιούν το arg_2.

(i) Η εντολή $x = y$ γράφεται ως

```
=, y, -, x
```

(ii) Ενώ η $x =$ *minus* y γράφεται ως

```
minus, y, -, x
```

Τετράδες
Quadruples

Εντολές που έχουν μόνο ένα όρισμα, δεν χρησιμοποιούν το arg_2.

(i) Η εντολή $x = y$ γράφεται ως

```
=, y, -, x
```

(ii) Ενώ η $x =$ *minus* y γράφεται ως

```
minus, y, -, x
```

(iii) Η *param* x γράφεται ως

```
param, x, -, -
```

Τετράδες
Quadruples

Εντολές που έχουν μόνο ένα όρισμα, δεν χρησιμοποιούν το arg_2.

(i) Η εντολή $x = y$ γράφεται ως

```
=, y, -, x
```

(ii) Ενώ η $x =$ *minus* y γράφεται ως

```
minus, y, -, x
```

(iii) Η *param* x γράφεται ως

```
param, x, -, -
```

(iv) Η *goto* L γράφεται ως

```
goto, -, -, L
```

Τετράδες

Παράδειγμα

$$
\begin{aligned}
t_1 &= \mathit{minus}\ c \\
t_2 &= b * t_1 \\
t_3 &= \mathit{minus}\ c \\
t_4 &= b * t_3 \\
t_5 &= t_2 + t_4 \\
a &= t_5
\end{aligned}
$$

	op	*arg*1	*arg*2	*result*
0	*minus*	*c*		t_1
1	$*$	*b*	t_1	t_2
2	*minus*	*c*		t_3
3	$*$	*b*	t_3	t_4
4	$+$	t_2	t_4	t_5
5	$=$	t_5		*a*
		$\ldots$		

Για ευκολία χρησιμοποιούμε τα ονόματα των προσδιοριστικών (*a*, *b* ή *c*). Σε μια υλοποίηση θα είχαμε δείκτες προς τον πίνακα συμβόλων.

Τετράδες

Παράδειγμα

$$
\begin{aligned}
t_1 &= \mathit{minus}\ c \\
t_2 &= b * t_1 \\
t_3 &= \mathit{minus}\ c \\
t_4 &= b * t_3 \\
t_5 &= t_2 + t_4 \\
a &= t_5
\end{aligned}
$$

	op	*arg*1	*arg*2	*result*
0	*minus*	*c*		t_1
1	$*$	*b*	t_1	t_2
2	*minus*	*c*		t_3
3	$*$	*b*	t_3	t_4
4	$+$	t_2	t_4	t_5
5	$=$	t_5		*a*
		$\ldots$		

Για ευκολία χρησιμοποιούμε τα ονόματα των προσδιοριστικών (*a*, *b* ή *c*). Σε μια υλοποίηση θα είχαμε δείκτες προς τον πίνακα συμβόλων.

Αντίστοιχα τα προσωρινά ονόματα μπορούμε να τα εισάγουμε στον πίνακα συμβόλων, όπως θα γινόταν άμα ο προγραμματιστής τα είχε χρησιμοποιήσει. Μια άλλη δυνατότητα είναι να τα υλοποιήσουμε ως αντικείμενα μιας ειδικής κλάσης *Temp* που θα μας παρέχει τις απαραίτητες μεθόδους για την διαχείριση τους.

Τριάδες
Triples

Οι τριάδες έχουν μόνο τρία πεδία:

1. *op*
2. arg_1
3. arg_2

Όταν χρησιμοποιούμε τριάδες αναφερόμαστε το αποτέλεσμα μιας λειτουργίας *x op y* χρησιμοποιώντας την θέση της αντίστοιχης τριάδας.

Τριάδες

Παράδειγμα

	op	*arg*1	*arg*2
0	*minus*	*c*	
1	$*$	*b*	(0)
2	*minus*	*c*	
3	$*$	*b*	(2)
4	$+$	(1)	(3)
5	$=$	*a*	(4)
		...	

Τριάδες
Παράδειγμα

	op	*arg*1	*arg*2
0	*minus*	*c*	
1	$*$	*b*	(0)
2	*minus*	*c*	
3	$*$	*b*	(2)
4	$+$	(1)	(3)
5	$=$	*a*	(4)
		$\ldots$	

Αυτή η αναπαράσταση έχει μικρότερη κατανάλωση σε χώρο, αλλά μας δυσκολεύει στην μετακίνηση εντολών.

Ένας μεταγλωττιστής στην φάση της βελτιστοποίησης συχνά αλλάζει την σειρά των εντολών. Με αυτή την αναπαράσταση θα πρέπει να αλλάξουμε και τις αναφορές στις εντολές.

Έμμεσες Τριάδες

Το πρόγραμμα μας είναι μια λίστα από δείκτες στις πραγματικές τριάδες. Με αυτό τον τρόπο μπορούμε να αλλάξουμε την σειρά των εντολών χωρίς να πρέπει να αλλάξουμε τις εντολές.

	instruction
35	(0)
36	(1)
37	(2)
38	(3)
39	(4)
40	(5)
	...

	op	*arg*1	*arg*2
0	*minus*	*c*	
1	$*$	*b*	(0)
2	*minus*	*c*	
3	$*$	*b*	(2)
4	$+$	(1)	(3)
5	$=$	*a*	(4)
		...	

Οι έμμεσες τριάδες μας παρέχουν την ευκολία των τετράδων στην αντιμετάθεση εντολών και ταυτόχρονα καταναλώνουν λιγότερο χώρο.

Μετάφραση Εκφράσεων

Θα δούμε με λεπτομέρεια πως μπορούμε μέσω ”μετάφρασης οδηγούμενης από το συντακτικό” να μεταφράσουμε εκφράσεις σε ενδιάμεσο κώδικα 3-διευθύνσεων.

Σε περίπτωση που ο μεταγλωττιστής μας χρησιμοποιεί αφηρημένα συντακτικά δέντρα, η ίδια δουλειά θα μπορούσε να γίνει υλοποιώντας μια κατάλληλη διάσχιση του αφηρημένου συντακτικού δέντρου.

Μετάφραση Εκφράσεων

Θα χρησιμοποιήσουμε την διφορούμενη γραμματική για εκφράσεις μαζί με έναν επιπλέον κανόνα για τις αναθέσεις.

$$
\begin{array}{lcl}
S & \rightarrow & \textbf{id} = E\ ; \\
\\
E & \rightarrow & E + E \\
E & \rightarrow & - E \\
E & \rightarrow & (\ E\ ) \\
E & \rightarrow & \textbf{id}
\end{array}
$$

(i) Το μη-τερματικό S (από Statement) έχει μια σημασιολογική τιμή $S.code$ που αντιστοιχεί στον κώδικα για την πρόταση.

Μετάφραση Εκφράσεων

Θα χρησιμοποιήσουμε την διφορούμενη γραμματική για εκφράσεις μαζί με έναν επιπλέον κανόνα για τις αναθέσεις.

$$
\begin{array}{lcl}
S & \rightarrow & \textbf{id} = E\ ; \\
\\
E & \rightarrow & E + E \\
E & \rightarrow & - E \\
E & \rightarrow & (\ E\ ) \\
E & \rightarrow & \textbf{id}
\end{array}
$$

(i) Το μη-τερματικό S (από Statement) έχει μια σημασιολογική τιμή $S.code$ που αντιστοιχεί στον κώδικα για την πρόταση.

(ii) το μη-τερματικό E (από Expression) έχει σημασιολογική τιμή $E.code$ για τον κώδικα που αντιστοιχεί στην έκφραση και $E.addr$ για την διεύθυνση που θα κρατάει την τιμή της έκφρασης.

Μετάφραση Εκφράσεων

$E \rightarrow \textbf{id} \quad \{ \quad E.\text{addr} = \text{top.lookup(id.yylval)}; \quad E.\text{code} = \text{""} \quad \}$

(i) Όταν η έκφραση είναι ένα αναγνωριστικό, έστω x, τότε το x κρατάει την τιμή της έκφρασης.

Μετάφραση Εκφράσεων

$E \rightarrow$ **id** { E.addr = top.lookup(id.yylval);
E.code = "" }

(i) Όταν η έκφραση είναι ένα αναγνωριστικό, έστω x, τότε το x κρατάει την τιμή της έκφρασης.

(ii) Οι σημασιολογικές ρουτίνες για αυτή την παραγωγή θέτουν την τιμή E.addr να δείχνει στην εγγραφή στον πίνακα συμβόλων που αντιστοιχεί στο όνομα id.yylval.

Μετάφραση Εκφράσεων

$E \rightarrow$ **id** { E.addr = top.lookup(id.yylval);
E.code = "" }

(i) Όταν η έκφραση είναι ένα αναγνωριστικό, έστω x, τότε το x κρατάει την τιμή της έκφρασης.

(ii) Οι σημασιολογικές ρουτίνες για αυτή την παραγωγή θέτουν την τιμή E.addr να δείχνει στην εγγραφή στον πίνακα συμβόλων που αντιστοιχεί στο όνομα id.yylval.

(iii) Ο κανόνας αυτός δεν παράγει κώδικα.

Μετάφραση Εκφράσεων

$E \rightarrow$ **id** { E.addr = top.lookup(id.yylval);
E.code = "" }

(i) Όταν η έκφραση είναι ένα αναγνωριστικό, έστω x, τότε το x κρατάει την τιμή της έκφρασης.

(ii) Οι σημασιολογικές ρουτίνες για αυτή την παραγωγή θέτουν την τιμή E.addr να δείχνει στην εγγραφή στον πίνακα συμβόλων που αντιστοιχεί στο όνομα id.yylval.

(iii) Ο κανόνας αυτός δεν παράγει κώδικα.

(iv) Παρόλο που δεν φαίνεται εδώ, η top.lookup() πρέπει να πετάει μια εξαίρεση (exception) εαν δεν βρει στον πίνακα συμβόλων αντίστοιχη εγγραφή.

Μετάφραση Εκφράσεων

$E \rightarrow (\ E_1\ )$ { E.addr = E_1.addr;
E.code = E_1.code; }

(i) Ο κώδικας για την έκφραση δεν αλλάζει.

(ii) Το ίδιο συμβαίνει και με την διεύθυνση που βρίσκεται η τιμή του αποτελέσματος της έκφρασης.

Μετάφραση Εκφράσεων

$E \rightarrow$ - E_1 { E.addr = new Temp();
E.code = E_1.code || gen(E.addr '=' 'minus' E_1.addr); }

(i) Δημιουργούμε μια προσωρινή μεταβλητή.

(ii) Ο κώδικας είναι η παράθεση (||)
- του κώδικα για την έκφραση E_1
- μαζί με μια επιπλέον ανάθεση στην προσωρινή μεταβλητή μας

Μετάφραση Εκφράσεων

$E \rightarrow$ - E_1 { E.addr = new Temp();
E.code = E_1.code || gen(E.addr '=' 'minus' E_1.addr); }

(i) Δημιουργούμε μια προσωρινή μεταβλητή.
(ii) Ο κώδικας είναι η παράθεση (||)
- του κώδικα για την έκφραση E_1
- μαζί με μια επιπλέον ανάθεση στην προσωρινή μεταβλητή μας

π.χ εαν E_1.addr = y και E_1.code είναι

```
y = x + 4
```

τότε εμείς δημιουργούμε μια προσωρινή μεταβλητή, έστω $t1$, θέτουμε E.addr = t1 και αλλάζουμε τον κώδικα σε

```
y = x + 4
t1 = - y
```

Μετάφραση Εκφράσεων

$E \rightarrow E_1 + E_2$ { E.addr = new Temp();
E.code = E_1.code || E_2.code ||
gen(E.addr '=' E_1.addr '+' E_2.addr); }

(i) Δημιουργούμε μια προσωρινή μεταβλητή για το αποτέλεσμα.
(ii) Ο κώδικας είναι η παράθεση (||)
- του κώδικα για την έκφραση E_1
- του κώδικα για την έκφραση E_2
- μαζί με μια ανάθεση του αθροίσματος στην προσωρινή μεταβλητή

Μετάφραση Εκφράσεων

$E \rightarrow E_1 + E_2$ { E.addr = new Temp();
E.code = E_1.code || E_2.code ||
gen(E.addr '=' E_1.addr '+' E_2.addr); }

π.χ εαν E_1.addr = t1 και E_1.code είναι

```
t1 = x + 4
```

και E_2.addr = t2 και E_2.code είναι

```
y = x + 4
t2 = - y
```

τότε εμείς δημιουργούμε μια προσωρινή μεταβλητή, έστω t3, θέτουμε E.addr = t3 και αλλάζουμε τον κώδικα σε

```
t1 = x + 4
y = x + 4
t2 = - y
t3 = t1 + t2
```

Μετάφραση Εκφράσεων

$S \rightarrow$ **id =** E { S.code = E.code ||
gen(top.lookup(id.yylval) '=' E.addr); }

(i) Ο κώδικας είναι η παράθεση (||)
- του κώδικα για την έκφραση E
- μαζί με μια επιπλέον ανάθεση στο αναγνωριστικό του αποτελέσματος της έκφρασης

Μετάφραση Εκφράσεων

$S \rightarrow$ **id =** E { S.code = E.code ||
gen(top.lookup(id.yylval) '=' E.addr); }

π.χ εαν E.addr = t3 και E.code είναι

```
t1 = x + 4
y = x + 4
t2 = - y
t3 = t1 + t2
```

και έχουμε πως id.yylval = q θέτουμε ως S.code

```
t1 = x + 4
y = x + 4
t2 = - y
t3 = t1 + t2
q = t3
```

Μετάφραση Εκφράσεων

Οι σημασιολογική τιμή code που είχαμε στις εκφράσεις και στις προτάσεις μπορεί να γίνει πολύ μεγάλη σε μήκος.

Για αυτό το λόγο, δεν διατηρούμε αυτές τις τιμές, απλά φροντίζουμε η συνάρτηση $gen(\cdot)$ που παράγει τον κώδικα να προσθέτει την εντολή σε κάποια λίστα ή πίνακα εντολών.

Μετάφραση Εκφράσεων

Καταλήγουμε στο εξής μεταφραστικό σχήμα.

S	$\rightarrow$	**id =** E **;**	{	gen(top.lookup(id.yylval) '=' E.addr);	}
E	$\rightarrow$	E **+** E	{	E.addr = new Temp(); gen(E.addr '=' E_1.addr '+' E_2.addr);	}
E	$\rightarrow$	**-** E	{	E.addr = new Temp(); gen(E.addr '=' 'minus' E_1.addr);	}
E	$\rightarrow$	**(** E **)**	{	E.addr = E_1.addr	}
E	$\rightarrow$	**id**	{	E.addr = top.lookup(id.yylval);	}

Διευθυνσιοδότητη Πινάκων

Εαν αποθηκεύσουμε τα στοιχεία ενός πίνακα συνεχόμενα στην μνήμη, μπορούμε να έχουμε άμεση πρόσβαση σε αυτά.

- Γλώσσες όπως η C, C++ και η Java αριθμούν τα n στοιχεία ενός πίνακα από 0 έως και $n-1$.

Διευθυνσιοδότητη Πινάκων

Εαν αποθηκεύσουμε τα στοιχεία ενός πίνακα συνεχόμενα στην μνήμη, μπορούμε να έχουμε άμεση πρόσβαση σε αυτά.

(i) Γλώσσες όπως η C, C++ και η Java αριθμούν τα n στοιχεία ενός πίνακα από 0 έως και $n-1$.

(ii) Εαν το πλάτος κάθε στοιχείου ενός πίνακα A είναι w, τότε το i-οστό στοιχείο του πίνακα βρίσκεται στην θέση

$$base + i \times w$$

όπου $base$ είναι η διεύθυνση του πίνακα, δηλαδή η διεύθυνση που έχει δωθεί στο στοιχείο $A[0]$.

Διευθυνσιοδότητη Πινάκων

Σε περίπτωση που έχουμε έναν πίνακα A με 2 διαστάσεις όπου

- κάθε γραμμή έχει πλάτος w_1
- κάθε στοιχειο έχει πλάτος w_2

τότε η σχετική διεύθυνση του $A[i_1][i_2]$ είναι

$$base + i_1 \times w_1 + i_2 \times w_2.$$

Ο παραπάνω τύπος μπορεί εύκολα να γραφεί και για την περίπτωση των k διαστάσεων.

Διευθυνσιοδότητη Πινάκων

Μπορούμε εναλλακτικά να υπολογίσουμε την διεύθυνση συναρτήσει του αριθμού n_j των στοιχείων κατά μήκος της διάστασης j του πίνακα, και το πλάτος w ενός στοιχείου του πίνακα.

2 διαστάσεις

Για δύο διαστάσεις, η διεύθυνση του στοιχείου $A[i_1][i_2]$ είναι

$$base + (i_1 \times n_2 + i_2) \times w.$$

k διαστάσεις

Για k διαστάσεις, η διεύθυνση του στοιχείου $A[i_1][i_2]\cdots[i_k]$ είναι

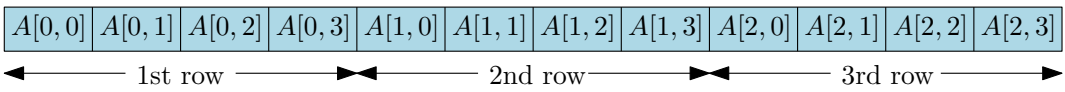
$$base + ((\cdots((i_1 \times n_2 + i_2) \times n_3 + i_3)\cdots) \times n_k + i_k) \times w.$$

Διευθυνσιοδότητη Πινάκων

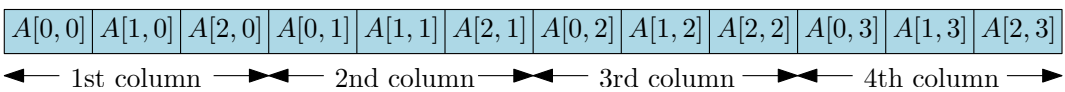
Ο τρόπος υπολογισμού διευθύνσεων που είδαμε ονομάζεται row-major μορφή και χρησιμοποιείται σε γλώσσες όπως η C, C++ και η Java.

$A[0,0]$	$A[0,1]$	$A[0,2]$	$A[0,3]$
$A[1,0]$	$A[1,1]$	$A[1,2]$	$A[1,3]$
$A[2,0]$	$A[2,1]$	$A[2,2]$	$A[2,3]$

row-major



column-major



Γλώσσες όπως η Fortran χρησιμοποιούν column-major μορφή.

Μετάφραση Αναφορών σε Πίνακες

Η δυσκολία που παρουσιάζει η μετάφραση εκφράσεων που περιέχουν αναφορές σε πίνακες είναι να μετατρέψουμε την συζήτηση για τον υπολογισμό των διευθύνσεων των προηγούμενων διαφανειών σε σημασιολογικές ρουτίνες.

(i) Μαζί με τους τύπους έχουμε υπολογίσει και το πλάτος τους.

(ii) Θα χρησιμοποιήσουμε τις εκφράσεις υπολογισμού διευθύνσεων που βασίζονται στους τύπους.

Μετάφραση Αναφορών σε Πίνακες

Καταρχήν θα προσθέσουμε ένα καινούριο μη-τερματικό L και απαραίτητους κανόνες για τους πίνακες.

$$
\begin{array}{lcl}
S & \rightarrow & \textbf{id} = E\ ; \\
S & \rightarrow & L = E\ ;
\end{array}
$$

$$
\begin{array}{lcl}
E & \rightarrow & E + E \\
E & \rightarrow & - E \\
E & \rightarrow & (\ E\ ) \\
E & \rightarrow & \textbf{id} \\
E & \rightarrow & L
\end{array}
$$

$$
\begin{array}{lcl}
L & \rightarrow & \textbf{id}\ [\ E\ ] \\
L & \rightarrow & L\ [\ E\ ]
\end{array}
$$

Μετάφραση Αναφορών σε Πίνακες

Το μη-τερματικό L έχει τα εξής χαρακτηριστικά

- L.addr είναι μια προσωρινή μεταβλητή που χρησιμοποιείται για να υπολογίσουμε την διεύθυνση της αναφοράς

Μετάφραση Αναφορών σε Πίνακες

Το μη-τερματικό L έχει τα εξής χαρακτηριστικά

(i) L.addr είναι μια προσωρινή μεταβλητή που χρησιμοποιείται για να υπολογίσουμε την διεύθυνση της αναφοράς

(ii) L.array είναι ένας δείκτης στην καταχώρηση για τον πίνακα στον πίνακα συμβόλων. Η L.array.base είναι η αρχική διεύθυνση του πίνακα.

Μετάφραση Αναφορών σε Πίνακες

Το μη-τερματικό *L* έχει τα εξής χαρακτηριστικά

(i) *L*.addr είναι μια προσωρινή μεταβλητή που χρησιμοποιείται για να υπολογίσουμε την διεύθυνση της αναφοράς

(ii) *L*.array είναι ένας δείκτης στην καταχώρηση για τον πίνακα στον πίνακα συμβόλων. Η *L*.array.base είναι η αρχική διεύθυνση του πίνακα.

(iii) *L*.type είναι ο τύπος του υποπίνακα που αντιστοιχεί στο *L*. Είδαμε μεταφραστικό σχήμα για τον υπολογισμό του τύπου και του πλάτους στο κεφάλαιο για την σημασιολογική ανάλυση.

Ο υπολογισμός των τύπων έχει ήδη γίνει όταν κάναμε συντακτική ανάλυση των δηλώσεων των μεταβλητών.

Θεωρούμε πως ένας τύπος *t* ξέρει το πλάτος του *t.width*. Επίσης ένας τύπος που είναι πίνακας ξέρει τον τύπο των στοιχείων του ως *t.elem*.

Μετάφραση Αναφορών σε Πίνακες

Το μη-τερματικό L έχει τα εξής χαρακτηριστικά

(i) L.addr είναι μια προσωρινή μεταβλητή που χρησιμοποιείται για να υπολογίσουμε την διεύθυνση της αναφοράς

(ii) L.array είναι ένας δείκτης στην καταχώρηση για τον πίνακα στον πίνακα συμβόλων. Η L.array.base είναι η αρχική διεύθυνση του πίνακα.

(iii) L.type είναι ο τύπος του υποπίνακα που αντιστοιχεί στο L. Είδαμε μεταφραστικό σχήμα για τον υπολογισμό του τύπου και του πλάτους στο κεφάλαιο για την σημασιολογική ανάλυση.

Ο υπολογισμός των τύπων έχει ήδη γίνει όταν κάναμε συντακτική ανάλυση των δηλώσεων των μεταβλητών.

Θεωρούμε πως ένας τύπος t ξέρει το πλάτος του $t.width$. Επίσης ένας τύπος που είναι πίνακας ξέρει τον τύπο των στοιχείων του ως $t.elem$.

Στην συνέχεια επεξηγούμε με λεπτομέρεια μόνο τα μέρη της γραμματικής εκφράσεων που αλλάζουν λόγω της παρουσίας των πινάκων.

Μετάφραση Αναφορών σε Πίνακες

$S \rightarrow L = E \quad \{ \quad \text{gen}(\ L.\text{array.base}\ \text{'['}\ L.\text{addr}\ \text{']'}\ \text{'='}\ E.\text{addr});\quad \}$

(i) Διαβάζουμε από τον πίνακα συμβόλων την διεύθυνση του πίνακα *L*.array.base

(ii) Έχουμε υπολογίσει την διεύθυνση του στοιχείου που μας ενδιαφέρει σε σχέση με την αρχή του πίνακα.

π.χ εαν *E*.array.base = a και *L*.addr = t1, *E*.addr = t2 καλούμε

$$gen(a[t1] = t2)$$

που παράγει την τετράδα

```
a[t1] = t2
```

Μετάφραση Αναφορών σε Πίνακες

$E \rightarrow L$ { *E*.addr = new Temp();
gen(*E*.addr '=' *L*.array.base '[' *L*.addr ']'); }

(i) Παράγουμε καινούρια προσωρινή μεταβλητή για την έκφραση
(ii) Διαβάζουμε από το *L* την διεύθυνση του πίνακα που μας ενδιαφέρει
(iii) Διαβάζουμε την ήδη υπολογισμένη θέση του στοιχείου που μας ενδιαφέρει σε σχέση με την αρχή του πίνακα.
(iv) Παράγουμε την εντολή ανάθεσης.

π.χ εαν *L*.array.base = a και *L*.addr = t4 θέτουμε ως *E*.addr = t5
και καλούμε

$$gen(t5 = a[t4]);$$

που παράγει την τετράδα

```
t5 = a[t4]
```

Μετάφραση Αναφορών σε Πίνακες

$L \rightarrow$ **id [** E **]** { L.array = top.lookup(id.yylval);
L.type = L.array.type.elem
L.addr = new Temp();
gen(L.addr '=' E.addr '*' L.type.width); }

(i) Βρίσκουμε το προσδιοριστικό στον πίνακα συμβόλων ή πετάμε μια εξαίρεση

(ii) Αναθέτουμε τον τύπο των στοιχείων του πίνακα στο L.type (ουσιαστικά αφαιρούμε μια διάσταση στον τύπο ώστε να μπορούν τα πλάτη να υπολογιστούν σωστά).

Εαν L.array.type = array(3, array(5, int)) τότε L.array.type.elem = array(5,int).

(iii) Δημιουργούμε προσωρινή διεύθυνση για το αποτέλεσμα

(iv) Υπολογίζουμε την σχετική διεύθυνση στην προσωρινή διεύθυνση.

Μετάφραση Αναφορών σε Πίνακες

$L \rightarrow$ **id [** E **]** { L.array = top.lookup(id.yylval);
L.type = L.array.type.elem
L.addr = new Temp();
gen(L.addr '=' E.addr '*' L.type.width); }

π.χ αν

- *id*.yylval = a
- ο πίνακας a έχει δηλωθεί ως πίνακας 3×4 ακεραίων, δηλαδή array(3, array(4,int))
- E.addr = t7

τότε έχουμε

- L.array = διεύθυνση καταχώρησης για τον a στον πίνακα συμβόλων
- L.type = array(4,int)
- L.addr = t8 (καινούρια προσωρινή διεύθυνση)
- καλούμε gen(t8 = t7 * 16) αφού L.type.width = 16 ως πίνακας τεσσάρων ακεραίων.

```
t8 = t7 * 16
```

Μετάφραση Αναφορών σε Πίνακες

$L \rightarrow L_1$ **[** E **]** { L.array = L_1.array
L.type = L_1.type.elem
t = new Temp();
L.addr = new Temp();
gen(t '=' E.addr '*' L.type.width); }
gen(L.addr '=' L_1.addr '+' t); }

(i) Στέλνουμε την εγγραφή του πίνακα συμβόλων προς τα πάνω στο δέντρο
(ii) Αφαιρούμε ένα array από τον τύπο
(iii) Υπολογίζουμε την καινούρια σχετική διεύθυνση για το στοιχείο

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα

Έστω το παρακάτω πρόγραμμα

```
1  int main() {
2      int a[4][3][5];
3
4      a[2][2][4] = 1;
5  }
```

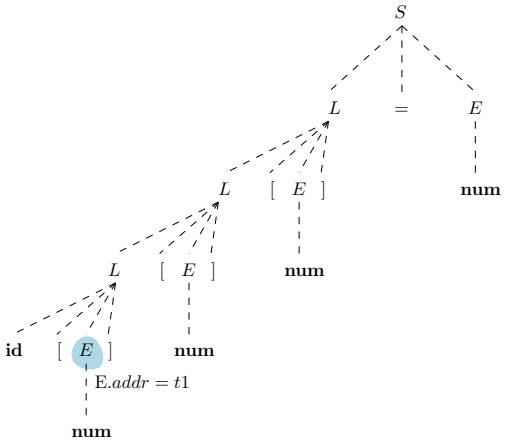
Θα δούμε με λεπτομέρεια πως μεταφράζεται η γραμμή 4 σε ενδιάμεσο κώδικα 3-διευθύνσεων από την γραμματική και το μεταφραστικό σχήμα που παρουσιάσαμε.

Αφού ο συντακτικός αναλυτής διαβάσει την γραμμή 2, ο πίνακας συμβόλων περιέχει μια εγγραφή με το όνομα *a* και τύπο ίσο με

- array(4, array(3, array(5, int)))

Μετάφραση Αναφορών σε Πίνακες

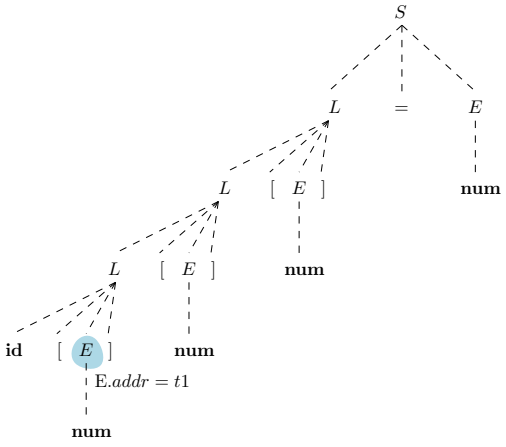
Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$



1. *E*.addr = new Temp() = t1

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$

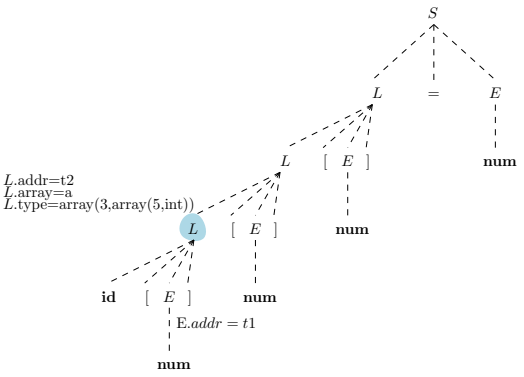



```
t1 = 2
```

1. *E*.addr = new Temp() = t1
2. gen(t1 = num.yylval)

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$

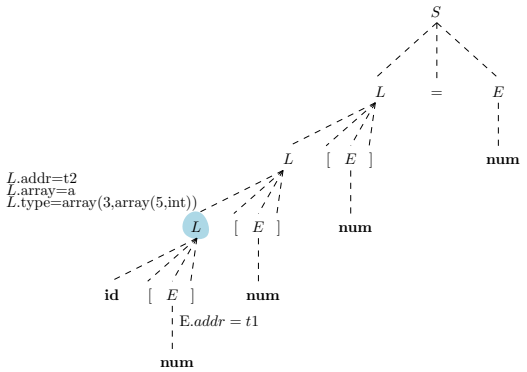



```
t1 = 2
```

1. *L*.array = top.get(id.yylval) = symbol table entry for a

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$

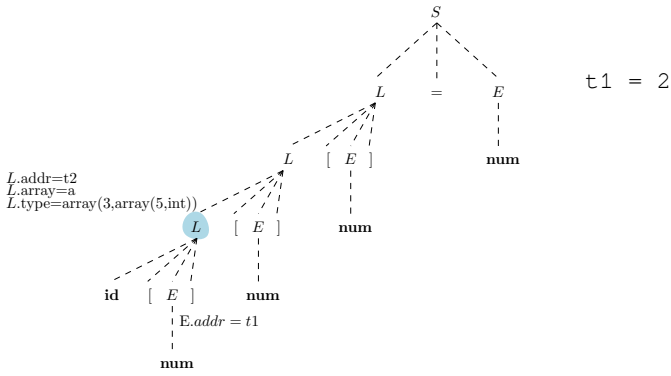



```
t1 = 2
```

1. *L*.array = top.get(id.yylval) = symbol table entry for a
2. *L*.type = L.array.type.elem = array(3, array(5,int))

Μετάφραση Αναφορών σε Πίνακες

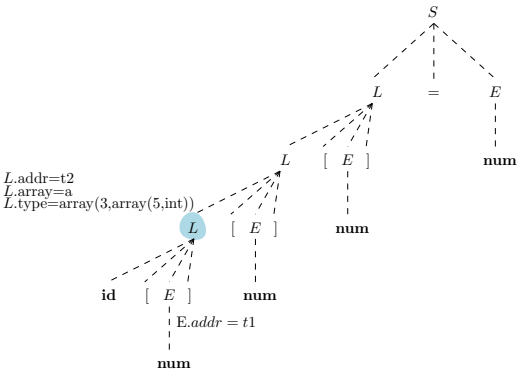
Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$



1. *L*.array = top.get(id.yylval) = symbol table entry for a
2. *L*.type = L.array.type.elem = array(3, array(5,int))
3. *L*.addr = new Temp() = t2

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$

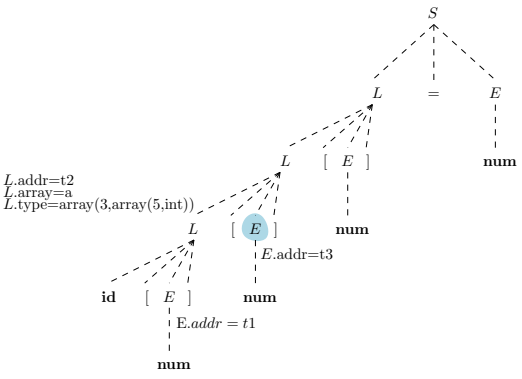



```
t1 = 2
t2 = t1 * 60
```

1. *L*.array = top.get(id.yylval) = symbol table entry for a
2. *L*.type = L.array.type.elem = array(3, array(5,int))
3. *L*.addr = new Temp() = t2
4. gen(t2 = t1 * *L*.type.width)

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$

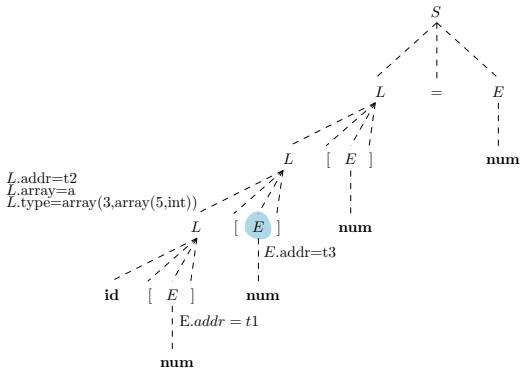



```
t1 = 2
t2 = t1 * 60
```

1. *E*.addr = new Temp() = t3;

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$

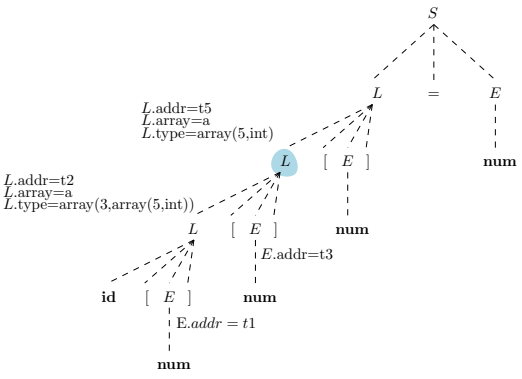



```
t1 = 2
t2 = t1 * 60
t3 = 2
```

1. *E*.addr = new Temp() = t3;
2. gen(t3 = num.yylval)

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$

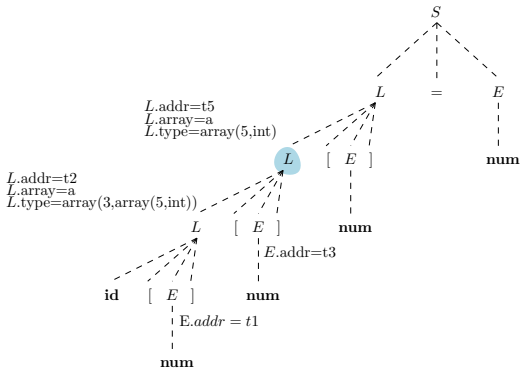



```
t1 = 2
t2 = t1 * 60
t3 = 2
```

1. *L*.array = L_1.array = symbol table entry for a

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$

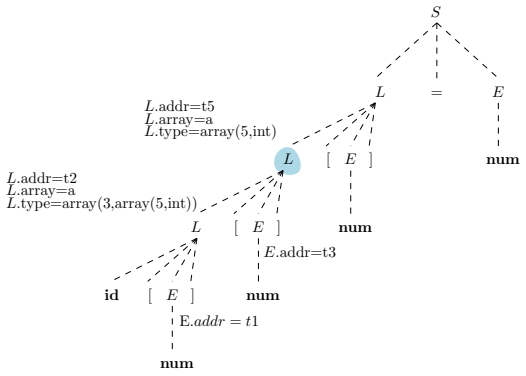



```
t1 = 2
t2 = t1 * 60
t3 = 2
```

1. *L*.array = L_1.array = symbol table entry for a
2. *L*.type = L_1.type.elem = array(5,int)

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$

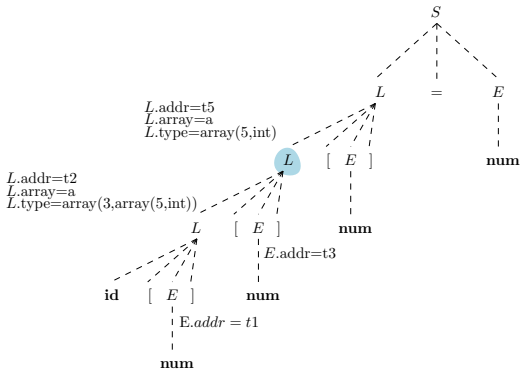



```
t1 = 2
t2 = t1 * 60
t3 = 2
```

1. L.array = L_1.array = symbol table entry for a
2. L.type = L_1.type.elem = array(5,int)
3. t = new Temp() = t4

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$

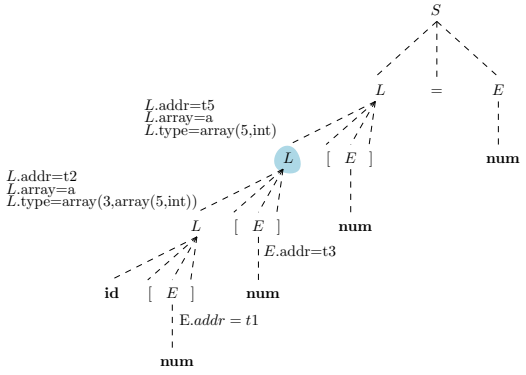



```
t1 = 2
t2 = t1 * 60
t3 = 2
```

1. L.array = L_1.array = symbol table entry for a
2. L.type = L_1.type.elem = array(5,int)
3. t = new Temp() = t4
4. L.addr = new Temp() = t5

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$

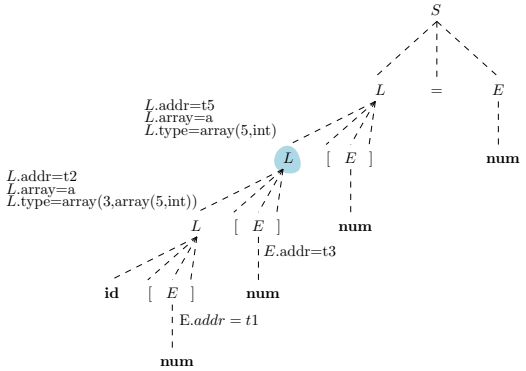



```
t1 = 2
t2 = t1 * 60
t3 = 2
t4 = t3 * 20
```

1. L.array = L_1.array = symbol table entry for a
2. L.type = L_1.type.elem = array(5,int)
3. t = new Temp() = t4
4. L.addr = new Temp() = t5
5. gen(t4 = t3 * 20)

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$

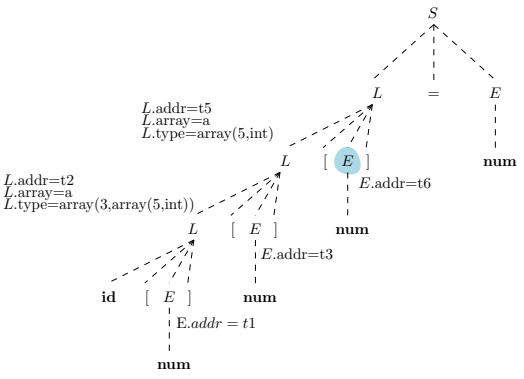



```
t1 = 2
t2 = t1 * 60
t3 = 2
t4 = t3 * 20
t5 = t2 + t4
```

1. L.array = L_1.array = symbol table entry for a
2. L.type = L_1.type.elem = array(5,int)
3. t = new Temp() = t4
4. L.addr = new Temp() = t5
5. gen(t4 = t3 * 20)
6. gen(t5 = t2 + t4)

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$

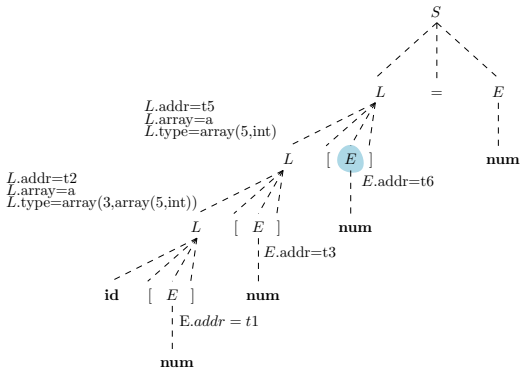



```
t1 = 2
t2 = t1 * 60
t3 = 2
t4 = t3 * 20
t5 = t2 + t4
```

1. *E*.addr = new Temp() = t6

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$

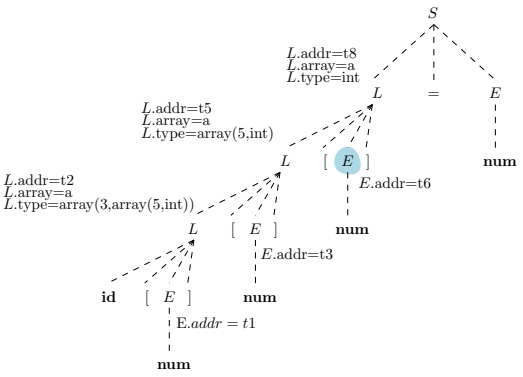



```
t1 = 2
t2 = t1 * 60
t3 = 2
t4 = t3 * 20
t5 = t2 + t4
t6 = 4
```

1. *E*.addr = new Temp() = t6
2. gen(t6 = num.yylval)

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$

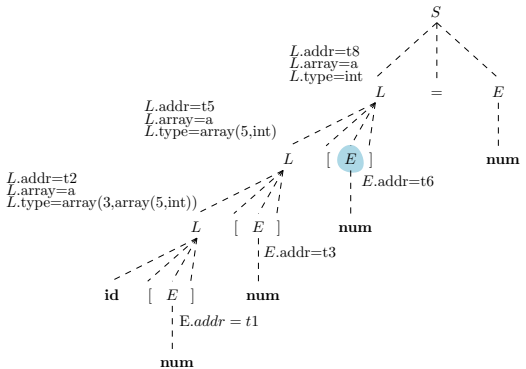



```
t1 = 2
t2 = t1 * 60
t3 = 2
t4 = t3 * 20
t5 = t2 + t4
t6 = 4
```

1. *L*.array = L_1.array = symbol table entry for a

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$

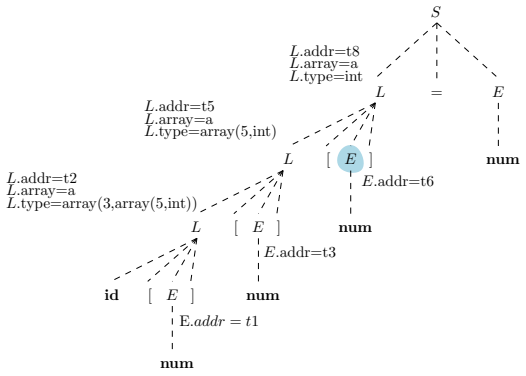



```
t1 = 2
t2 = t1 * 60
t3 = 2
t4 = t3 * 20
t5 = t2 + t4
t6 = 4
```

1. L.array = L_1.array = symbol table entry for a
2. L.type = L_1.type.elem = int

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$

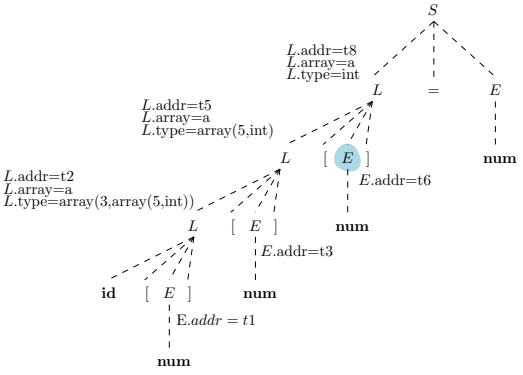



```
t1 = 2
t2 = t1 * 60
t3 = 2
t4 = t3 * 20
t5 = t2 + t4
t6 = 4
```

1. L.array = L_1.array = symbol table entry for a
2. L.type = L_1.type.elem = int
3. t = new Temp() = t7

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$

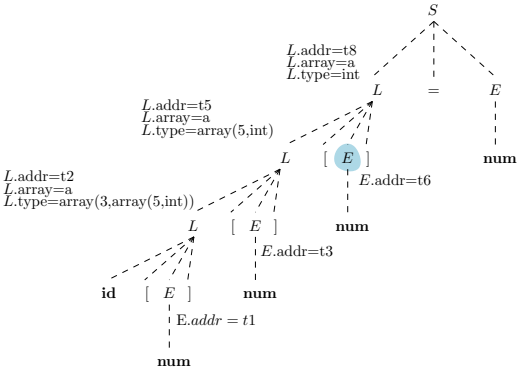



```
t1 = 2
t2 = t1 * 60
t3 = 2
t4 = t3 * 20
t5 = t2 + t4
t6 = 4
```

1. L.array = L_1.array = symbol table entry for a
2. L.type = L_1.type.elem = int
3. t = new Temp() = t7
4. L.addr = new Temp() = t8

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$

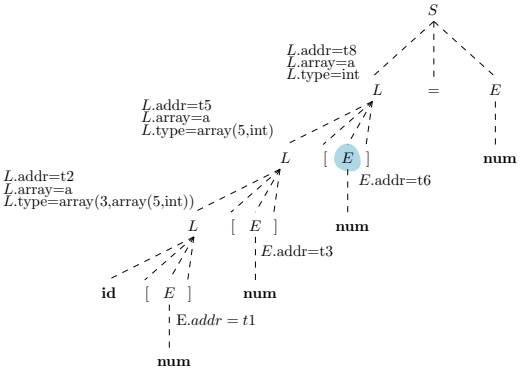



```
t1 = 2
t2 = t1 * 60
t3 = 2
t4 = t3 * 20
t5 = t2 + t4
t6 = 4
t7 = 4 * 4
```

1. L.array = L_1.array = symbol table entry for a
2. L.type = L_1.type.elem = int
3. t = new Temp() = t7
4. L.addr = new Temp() = t8
5. gen(t7 = 4 * L.type.width)

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$

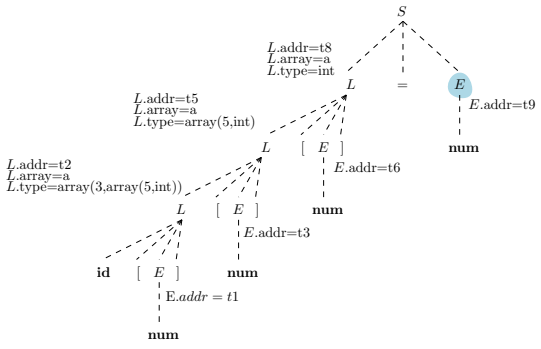



```
t1 = 2
t2 = t1 * 60
t3 = 2
t4 = t3 * 20
t5 = t2 + t4
t6 = 4
t7 = 4 * 4
t8 = t5 + t7
```

1. *L*.array = L_1.array = symbol table entry for a
2. *L*.type = L_1.type.elem = int
3. t = new Temp() = t7
4. *L*.addr = new Temp() = t8
5. gen(t7 = 4 * *L*.type.width)
6. gen(t8 = t5 + t7)

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$

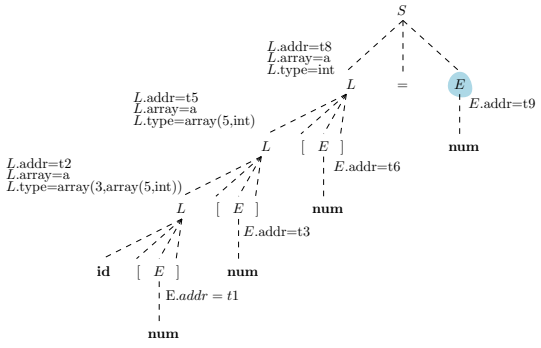



```
t1 = 2
t2 = t1 * 60
t3 = 2
t4 = t3 * 20
t5 = t2 + t4
t6 = 4
t7 = 4 * 4
t8 = t5 + t7
```

1. *E*.addr = new Temp() = t9

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$

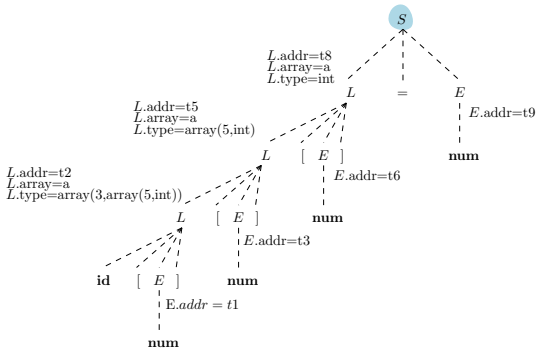



```
t1 = 2
t2 = t1 * 60
t3 = 2
t4 = t3 * 20
t5 = t2 + t4
t6 = 4
t7 = 4 * 4
t8 = t5 + t7
t9 = 1
```

1. *E*.addr = new Temp() = t9
2. gen(t9 = num.yylval)

Μετάφραση Αναφορών σε Πίνακες

Παράδειγμα: a [2][2][4]=1; με διαστάσεις πίνακα $4 \times 3 \times 5$




```
t1 = 2
t2 = t1 * 60
t3 = 2
t4 = t3 * 20
t5 = t2 + t4
t6 = 4
t7 = 4 * 4
t8 = t5 + t7
t9 = 1
a[t8] = t9
```

1. gen(*L*.array.base [*L*.addr] = *E*.addr) = gen(a [t8] = t9)

Έλεγχος Ροής

Η μετάφραση προτάσεων όπως **if-else** και **while** είναι συνδεδεμένη πολύ στενά με την μετάφραση των λογικών εκφράσεων.

Οι λογικές εκφράσεις χρησιμοποιούνται σε ένα πρόγραμμα

1. για την αλλαγή του ελέγχου ροής
2. για τον υπολογισμό λογικών τιμών.

Η μετάφραση στην δεύτερη περίπτωση είναι πανομοιότυπη με την μετάφραση που είδαμε ήδη για τις αριθμητικές εκφράσεις.

Εδώ μας ενδιαφέρει να μελετήσουμε την πρώτη περίπτωση.

Λογικές Εκφράσεις

Αλλαγή Ελέγχου Ροής ή Λογική Τιμή

Η χρήση μιας λογικής έκφρασης (ως τιμή ή για την αλλαγή του ελέγχου) καθορίζεται από την σύνταξη.

π.χ μετά από την δεσμευμένη λέξη **if** μια λογική έκφραση έχει ως σκοπό την αλλαγή του ελέγχου, ενώ στην δεξιά πλευρά μια ανάθεσης είναι λογική τιμή.

Η εξακρίβωση της κατηγορίας της λογικής έκφρασης μπορεί να γίνει με διάφορους τρόπους όπως

- μπορούμε να χρησιμοποιήσουμε 2 διαφορετικά μη-τερματικά
- να χρησιμοποιήσουμε κληρονομήσιμες ιδιότητες
- να θέσουμε μια εξωτερική μεταβλητή (σημαία) κατά την διάρκεια της συντακτικής ανάλυσης
- μπορούμε να φτιάξουμε ένα αφηρημένο συντακτικό δέντρο και να καλέσουμε διαφορετικές συναρτήσεις για κάθε λογική έκφραση, ανάλογα με την χρήση της.

Λογικές Εκφράσεις

Έστω πως έχουμε ένα ξεχωριστό μη-τερματικό **B** για τις λογικές εκφράσεις.

$$
\begin{array}{lcl}
B & \rightarrow & B \,||\, B \\
 & | & B \,\&\&\, B \\
 & | & !\ B \\
 & | & (\ B\ ) \\
 & | & E \ \textbf{rel}\ E \\
 & | & \textbf{true} \\
 & | & \textbf{false}
\end{array}
$$

Έστω πως το τερματικό **rel** είναι ένα από τα $<$, $<=$, $=$, $!=$, $>$ ή $>=$.

Οι τελεστές $||$ και $\&\&$ είναι αριστερά προσεταιριστικοί και ο τελεστής $||$ έχει μικρότερη προτεραιότητα από τον $\&\&$, ο οποίος έχει μικρότερη από τον !.

Η πρώτη δυνατή λύση είναι να προσθέσουμε τους λογικούς τελεστές στην ενδιάμεση γλώσσα και να υπολογίζουμε κανονικά τις τιμές όπως και στις αριθμητικές εκφράσεις.

π.χ ο κώδικας

```
if ( x < 100 || x > 200 && x != y )
    x = 0;
```

Θα μπορούσε να μεταφραστεί ως:

```
    t1 = x > 200
    t2 = x != y
    t3 = t1 && t2
    t4 = x < 100
    t5 = t4 || t3
    ifFalse t5 goto L2
L1: x = 0
L2:
```

Λογικές Εκφράσεις για την Αλλαγή του Ελέγχου Ροής

Ενδιάμεσος Κώδικας

Η πρώτη δυνατή λύση είναι να προσθέσουμε τους λογικούς τελεστές στην ενδιάμεση γλώσσα και να υπολογίζουμε κανονικά τις τιμές όπως και στις αριθμητικές εκφράσεις.

π.χ ο κώδικας

```
if ( x < 100 || x > 200 && x != y )
    x = 0;
```

Θα μπορούσε να μεταφραστεί ως:

```
    t1 = x > 200
    t2 = x != y
    t3 = t1 && t2
    t4 = x < 100
    t5 = t4 || t3
    ifFalse t5 goto L2
L1: x = 0
L2:
```

Το μειονέκτημα αυτής της μεθόδου είναι πως πρέπει να υπολογίσουμε όλη την έκφραση, παρόλο που θα μπορούσαμε να καταλάβουμε πιο σύντομα την τιμή της

Short-Circuit Code (Jumping)

Στον κώδικα αυτό οι τελεστές &&, || και ! δεν εμφανίζονται στον κώδικα. Η τιμή μιας λογικής έκφρασης προσδιορίζεται από μια γραμμή στο πρόγραμμα.

π.χ ο κώδικας

```
if ( x < 100 || x > 200 && x != y )
    x = 0;
```

μεταφράζεται σε

```
    if x < 100 goto L1
    ifFalse x > 200 goto L2
    ifFalse x != y goto L2
L1: x = 0
L2:
```

Short-Circuit Code (Jumping)

Στον κώδικα αυτό οι τελεστές &&, || και ! δεν εμφανίζονται στον κώδικα. Η τιμή μιας λογικής έκφρασης προσδιορίζεται από μια γραμμή στο πρόγραμμα.

π.χ ο κώδικας

```
if ( x < 100 || x > 200 && x != y )
    x = 0;
```

μεταφράζεται σε

```
    if x < 100 goto L1
    ifFalse x > 200 goto L2
    ifFalse x != y goto L2
L1: x = 0
L2:
```

Εαν ο κώδικας φτάσει στην γραμμή L1 τότε η λογική έκφραση είναι αλήθεια, αν φτάσει στην γραμμή L2 τότε η λογική έκφραση είναι ψέμα.

Short-Circuit Code (Jumping)

Το πλεονέκτημα της jumping μεθόδου είναι πως δεν χρειάζεται να υπολογίσουμε όλη την έκφραση.

Στο παρακάτω παράδειγμα

```
if ( x < 100 || x > 200 && x != y )
    x = 0;
```

ανάλογα με την τιμή του *x* ο παρακάτω κώδικας δεν εκτελεί όλες τις εντολές

```
    if x < 100 goto L1
    ifFalse x > 200 goto L2
    ifFalse x != y goto L2
L1: x = 0
L2:
```

Αλλαγή Ελέγχου Ροής

Θα ασχοληθούμε με την μετάφραση των λογικών εκφράσεων σε κώδικα 3-διευθύνσεων όσο αναφορά την παρακάτω γραμματική:

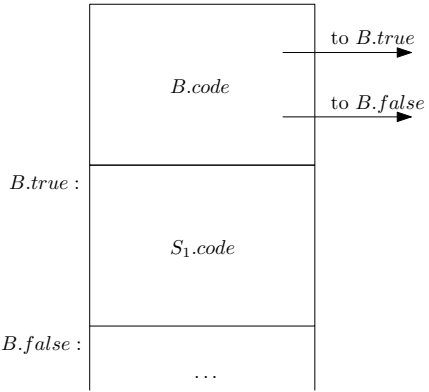
$$
\begin{array}{lcl}
S & \rightarrow & \textbf{if (} B \textbf{)} \; S_1 \\
 & | & \textbf{if (} B \textbf{)} \; S_1 \; \textbf{else} \; S_2 \\
 & | & \textbf{while (} B \textbf{)} \; S_1
\end{array}
$$

Όπου το μη-τερματικό B αναπαριστά μια λογική έκφραση και το μη-τερματικό S μια πρόταση (statement).

Αλλαγή Ελέγχου Ροής

If

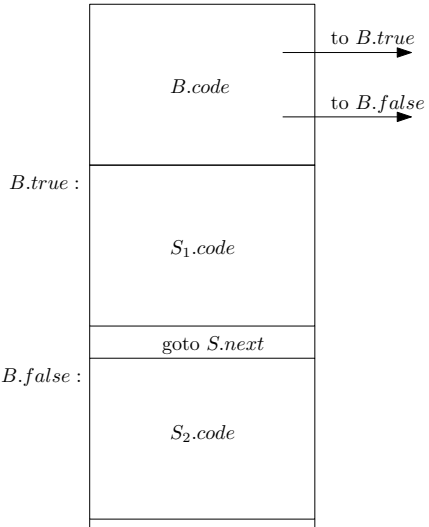
$S \rightarrow$ **if (** B **)** S_1



Αλλαγή Ελέγχου Ροής

If-Else

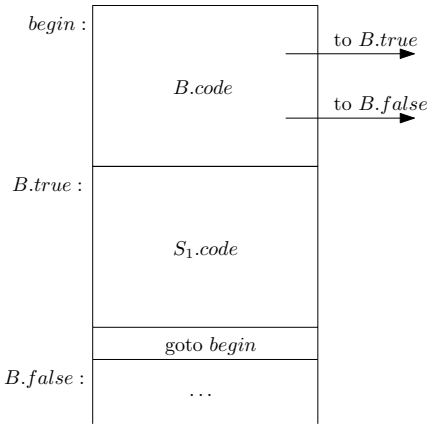
$S \rightarrow$ **if (** B **)** S_1 **else** S_2



Αλλαγή Ελέγχου Ροής

While

$S \rightarrow$ **while (** B **)** S_1



Μετάφραση Εντολών Ελέγχου Ροής

Το βασικό πρόβλημα που αντιμετωπίζουμε στην παραγωγή κώδικα για τις λογικές εκφράσεις και τις προτάσεις αλλαγής της ροής ελέγχου είναι η αντιστοίχιση των εντολών διακλάδωσης (jump) με το στόχο της διακλάδωσης (jump target).

Μετάφραση Εντολών Ελέγχου Ροής

π.χ για την μετάφραση της λογικής έκφρασης *B* στην πρόταση

if (*B* **)** *S*

ο κώδικας για το *B* περιέχει μια διακλάδωση για όταν η *B* είναι λάθος στην επόμενη εντολή μετά την τελευταία γραμμή του *S*.

(i) Σε μια μετάφραση ενός περάσματος οι εντολές για το *B* πρέπει να παραχθούν πριν από την παραγωγή του *S*.

(ii) Θα δούμε μια τεχνική που με την χρήση συνθέσιμων ιδιοτήτων μας επιτρέπει να παράγουμε ενδιάμεσο κώδικα σε ένα μόνο πέρασμα.

Backpatching

Η ιδέα είναι πως θα παράγουμε εντολές διακλάδωσης της μορφής

```
goto _
```

και μόλις βρούμε τον στόχο θα πρέπει να μπορούμε να πάμε πίσω στην παραπάνω εντολή και να προσθέσουμε την επιπλέον πληροφορία.

Για τον σκοπό αυτό θα διατηρούμε λίστες με γραμμές εντολών στις οποίες πρέπει να προσθέσουμε την επιπλέον πληροφορία μόλις μας γίνει διαθέσιμη.

Πίνακας Εντολών

Για να μπορούμε να κάνουμε αλλαγές στο ενδιάμεσο πρόγραμμα δεν το εκτυπώνουμε απευθείας στην έξοδο, αλλά το αποθηκεύουμε σε έναν πίνακα.

Κάθε θέση του πίνακα είναι μια εγγραφή που αντιστοιχεί σε μια εντολή ενδιάμεσου κώδικα.

Επόμενη Εντολή

Κρατάμε επίσης και μια μεταβλητή που μας λέει πόσο μεγάλο είναι το πρόγραμμα μας, δηλαδή ποια είναι η επόμενη θέση που θα προσθέσουμε μια νέα εντολή.

Θεωρούμε πως έχουμε διαθέσιμη μια συνάρτηση *nextinstr*() που μας παρέχει ανά πάσα στιγμή την διεύθυνση της επόμενης εντολής που θα παραχθεί στο πρόγραμμα.

Backpatching

Φανταστείτε την μετάφραση του προγράμματος

```
if ( x < 10 || y < 10 || z < 10 )
    x = 2;
else
    x = 1;
```

```
0:  if x < 10 goto _
1:  if y < 10 goto _
2:  if z < 10 goto _
3:  goto _
4:  x = 2
5:  x = 1
```

Καθώς φτάνουμε στην γραμμή 4 πλέον γνωρίζουμε πως στις εντολές 0, 1, 2 μπορούμε να πάμε και να προσθέσουμε την επιπλέον πληροφορία. Αντίστοιχα στην εντολή 5 γνωρίζουμε την πληροφορία που έλειπε για την γραμμή 3.

Η τεχνική backpatching υλοποιείται με την χρήση λιστών που περιέχουν ακριβώς αυτή την πληροφορία.

Backpatching

Θα κάνουμε μερικές μικρές αλλαγές στην παρακάτω γραμματική που θα βοηθήσουν στην υλοποίηση αυτής της ιδέας.

$$
\begin{array}{lcl}
B & \rightarrow & B \,||\, B \\
 & | & B \,\&\&\, B \\
 & | & !\ B \\
 & | & (\ B\ ) \\
 & | & E \ \textbf{rel}\ E \\
 & | & \textbf{true} \\
 & | & \textbf{false}
\end{array}
$$

$$
\begin{array}{lcl}
S & \rightarrow & \textbf{if}\ (\ B\ )\ S_1 \\
 & | & \textbf{if}\ (\ B\ )\ S_1\ \textbf{else}\ S_2 \\
 & | & \textbf{while}\ (\ B\ )\ S_1 \\
 & | & \{\ L\ \} \\
 & | & A
\end{array}
$$

$$
\begin{array}{lcl}
L & \rightarrow & L_1\ S \\
 & | & S
\end{array}
$$

B	Boolean expression
E	Expression
S	Statement
L	statement List
A	Assignment

Backpatching

Θεωρούμε πως κάθε μη-τερματικό B έχει συνθέσιμες ιδιότητες

- truelist
- falselist

Η ιδιότητα $B.truelist$ είναι μια λίστα από γραμμές εντολών διακλάδωσης όπου πρέπει να προσθέσουμε την ετικέτα ελέγχου για την περίπτωση όπου η έκφραση B είναι αλήθεια.

Η ιδιότητα $B.falselist$ είναι μια λίστα από γραμμές εντολών διακλάδωσης όπου πρέπει να προσθέσουμε την ετικέτα ελέγχου για την περίπτωση όπου η έκφραση B είναι ψέματα.

Backpatching

Θεωρούμε πως κάθε μη-τερματικό B έχει συνθέσιμες ιδιότητες

- truelist
- falselist

Καθώς παράγεται ο κώδικας για την λογική έκφραση B, οι διακλαδώσεις για τις περιπτώσεις αλήθεια και ψέματα της B παραμένουν κενές, αλλά η διεύθυνση τους καταγράφεται στις λίστες B.truelist και B.falselist.

Μετά την παραγωγή των εντολών στις γραμμές B.true και B.false, οι λίστες μας δίνουν τις γραμμές που πρέπει να προσθέσουμε την επιπλέον πληροφορία.

Backpatching

Θεωρούμε πως κάθε

- μη-τερματικό *S* (που αντιστοιχεί σε ένα statement)
- και κάθε μη-τερματικό *L* (που αντιστοιχεί σε λίστα από statements)

έχει μια συνθέσιμη ιδιότητα nextlist .

Η ιδιότητα *S.nextlist* (αντίστοιχα η *L*.nextlist) είναι μια λίστα διευθύνσεων εντολών διακλάδωσης στην εντολή που ακολουθεί το *S*.

Μόλις η εντολή που είναι στην θέση *S.next* παραχθεί, μπορούμε μέσω της λίστας *S.nextlist* να προσθέσουμε την επιπλέον πληροφορία.

Διαχείριση Λιστών και Βοηθητικές Συναρτήσεις

Για την υλοποίηση του μεταφραστικού σχήματος θα μας χρειαστούν μερικές βασικές λειτουργίες σε λίστες:

1. *makelist*(i) φτιάχνει μια νέα λίστα που περιέχει μόνο τον ακέραιο *i* και επιστρέφει έναν δείκτη στην αρχή της λίστας
2. *merge*($L1, L2$) συνεννώνει τις λίστες $L1$ και $L2$ και επιστρέφει έναν δείκτη στην νέα λίστα
3. backpatch(L, i) για κάθε ακέραιο *j* στην λίστα *L*, πηγαίνει στην εντολή *j* του προγράμματος (που πρέπει να είναι εντολή διακλάδωσης) και θέτει ως στόχο της την εντολή *i*.

Μεταφραστικό Σχήμα Λογικών Εκφράσεων

$B \rightarrow E_1$ **relop** E_2 { *B*.truelist = makelist(nextinstr());
B.falselist = makelist(nextinstr() + 1);
gen('if' E_1.addr relop.yylval E_2.addr 'goto _');
gen('goto _');

π.χ για την έκφραση $a > b$ παράγουμε

```
10:  if a > b goto _
11:  goto _
```

με $B.truelist = \{10\}$ και $B.falselist = \{11\}$.

Μεταφραστικό Σχήμα Λογικών Εκφράσεων

$B \rightarrow E_1$ **relop** E_2 { *B.truelist* = makelist(nextinstr());
B.falselist = makelist(nextinstr() + 1);
gen('if' E_1.addr relop.yylval E_2.addr 'goto _');
gen('goto _'); }

$B \rightarrow$ **true** { *B.truelist* = makelist(nextinstr());
gen('goto _'); }

$B \rightarrow$ **false** { *B.falselist* = makelist(nextinstr());
gen('goto _'); }

Μεταφραστικό Σχήμα Λογικών Εκφράσεων

$$
\begin{array}{lcll}
B & \rightarrow & B_1 \;||\; M\; B_2 & \{ \quad \text{backpatch}(B_1.\mathit{falselist}, M.\mathit{instr}); \\
 & & & \quad B.\mathit{truelist} = \text{merge}(B_1.\mathit{truelist}, B_2.\mathit{truelist}); \\
 & & & \quad B.\mathit{falselist} = B_2.\mathit{falselist}; \quad \} \\
M & \rightarrow & \epsilon & \{ \quad M.\mathit{instr} = \text{nextinstr}(); \quad \}
\end{array}
$$

Επειδή πρέπει να γνωρίζουμε την διεύθυνση εντολών στην μέση των κανόνων, χρησιμοποιούμε ένα επιπλέον μη-τερματικό M (Marker) το οποίο επιστρέφει ως συνθέσιμη ιδιότητα την διεύθυνση της επόμενης εντολής.

Μεταφραστικό Σχήμα Λογικών Εκφράσεων

$$
\begin{array}{lcll}
B & \rightarrow & B_1 \;||\; M\; B_2 & \{ \quad \text{backpatch}(B_1.\mathit{falselist}, M.\mathit{instr}); \\
 & & & \quad\; B.\mathit{truelist} = \text{merge}(B_1.\mathit{truelist}, B_2.\mathit{truelist}); \\
 & & & \quad\; B.\mathit{falselist} = B_2.\mathit{falselist}; \qquad \} \\
M & \rightarrow & \epsilon & \{ \quad M.\mathit{instr} = \text{nextinstr}(); \qquad \}
\end{array}
$$

Στην περίπτωση του OR διαβάζουμε με την βοήθεια του μη-τερματικού M την γραμμή της πρώτης εντολής του μη-τερματικού B_2. Σε περίπτωση που το B_1 είναι ψέμα φροντίζουμε να εκτελεστεί ο κώδικας του B_2 για να πάρουμε απόφαση για την συνολική έκφραση.

Στην ουσία γνωρίζουμε πως το $B_1.\mathit{false}$ είναι η πρώτη γραμμή του B_2 που την έχουμε αποθηκευμένη στην ιδιότητα $M.\mathit{instr}$.

Μεταφραστικό Σχήμα Λογικών Εκφράσεων

$B \rightarrow B_1$ **&&** $M\ B_2$ { backpatch(B_1.*truelist*, *M.instr*);
B.truelist = B_2.*truelist*;
B.falselist = merge(B_1.*falselist*, B_2.*falselist*); }

$M \rightarrow \epsilon$ { *M.instr* = nextinstr(); }

Μεταφραστικό Σχήμα Λογικών Εκφράσεων

B	$\rightarrow$	B_1 **&&** M B_2	{	backpatch(B_1.*truelist*, M.*instr*); B.*truelist* = B_2.*truelist*; B.*falselist* = merge(B_1.*falselist*, B_2.*falselist*);	}
M	$\rightarrow$	ϵ	{	M.*instr* = nextinstr();	}

Στην περίπτωση του AND διαβάζουμε με την βοήθεια του μη-τερματικού M την γραμμή της πρώτης εντολής του μη-τερματικού B_2. Εαν το B_1 είναι αλήθεια φροντίζουμε ο έλεγχος να μεταφερθεί στο B_2. Σε περίπτωση που ένα από τα B_1 ή B_2 είναι ψέμα, τότε όλη η έκφραση είναι ψέμα.

Παράδειγμα

`x < 100 || x > 200 && x != y`

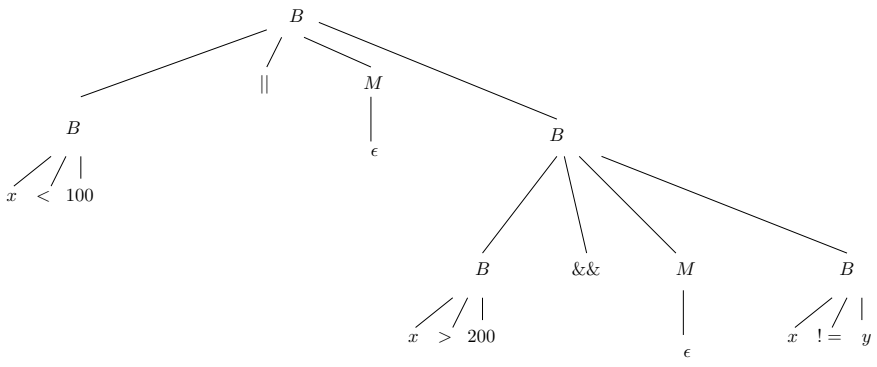
Έστω η λογική έκφραση

```
x < 100 || x > 200 && x != y
```

και έστω πως με την χρήση ενός ανοδικού αναλυτή παράγουμε ενδιάμεσο κώδικα του οποίου η επόμενη εντολή είναι η 100.

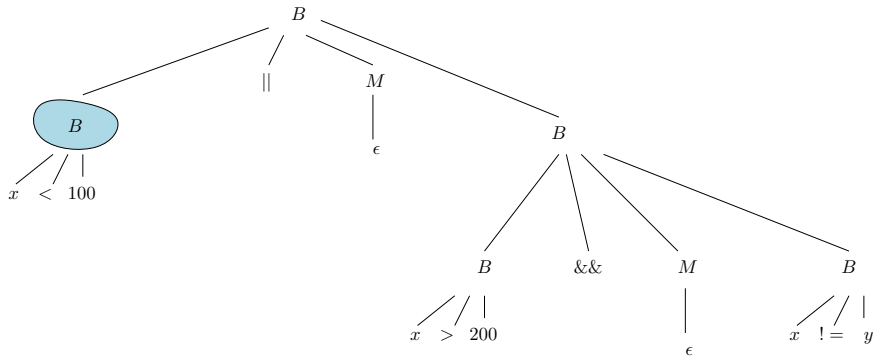
Παράδειγμα

`x < 100 || x > 200 && x != y`



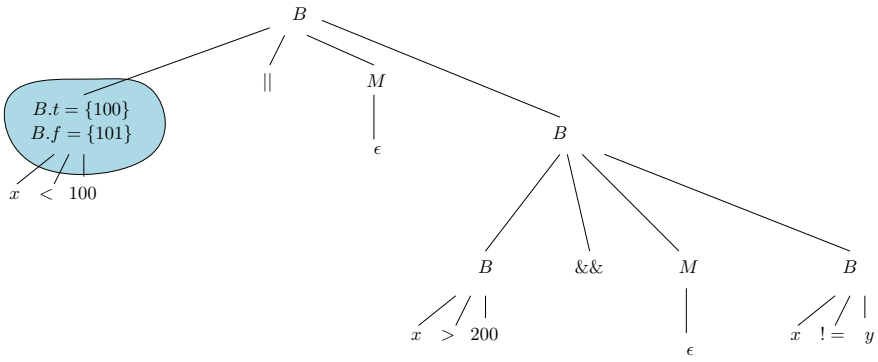
Παράδειγμα

`x < 100 || x > 200 && x != y`



Παράδειγμα

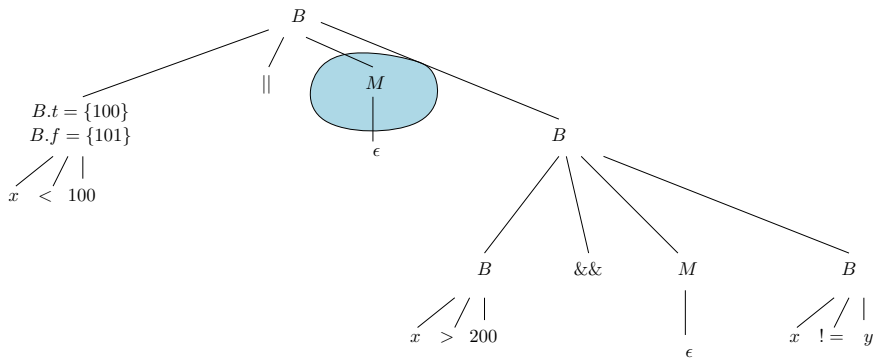
`x < 100 || x > 200 && x != y`




```
100: if x < 100 goto _
101: goto _
```

Παράδειγμα

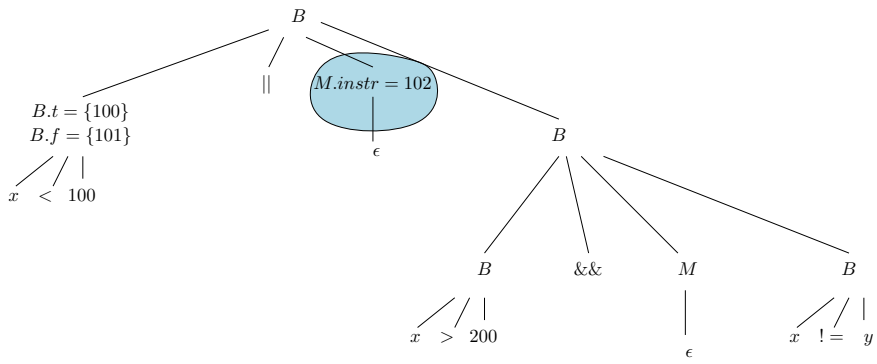
`x < 100 || x > 200 && x != y`




```
100: if x < 100 goto _
101: goto _
```

Παράδειγμα

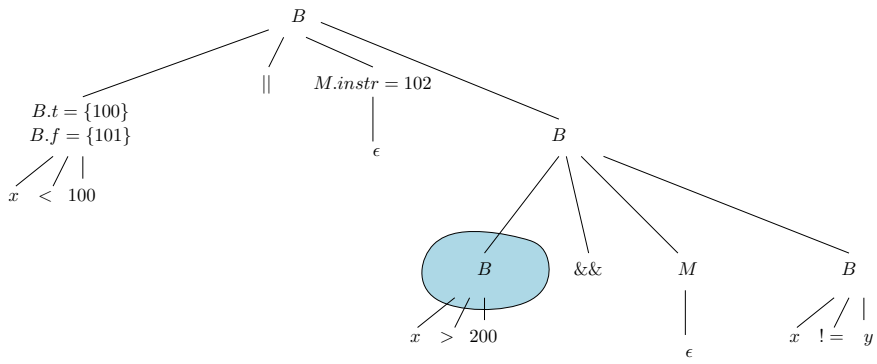
`x < 100 || x > 200 && x != y`




```
100: if x < 100 goto _
101: goto _
```

Παράδειγμα

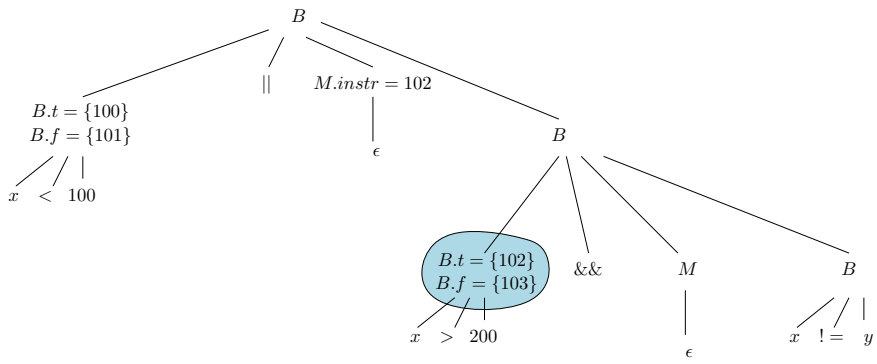
`x < 100 || x > 200 && x != y`




```
100: if x < 100 goto _
101: goto _
```

Παράδειγμα

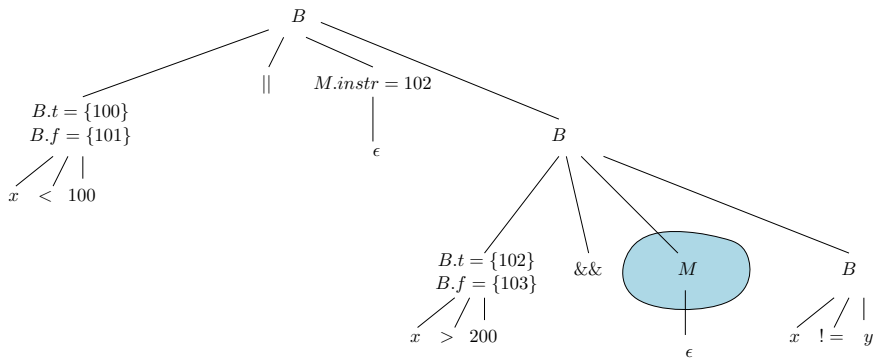
`x < 100 || x > 200 && x != y`




```
100: if x < 100 goto _
101: goto _
102: if x > 200 goto _
103: goto _
```

Παράδειγμα

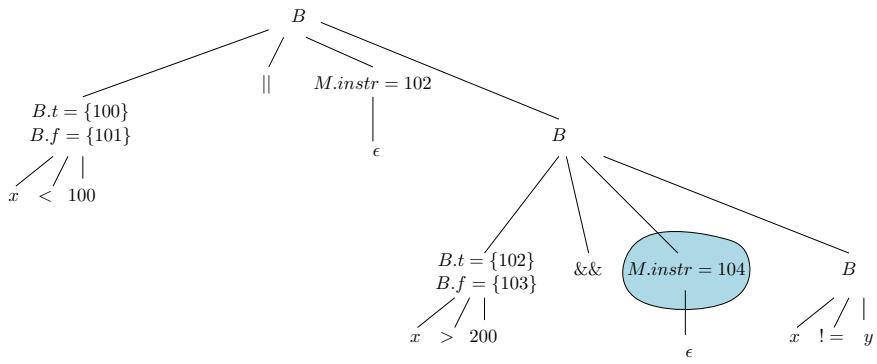
`x < 100 || x > 200 && x != y`




```
100: if x < 100 goto _
101: goto _
102: if x > 200 goto _
103: goto _
```

Παράδειγμα

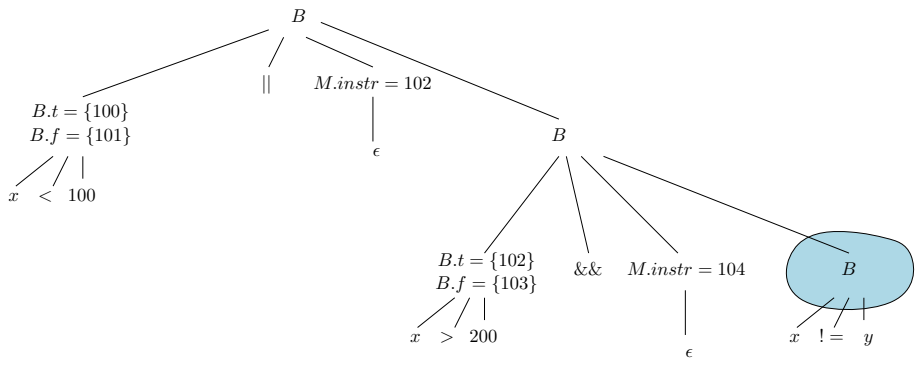
`x < 100 || x > 200 && x != y`




```
100: if x < 100 goto _
101: goto _
102: if x > 200 goto _
103: goto _
```

Παράδειγμα

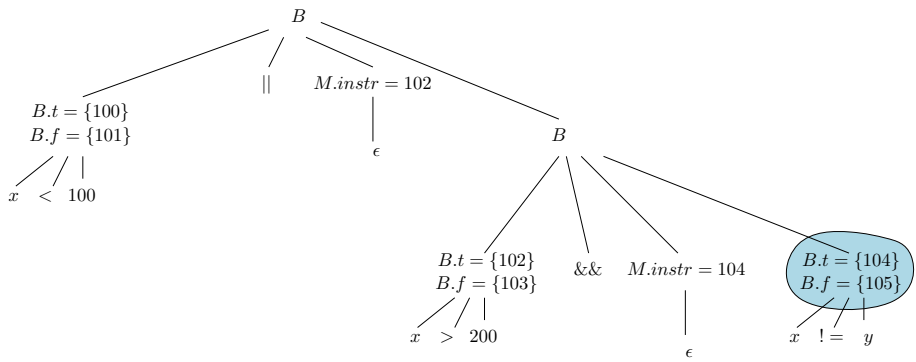
`x < 100 || x > 200 && x != y`




```
100: if x < 100 goto _
101: goto _
102: if x > 200 goto _
103: goto _
```

Παράδειγμα

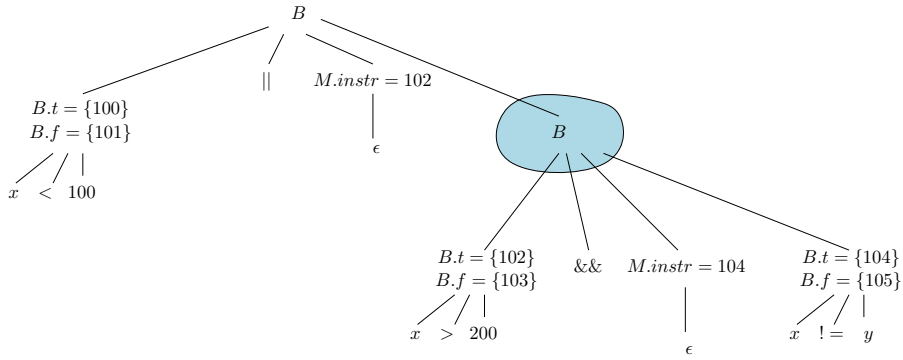
`x < 100 || x > 200 && x != y`




```
100: if x < 100 goto _
101: goto _
102: if x > 200 goto _
103: goto _
104: if x != y goto _
```

Παράδειγμα

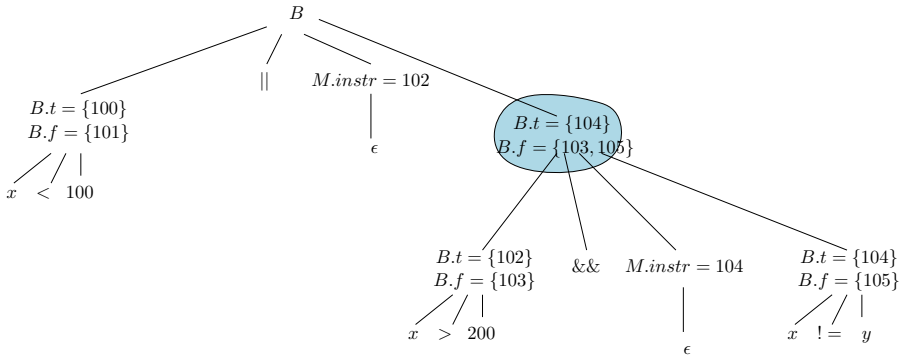
`x < 100 || x > 200 && x != y`




```
100: if x < 100 goto _
101: goto _
102: if x > 200 goto _
103: goto _
104: if x != y goto
```

Παράδειγμα

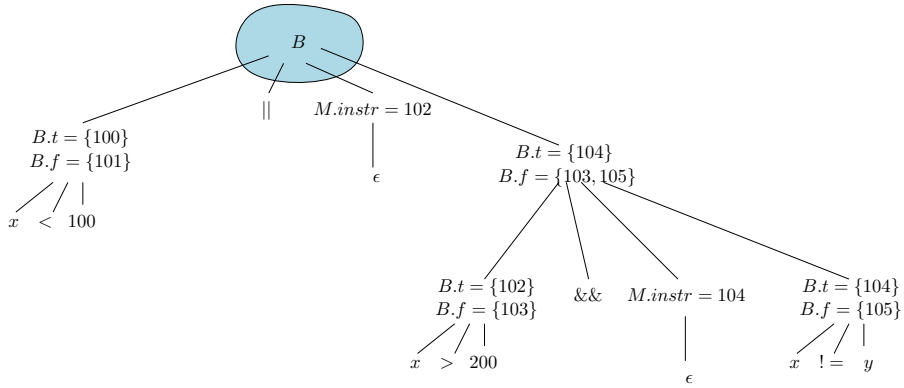
`x < 100 || x > 200 && x != y`




```
100: if x < 100 goto _
101: goto _
102: if x > 200 goto 104
103: goto _
104: if x != y goto
```

Παράδειγμα

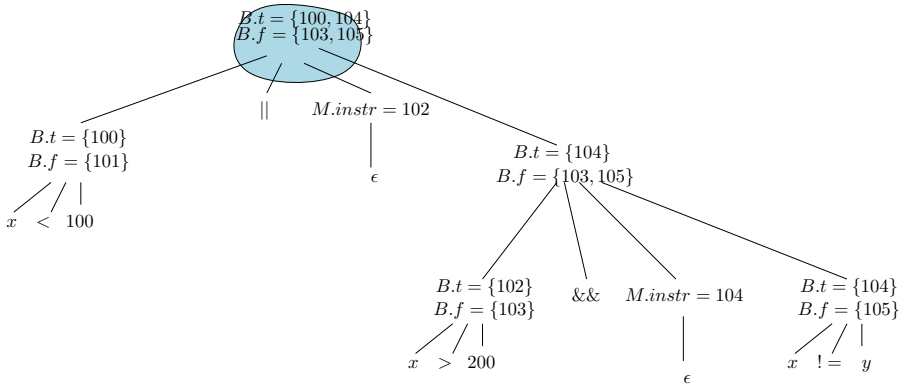
`x < 100 || x > 200 && x != y`




```
100: if x < 100 goto _
101: goto _
102: if x > 200 goto 104
103: goto _
104: if x != y goto
```

Παράδειγμα

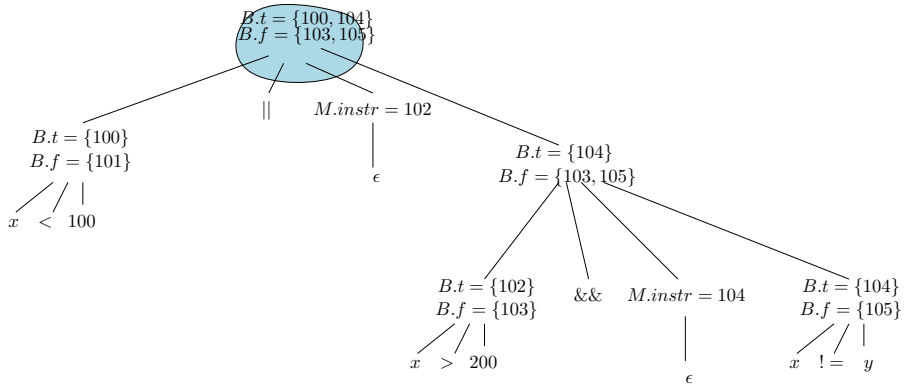
`x < 100 || x > 200 && x != y`




```
100: if x < 100 goto _
101: goto 102
102: if x > 200 goto 104
103: goto _
104: if x != y goto
```

Παράδειγμα

`x < 100 || x > 200 && x != y`




```
100: if x < 100 goto _
101: goto 102
102: if x > 200 goto 104
103: goto _
104: if x != y goto
```

Όλη η έκφραση είναι αλήθεια αν τα gotos στις γραμμές 100 ή 104 εκτελεστούν και ψέματα αν τα gotos στις γραμμές 103 ή 105 εκτελεστούν.

Μεταφραστικό Σχήμα Λογικών Εκφράσεων

$B \rightarrow \mathbf{!}\ B_1 \quad \{ \quad B.truelist = B_1.falselist;$
$B.falselist = B_1.truelist; \quad \}$

$B \rightarrow \mathbf{(}\ B_1\ \mathbf{)} \quad \{ \quad B.truelist = B_1.truelist;$
$B.falselist = B_1.falselist; \quad \}$

Μεταφραστικό Σχήμα Εντολών Αλλαγής Ροής Ελέγχου

While

$$
\begin{array}{lcll}
S & \rightarrow & \textbf{while}\ M_1\ (\ B\ )\ M_2\ S_1 & \{\ \ \text{backpatch}(S_1.\mathit{nextlist}, M_1.\mathit{instr});\\
 & & & \quad \text{backpatch}(B.\mathit{truelist}, M_2.\mathit{instr});\\
 & & & \quad S.\mathit{nextlist} = B.\mathit{falselist}\\
 & & & \quad \text{gen}(\text{'goto'}\ M_1.\mathit{instr});\ \ \}
\end{array}
$$

Μεταφραστικό Σχήμα Εντολών Αλλαγής Ροής Ελέγχου

While

$S \rightarrow$ **while** M_1 **(** B **)** M_2 S_1 { backpatch(S_1.*nextlist*, M_1.*instr*);
backpatch(B.*truelist*, M_2.*instr*);
S.*nextlist* = B.*falselist*
gen('goto' M_1.*instr*); }

(i) Για την υλοποίηση του **while** χρειαζόμαστε δύο M (Markers) ώστε να γνωρίζουμε την θέση της πρώτης γραμμής του B αλλά και του S_1.

Μεταφραστικό Σχήμα Εντολών Αλλαγής Ροής Ελέγχου

While

$$
\begin{array}{lcll}
S & \rightarrow & \textbf{while}\ M_1\ (\ B\ )\ M_2\ S_1 & \{\ \ \text{backpatch}(S_1.\mathit{nextlist}, M_1.\mathit{instr});\\
 & & & \quad \text{backpatch}(B.\mathit{truelist}, M_2.\mathit{instr});\\
 & & & \quad S.\mathit{nextlist} = B.\mathit{falselist}\\
 & & & \quad \text{gen}(\text{'goto'}\ M_1.\mathit{instr});\qquad\qquad \}
\end{array}
$$

(i) Για την υλοποίηση του **while** χρειαζόμαστε δύο M (Markers) ώστε να γνωρίζουμε την θέση της πρώτης γραμμής του B αλλά και του S_1.

(ii) Θέτουμε σε όλες τις διακλαδώσεις (jumps) για το $S_1.\mathit{nextlist}$ την πρώτη γραμμή του B.

Μεταφραστικό Σχήμα Εντολών Αλλαγής Ροής Ελέγχου

While

$$
\begin{array}{lcll}
S & \rightarrow & \textbf{while}\ M_1\ (\ B\ )\ M_2\ S_1 & \{\ \ \text{backpatch}(S_1.\mathit{nextlist}, M_1.\mathit{instr});\\
 & & & \quad \text{backpatch}(B.\mathit{truelist}, M_2.\mathit{instr});\\
 & & & \quad S.\mathit{nextlist} = B.\mathit{falselist}\\
 & & & \quad \text{gen}(\text{'goto'}\ M_1.\mathit{instr});\qquad\qquad \}
\end{array}
$$

(i) Για την υλοποίηση του **while** χρειαζόμαστε δύο M (Markers) ώστε να γνωρίζουμε την θέση της πρώτης γραμμής του B αλλά και του S_1.

(ii) Θέτουμε σε όλες τις διακλαδώσεις (jumps) για το $S_1.\mathit{nextlist}$ την πρώτη γραμμή του B.

(iii) Θέτουμε σε όλες τις διακλαδώσεις (jumps) για το $B.\mathit{truelist}$ την πρώτη γραμμή του S_1.

Μεταφραστικό Σχήμα Εντολών Αλλαγής Ροής Ελέγχου

While

$S \rightarrow$ **while** M_1 **(** B **)** M_2 S_1 { backpatch(S_1.*nextlist*, M_1.*instr*);
backpatch(B.*truelist*, M_2.*instr*);
S.*nextlist* = B.*falselist*
gen('goto' M_1.*instr*); }

(i) Για την υλοποίηση του **while** χρειαζόμαστε δύο M (Markers) ώστε να γνωρίζουμε την θέση της πρώτης γραμμής του B αλλά και του S_1.

(ii) Θέτουμε σε όλες τις διακλαδώσεις (jumps) για το S_1.*nextlist* την πρώτη γραμμή του B.

(iii) Θέτουμε σε όλες τις διακλαδώσεις (jumps) για το B.*truelist* την πρώτη γραμμή του S_1.

(iv) Στην περίπτωση που το B είναι false, όλες οι διακλαδώσεις πρέπει να φύγουν εκτός S.

Μεταφραστικό Σχήμα Εντολών Αλλαγής Ροής Ελέγχου

While

$S \rightarrow$ **while** M_1 **(** B **)** M_2 S_1 { backpatch(S_1.*nextlist*, M_1.*instr*);
backpatch(B.*truelist*, M_2.*instr*);
S.*nextlist* = B.*falselist*
gen('goto' M_1.*instr*); }

(i) Για την υλοποίηση του **while** χρειαζόμαστε δύο M (Markers) ώστε να γνωρίζουμε την θέση της πρώτης γραμμής του B αλλά και του S_1.

(ii) Θέτουμε σε όλες τις διακλαδώσεις (jumps) για το S_1.*nextlist* την πρώτη γραμμή του B.

(iii) Θέτουμε σε όλες τις διακλαδώσεις (jumps) για το B.*truelist* την πρώτη γραμμή του S_1.

(iv) Στην περίπτωση που το B είναι false, όλες οι διακλαδώσεις πρέπει να φύγουν εκτός S.

(v) Στο τέλος παράγουμε και ένα goto την πρώτη γραμμή του B, σε περίπτωση που ο κώδικας μέσα στο S_1 δεν εκτέλεσε κάποιο jump.

Μεταφραστικό Σχήμα Εντολών Αλλαγής Ροής Ελέγχου

While

$S \rightarrow$ **while** M_1 **(** B **)** M_2 S_1 { backpatch(S_1.*nextlist*, M_1.*instr*);
backpatch(B.*truelist*, M_2.*instr*);
S.*nextlist* = B.*falselist*;
gen('goto' M_1.*instr*); }

$S \rightarrow A$ { S.*nextlist* = null; }

$L \rightarrow S$ { L.*nextlist* = S.*nextlist*; }

$L \rightarrow L_1$ M S { backpatch(L_1.*nextlist*, M.*instr*); }
{ L.*nextlist* = S.*nextlist*; }

Μεταφραστικό Σχήμα Εντολών Αλλαγής Ροής Ελέγχου

While

S	$\rightarrow$	**while** M_1 **(** B **)** M_2 S_1	{	backpatch($S_1.nextlist$, $M_1.instr$);
				backpatch($B.truelist$, $M_2.instr$);
				$S.nextlist = B.falselist$;
				gen('goto' $M_1.instr$);
					}

S	$\rightarrow$	A	{	$S.nextlist$ = null;	}
L	$\rightarrow$	S	{	$L.nextlist = S.nextlist$;	}
L	$\rightarrow$	L_1 M S	{	backpatch($L_1.nextlist$, $M.instr$); $L.nextlist = S.nextlist$;	} }

(i) Στην περίπτωση που το S είναι ανάθεση A, αφήνουμε το nextlist κενό, αφού δεν υπάρχει κάποιο jump.

(ii) Στην περίπτωση που έχουμε λίστα από statements, στέλνουμε όλα τα jumps εκτός της λίστας στο επόμενο statement.

Παράδειγμα While

Ας δούμε πως μεταφράζεται ο παρακάτω κώδικας με βάση το μεταφραστικό σχήμα που έχουμε παρουσιάσει.

```
y = 300;
x = 0;
while( x < 100 || x > 200 && x != y )
    x = x + 1;
```

Παράδειγμα While

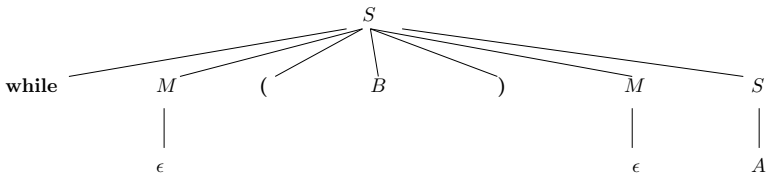
Ας δούμε πως μεταφράζεται ο παρακάτω κώδικας με βάση το μεταφραστικό σχήμα που έχουμε παρουσιάσει.

```
y = 300;
x = 0;
while( x < 100 || x > 200 && x != y )
    x = x + 1;
```

Θα κάνουμε την υπόθεση πως οι πρώτες δύο γραμμές παράγουν τον παρακάτω κώδικα

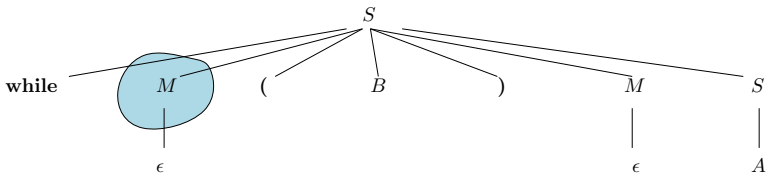
```
96: t1 = 300
97: y = t1
98: t2 = 0
99: x = t2
```

Παράδειγμα While



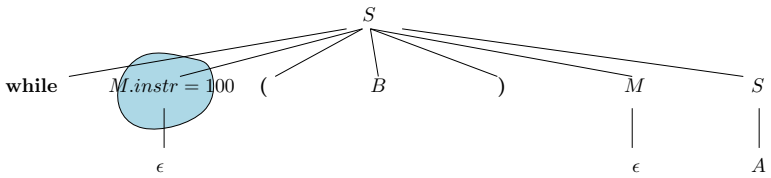

```
96: t1 = 300
97: y = t1
98: t2 = 0
99: x = t2
```

Παράδειγμα While



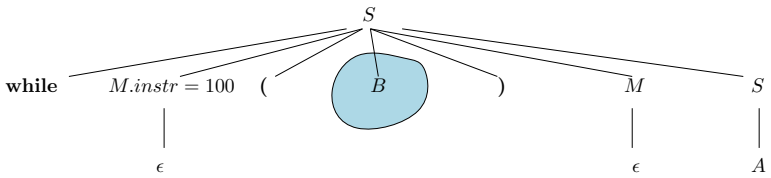

```
96: t1 = 300
97: y = t1
98: t2 = 0
99: x = t2
```

Παράδειγμα While



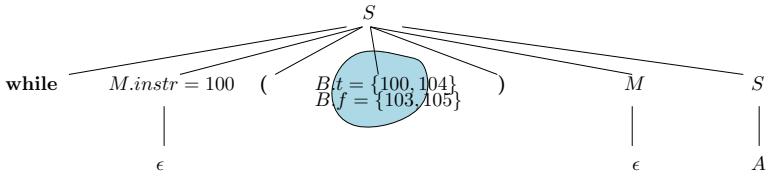

```
96: t1 = 300
97: y = t1
98: t2 = 0
99: x = t2
```

Παράδειγμα While



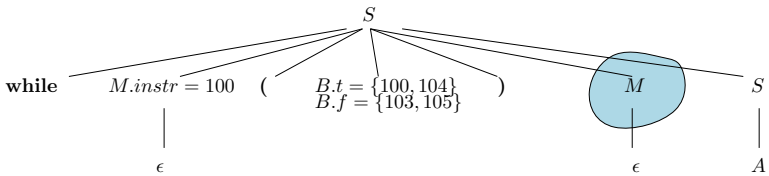

```
96: t1 = 300
97: y = t1
98: t2 = 0
99: x = t2
```

Παράδειγμα While



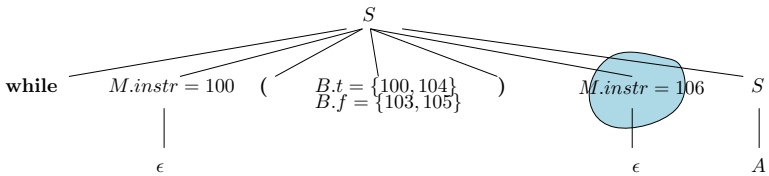

```
 96: t1 = 300
 97: y = t1
 98: t2 = 0
 99: x = t2
100: if x < 100 goto _
101: goto 102
102: if x > 200 goto 104
103: goto _
104: if x != y goto _
```

Παράδειγμα While



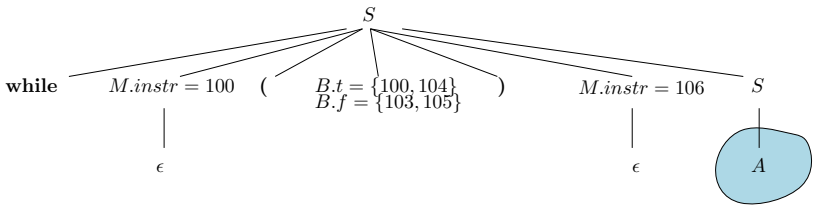

```
 96: t1 = 300
 97: y = t1
 98: t2 = 0
 99: x = t2
100: if x < 100 goto _
101: goto 102
102: if x > 200 goto 104
103: goto _
104: if x != y goto _
```

Παράδειγμα While



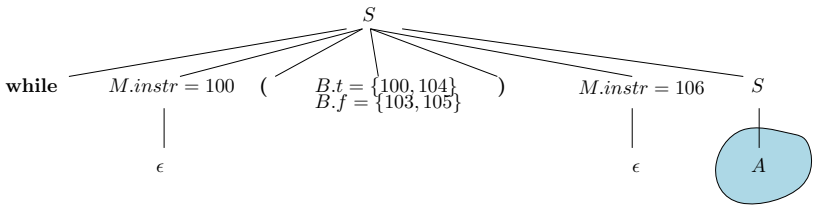

```
 96: t1 = 300
 97: y = t1
 98: t2 = 0
 99: x = t2
100: if x < 100 goto _
101: goto 102
102: if x > 200 goto 104
103: goto _
104: if x != y goto _
```

Παράδειγμα While




```
 96: t1 = 300
 97: y = t1
 98: t2 = 0
 99: x = t2
100: if x < 100 goto _
101: goto 102
102: if x > 200 goto 104
103: goto _
104: if x != y goto _
```

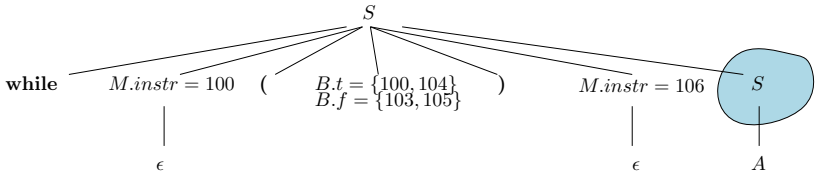
Παράδειγμα While




```
 96: t1 = 300
 97: y = t1
 98: t2 = 0
 99: x = t2
100: if x < 100 goto _
101: goto 102
102: if x > 200 goto 104
103: goto _
104: if x != y goto _
```

```
106: t3 = x + 1
107: x = t3
```

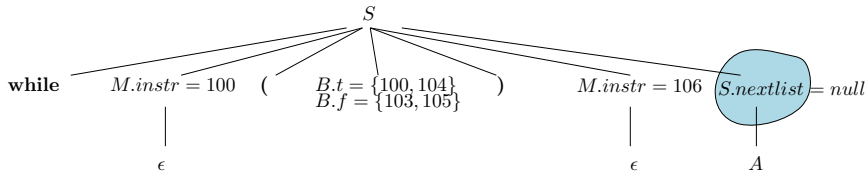
Παράδειγμα While




```
 96: t1 = 300
 97: y = t1
 98: t2 = 0
 99: x = t2
100: if x < 100 goto _
101: goto 102
102: if x > 200 goto 104
103: goto _
104: if x != y goto _
```

```
106: t3 = x + 1
107: x = t3
```

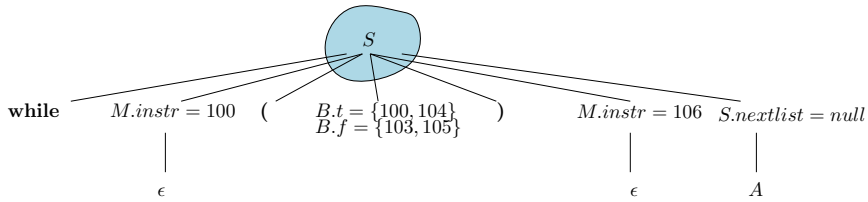
Παράδειγμα While




```
 96: t1 = 300
 97: y = t1
 98: t2 = 0
 99: x = t2
100: if x < 100 goto _
101: goto 102
102: if x > 200 goto 104
103: goto _
104: if x != y goto _
```

```
106: t3 = x + 1
107: x = t3
```

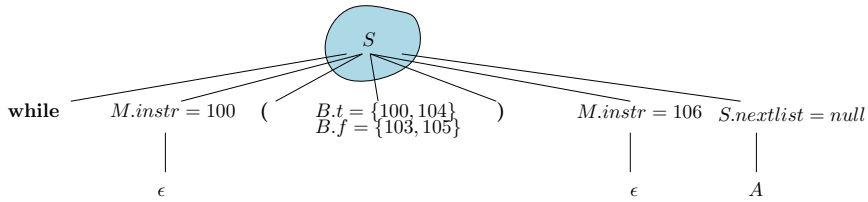
Παράδειγμα While




```
 96: t1 = 300
 97: y = t1
 98: t2 = 0
 99: x = t2
100: if x < 100 goto _
101: goto 102
102: if x > 200 goto 104
103: goto _
104: if x != y goto _
```

```
106: t3 = x + 1
107: x = t3
```

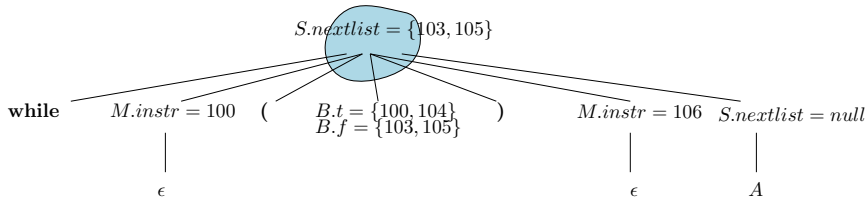
Παράδειγμα While




```
 96: t1 = 300
 97: y = t1
 98: t2 = 0
 99: x = t2
100: if x < 100 goto 106
101: goto 102
102: if x > 200 goto 104
103: goto _
104: if x != y goto 106
```

```
106: t3 = x + 1
107: x = t3
```

Παράδειγμα While




```
 96: t1 = 300
 97: y = t1
 98: t2 = 0
 99: x = t2
100: if x < 100 goto 106
101: goto 102
102: if x > 200 goto 104
103: goto _
104: if x != y goto 106
```

```
106: t3 = x + 1
107: x = t3
108: goto 100
```

Μεταφραστικό Σχήμα Εντολών Αλλαγής Ροής Ελέγχου

If

$$S \rightarrow \textbf{if (}\ B\ \textbf{)}\ M\ S_1 \quad \{ \quad \text{backpatch}(B.truelist, M.instr);$$
$$S.nextlist = \text{merge}(B.falselist, S_1.nextlist); \quad \}$$

$S \rightarrow$ **if (** B **)** $M\ S_1$ { backpatch($B.truelist$, $M.instr$);
$S.nextlist$ = merge($B.falselist$, $S_1.nextlist$); }

(i) Η υλοποίηση του If χρειάζεται μόνο ένα Marker ώστε να γνωρίζουμε την πρώτη γραμμή της πρότασης του if.

$S \rightarrow$ **if (** B **)** $M\ S_1$ { backpatch($B.truelist$, $M.instr$);
$S.nextlist$ = merge($B.falselist$, $S_1.nextlist$); }

(i) Η υλοποίηση του If χρειάζεται μόνο ένα Marker ώστε να γνωρίζουμε την πρώτη γραμμή της πρότασης του if.

(ii) Η πρώτη γραμμή της πρότασης S_1 είναι το $B.true$ και άρα κάνουμε backpatch.

Μεταφραστικό Σχήμα Εντολών Αλλαγής Ροής Ελέγχου

If-Else

$S \rightarrow$ **if (** B **)** $M_1\ S_1\ N$ **else** $M_2\ S_2$ {

backpatch($B.truelist$, $M_1.instr$);
backpatch($B.falselist$, $M_2.instr$);
T = merge($S_1.nextlist$, $N.nextlist$);
$S.nextlist$ = merge(T, $S_2.nextlist$);

}

$N \rightarrow \epsilon$ { $N.nextlist$ = makelist(nextinstr());
gen('goto _'); }

$M \rightarrow \epsilon$ { $M.instr$ = nextinstr(); }

Μεταφραστικό Σχήμα Εντολών Αλλαγής Ροής Ελέγχου

If-Else

$S \rightarrow$ **if (** B **)** $M_1\ S_1\ N$ **else** $M_2\ S_2$ {
backpatch($B.truelist$, $M_1.instr$);
backpatch($B.falselist$, $M_2.instr$);
T = merge($S_1.nextlist$, $N.nextlist$);
$S.nextlist$ = merge(T, $S_2.nextlist$);
}

$N \rightarrow \epsilon$ { $N.nextlist$ = makelist(nextinstr());
gen('goto _'); }

$M \rightarrow \epsilon$ { $M.instr$ = nextinstr(); }

- Η υλοποίηση του If-Else εκτός από Marker πριν από τις δύο προτάσεις, χρειάζεται και την προσθήκη μιας εντολής goto για να μην εκτελεστεί και το else σε περίπτωση που εκτελεστεί το if.

Μεταφραστικό Σχήμα Εντολών Αλλαγής Ροής Ελέγχου

If-Else

$S \rightarrow$ **if (** B **)** $M_1\ S_1\ N$ **else** $M_2\ S_2$ {
backpatch($B.truelist$, $M_1.instr$);
backpatch($B.falselist$, $M_2.instr$);
T = merge($S_1.nextlist$, $N.nextlist$);
$S.nextlist$ = merge(T, $S_2.nextlist$);
}

$N \rightarrow \epsilon$ { $N.nextlist$ = makelist(nextinstr());
gen('goto _'); }

$M \rightarrow \epsilon$ { $M.instr$ = nextinstr(); }

(i) Η υλοποίηση του If-Else εκτός από Marker πριν από τις δύο προτάσεις, χρειάζεται και την προσθήκη μιας εντολής goto για να μην εκτελεστεί και το else σε περίπτωση που εκτελεστεί το if.

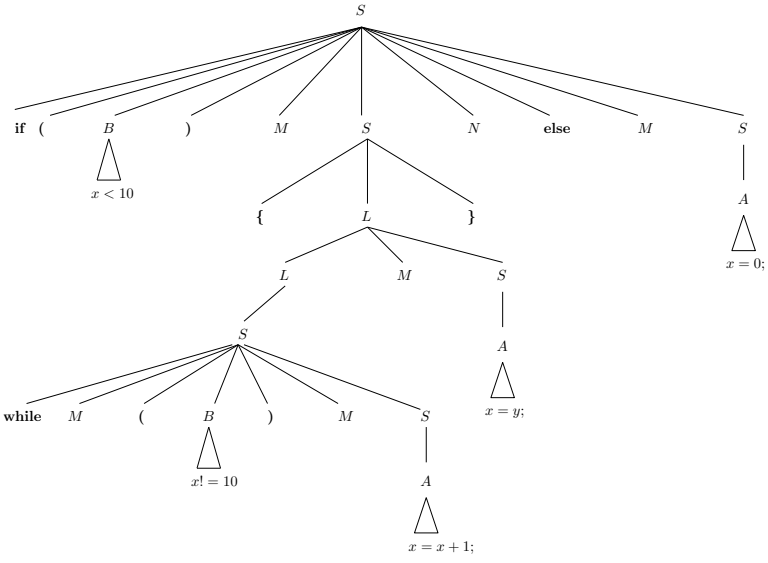
(ii) Χρησιμοποιούμε ένα καινούριο μη-τερματικό N για να υλοποιήσουμε αυτή την περίπτωση.

Παράδειγμα If-Else

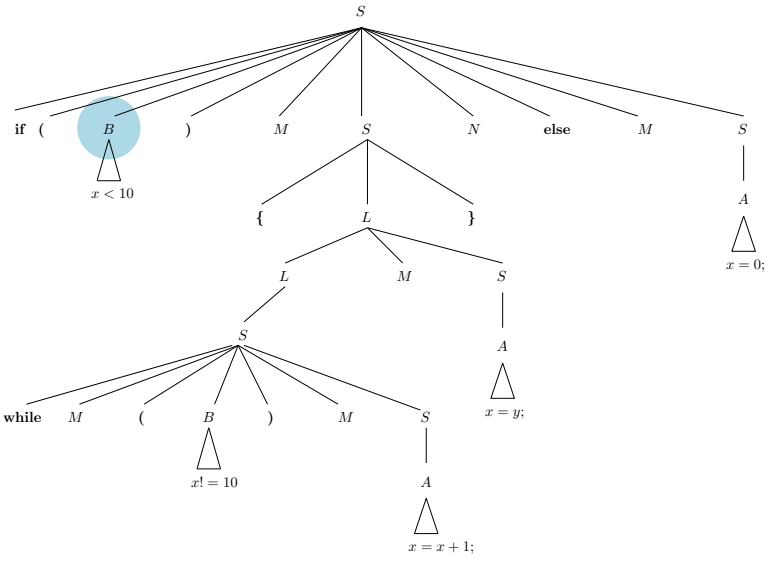
Ας δούμε πως μεταφράζεται το παρακάτω πρόγραμμα από το μεταφραστικό σχήμα που παρουσιάσαμε.

```
if ( x < 10 ) {
    while( x != 10 )
        x = x + 1;
    y = x;
}
else
    x = 0;
```

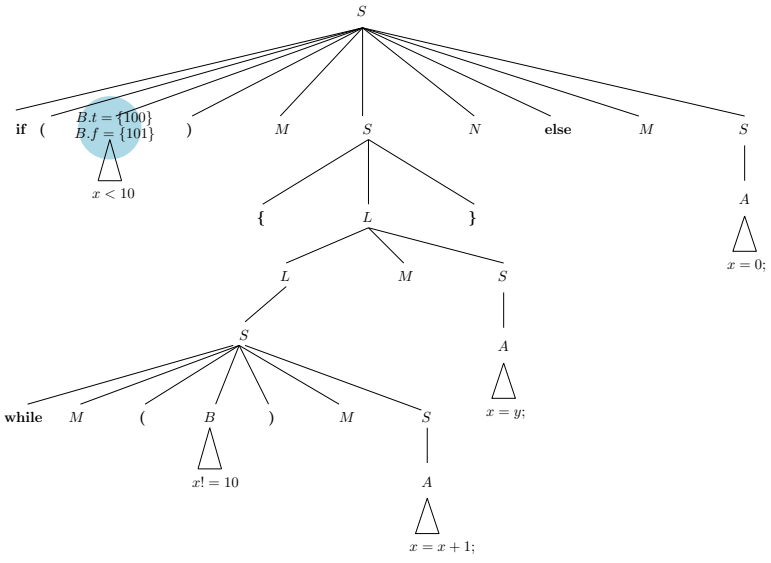
Παράδειγμα If-Else



Παράδειγμα If-Else

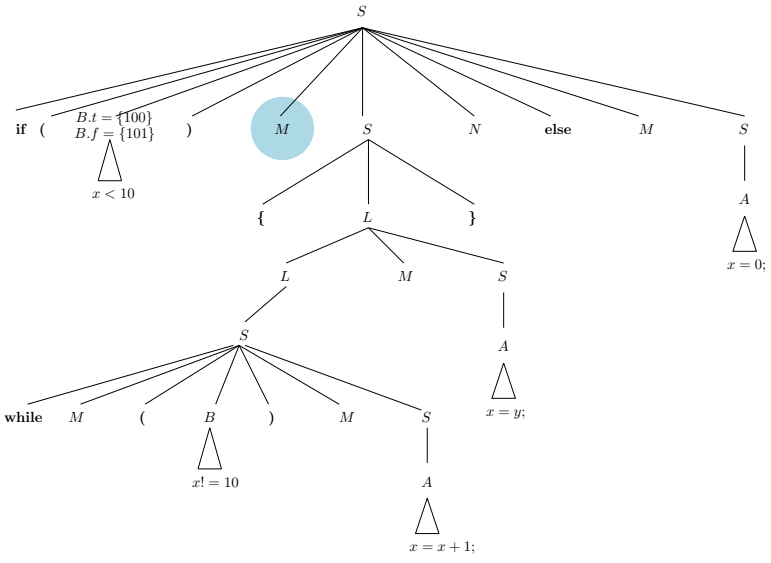


Παράδειγμα If-Else



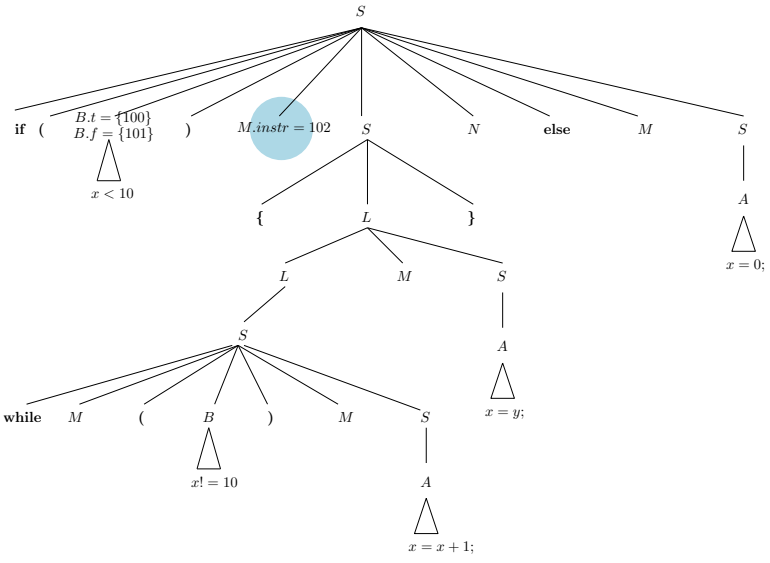

```
100: if x < 10 goto _
101: goto _
```

Παράδειγμα If-Else



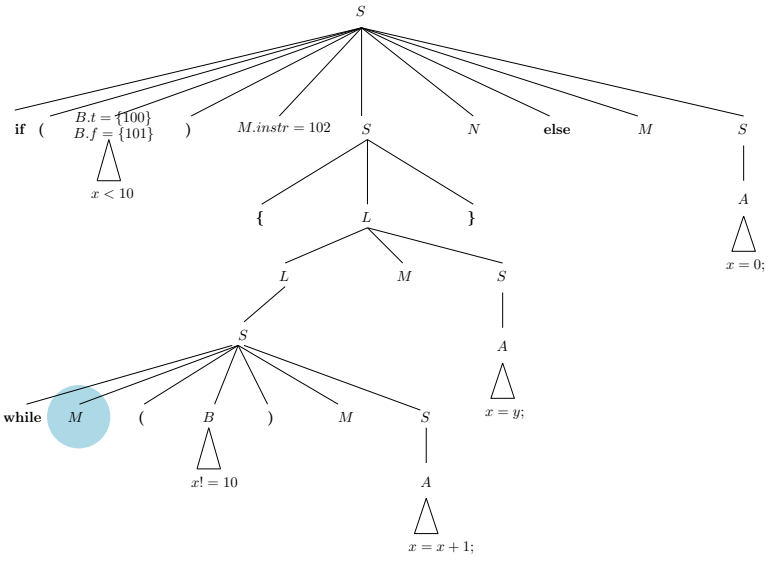

```
100: if x < 10 goto _
101: goto _
```

Παράδειγμα If-Else



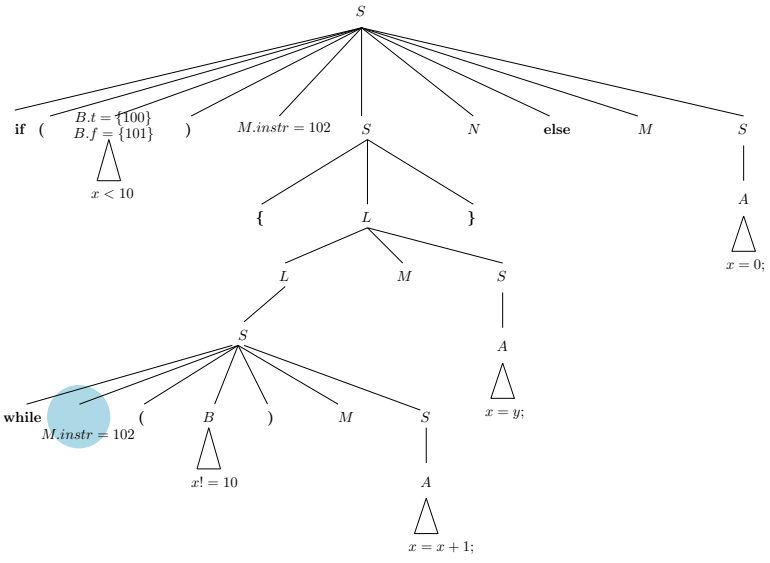

```
100: if x < 10 goto _
101: goto _
```

Παράδειγμα If-Else



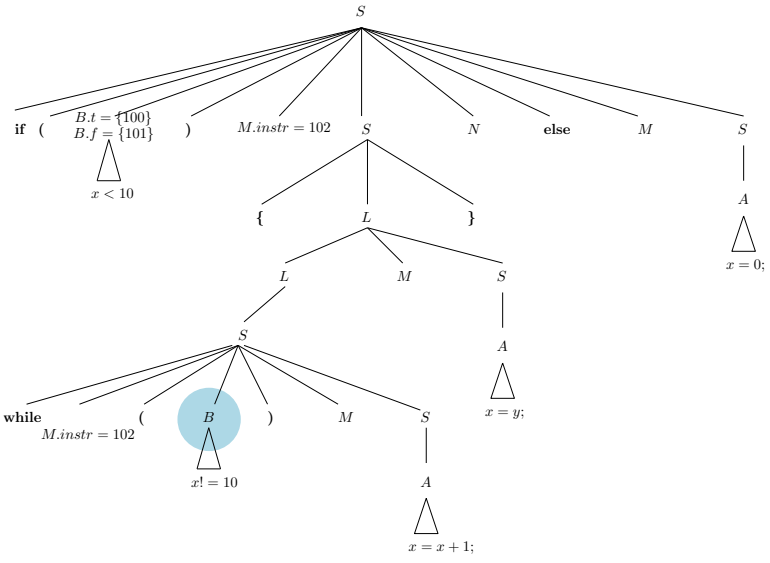

```
100: if x < 10 goto _
101: goto _
```

Παράδειγμα If-Else



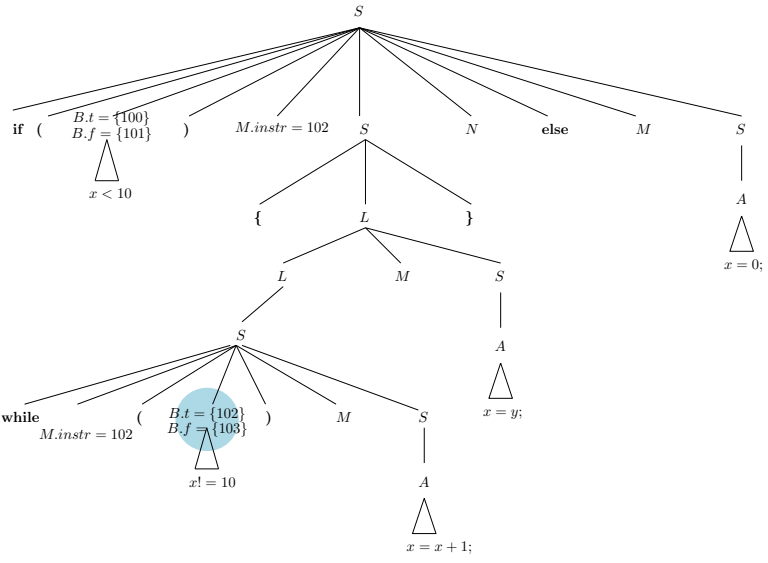

```
100: if x < 10 goto _
101: goto _
```

Παράδειγμα If-Else



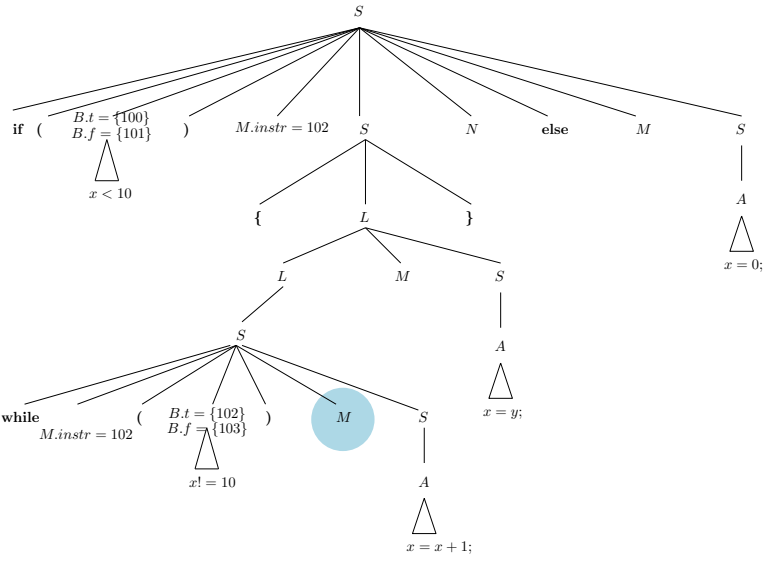

```
100: if x < 10 goto _
101: goto _
```

Παράδειγμα If-Else



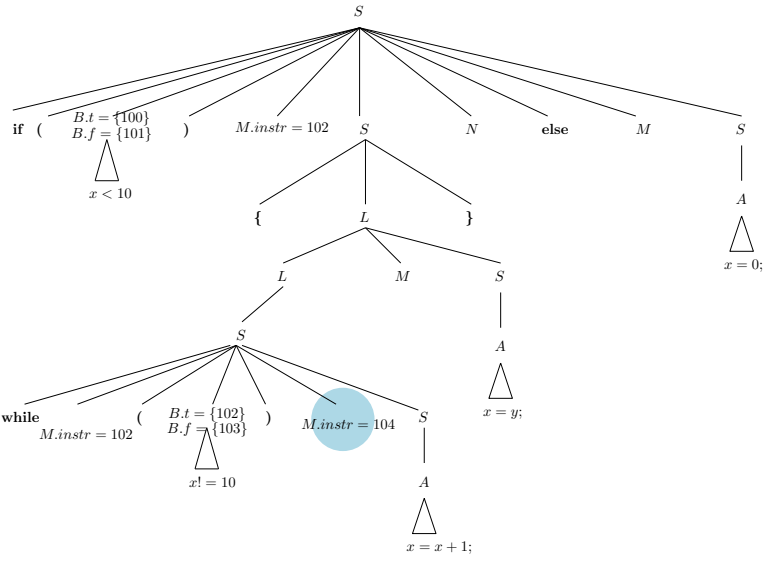

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto _
103: goto _
```

Παράδειγμα If-Else



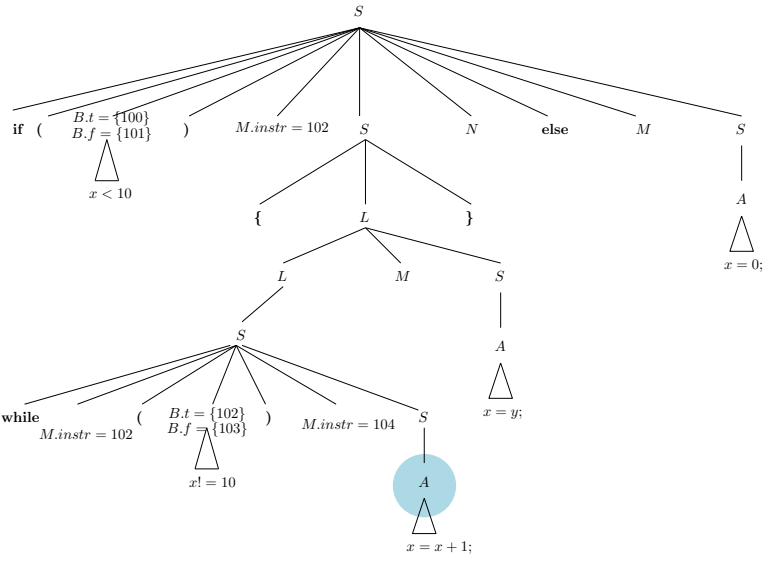

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto _
103: goto _
```

Παράδειγμα If-Else



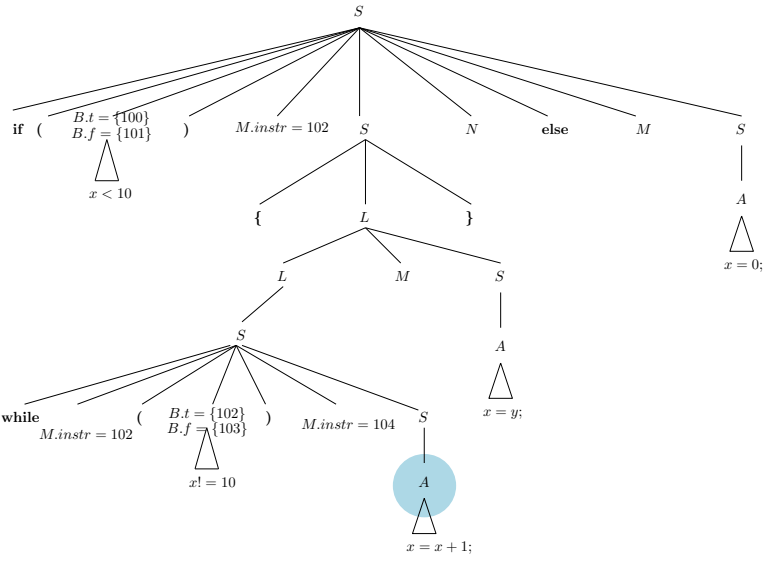

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto _
103: goto _
```

Παράδειγμα If-Else



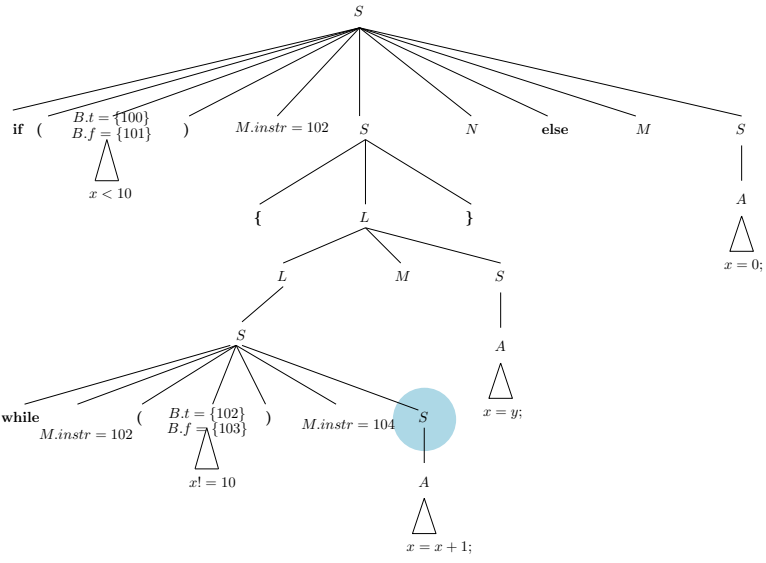

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto _
103: goto _
```

Παράδειγμα If-Else



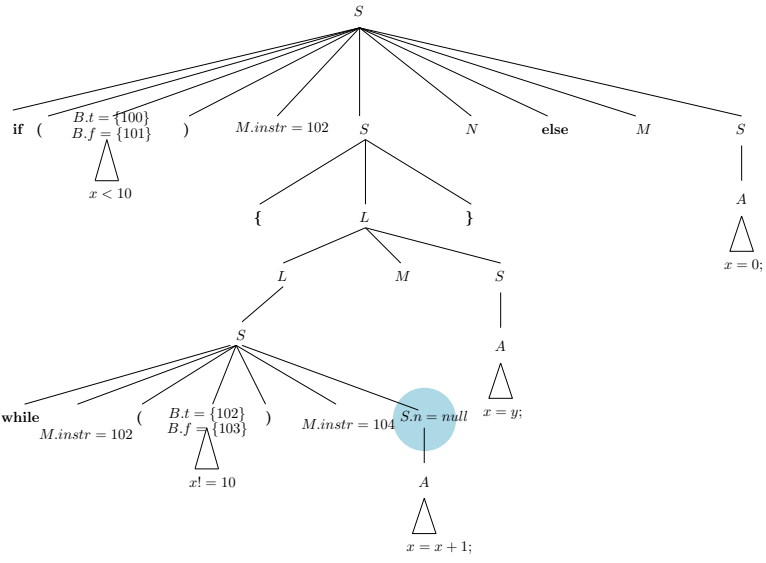

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto _
103: goto _
104: t1 = x + 1
105: x = t1
```

Παράδειγμα If-Else



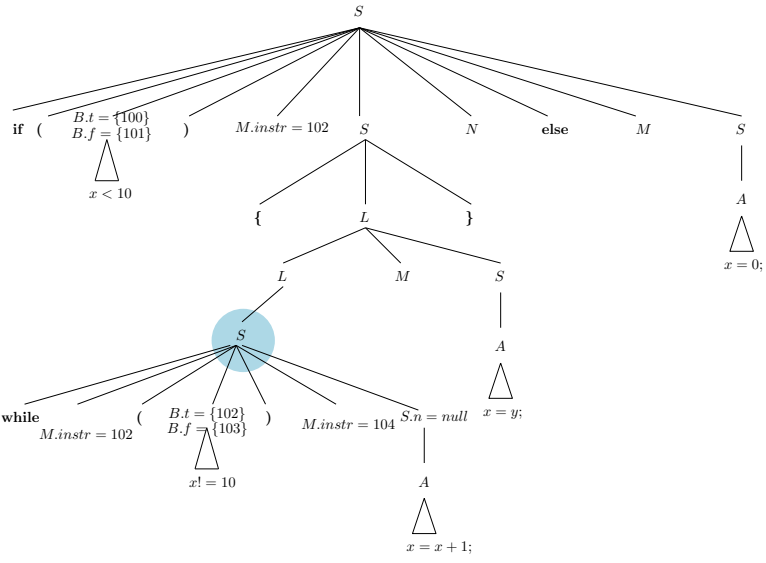

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto _
103: goto _
104: t1 = x + 1
105: x = t1
```

Παράδειγμα If-Else



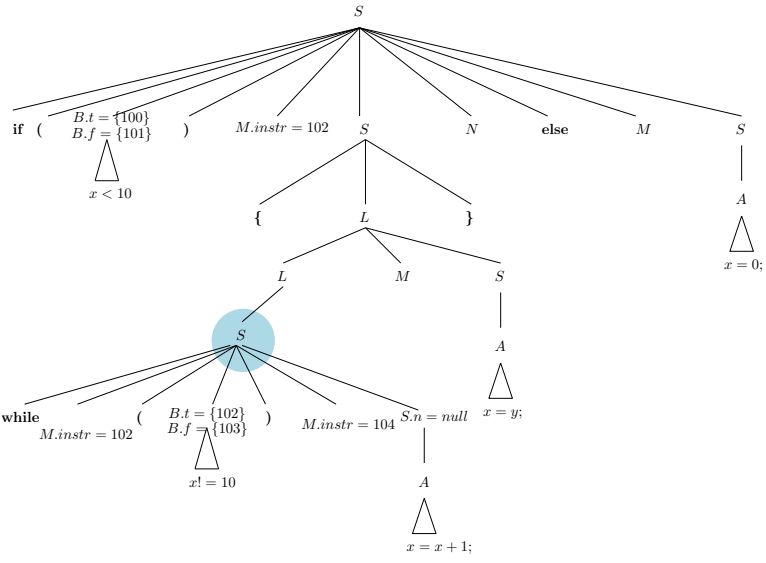

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto _
103: goto _
104: t1 = x + 1
105: x = t1
```

Παράδειγμα If-Else



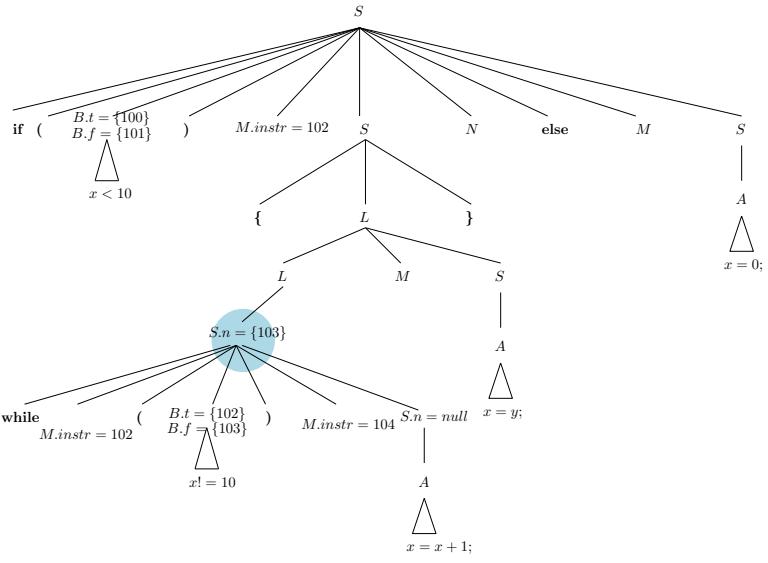

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto _
103: goto _
104: t1 = x + 1
105: x = t1
```

Παράδειγμα If-Else



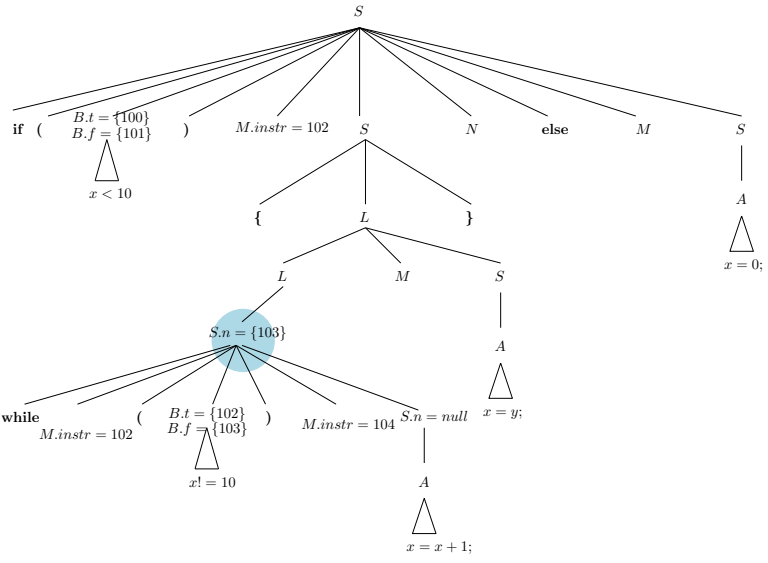

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto _
104: t1 = x + 1
105: x = t1
```

Παράδειγμα If-Else



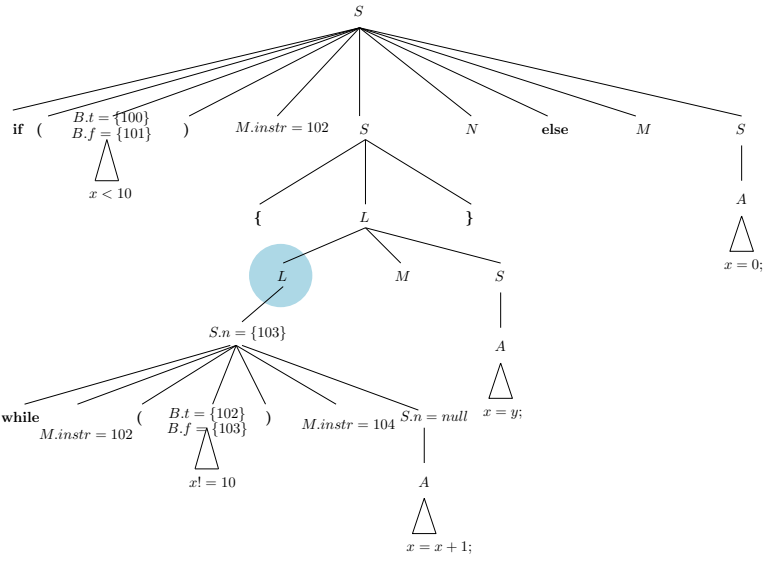

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto _
104: t1 = x + 1
105: x = t1
```

Παράδειγμα If-Else



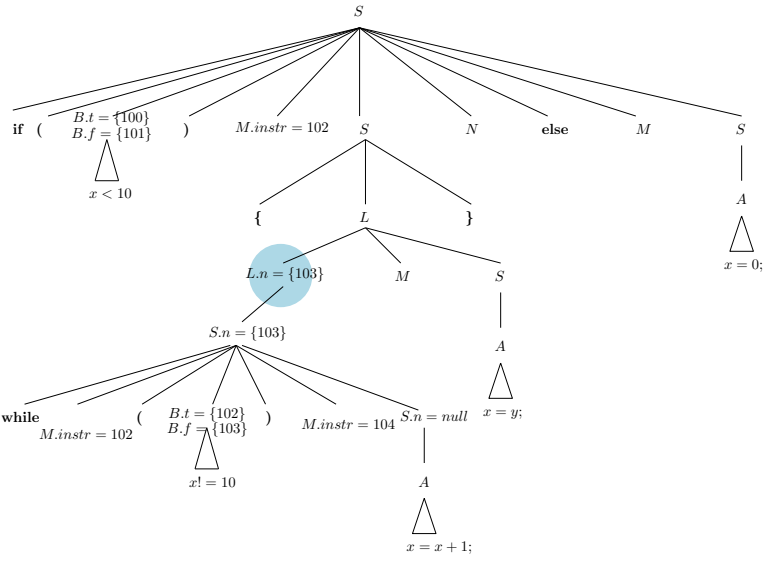

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto _
104: t1 = x + 1
105: x = t1
106: goto 102
```

Παράδειγμα If-Else



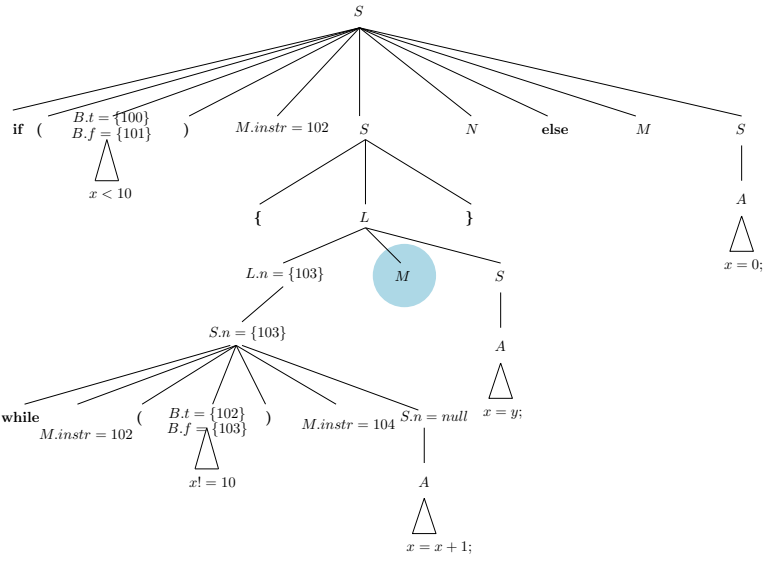

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto _
104: t1 = x + 1
105: x = t1
106: goto 102
```

Παράδειγμα If-Else



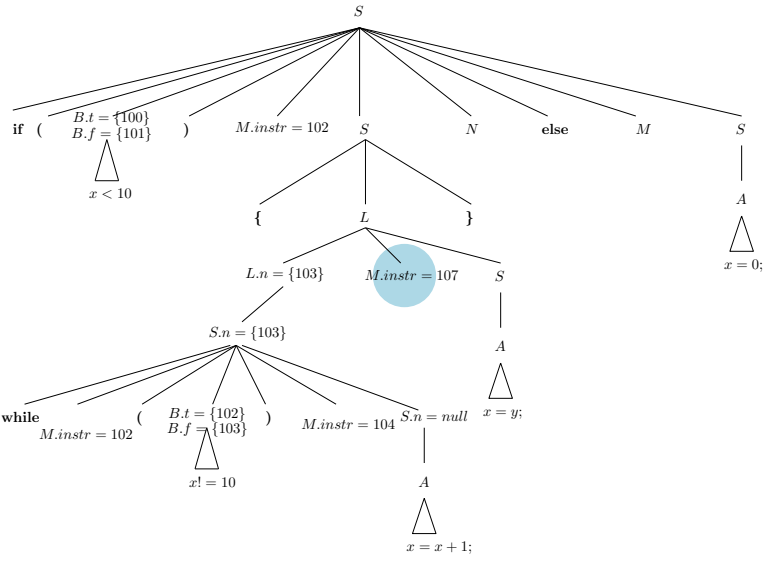

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto _
104: t1 = x + 1
105: x = t1
106: goto 102
```

Παράδειγμα If-Else



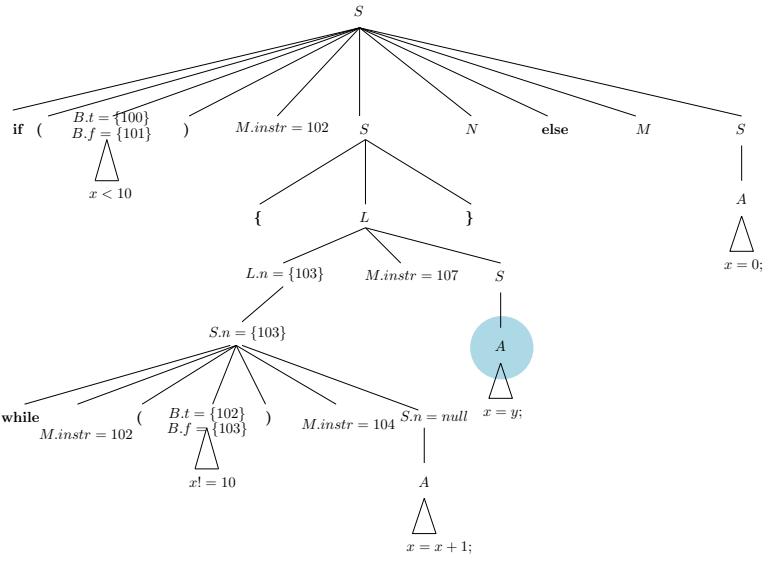

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto _
104: t1 = x + 1
105: x = t1
106: goto 102
```

Παράδειγμα If-Else



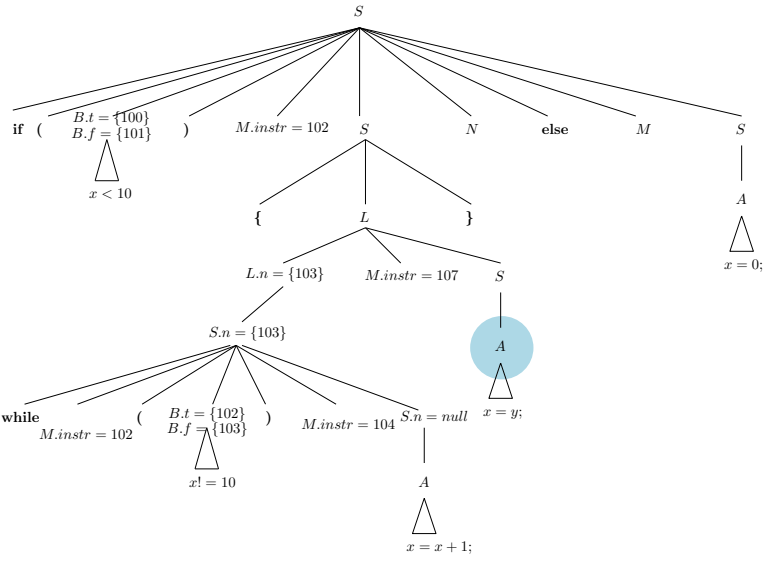

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto _
104: t1 = x + 1
105: x = t1
106: goto 102
```

Παράδειγμα If-Else



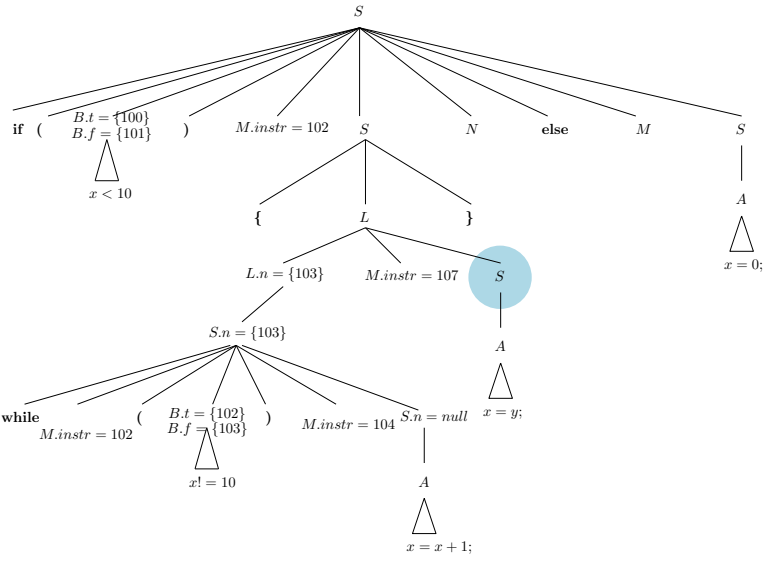

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto _
104: t1 = x + 1
105: x = t1
106: goto 102
```

Παράδειγμα If-Else



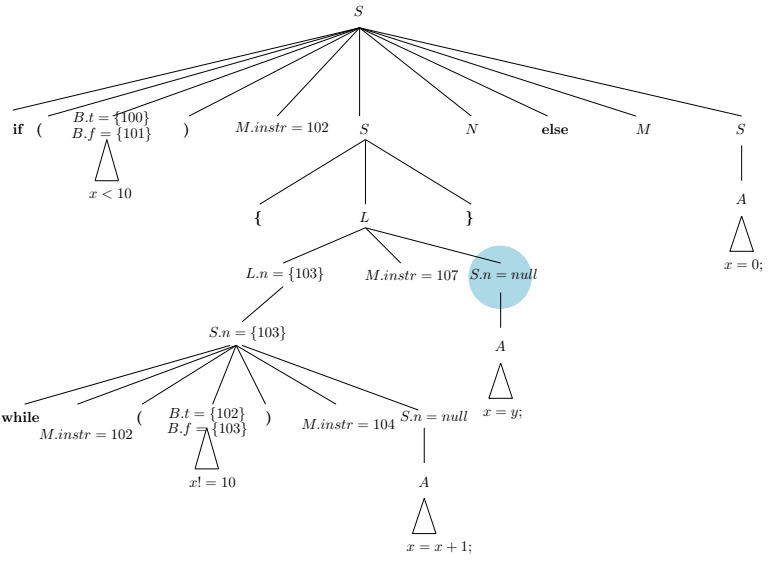

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto _
104: t1 = x + 1
105: x = t1
106: goto 102
107: t2 = y
108: x = t2
```

Παράδειγμα If-Else



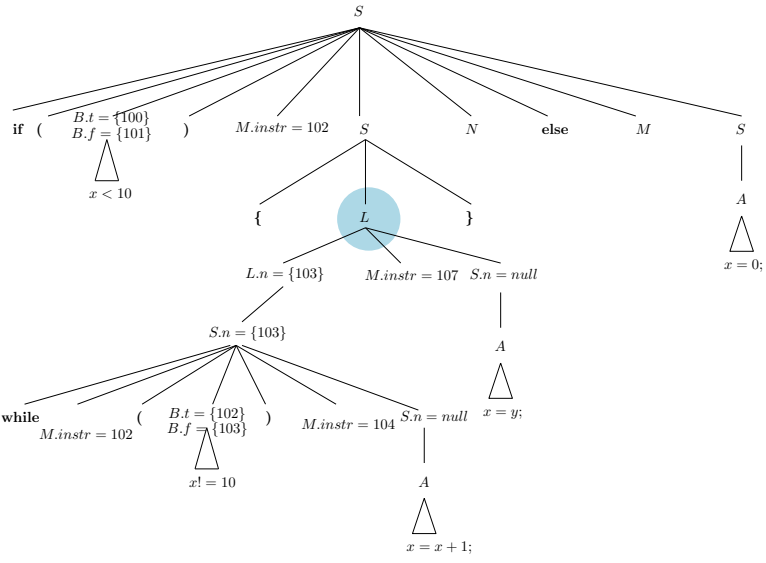

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto _
104: t1 = x + 1
105: x = t1
106: goto 102
107: t2 = y
108: x = t2
```

Παράδειγμα If-Else



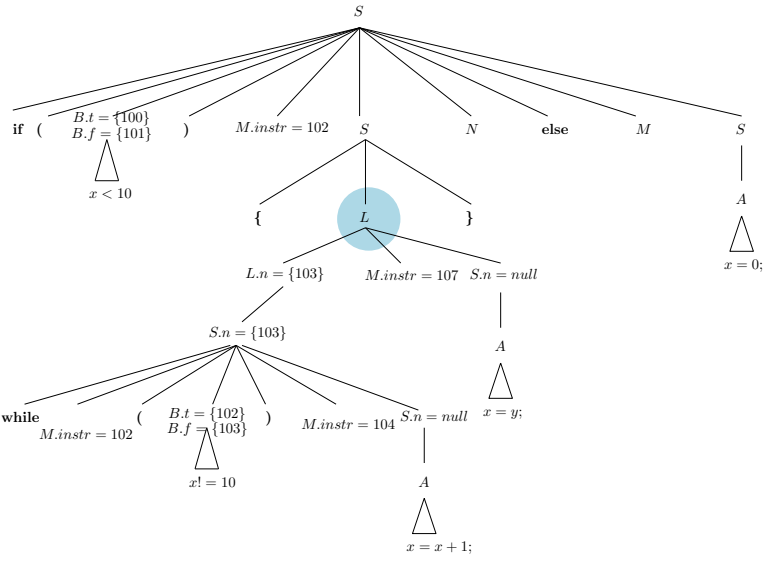

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto _
104: t1 = x + 1
105: x = t1
106: goto 102
107: t2 = y
108: x = t2
```

Παράδειγμα If-Else



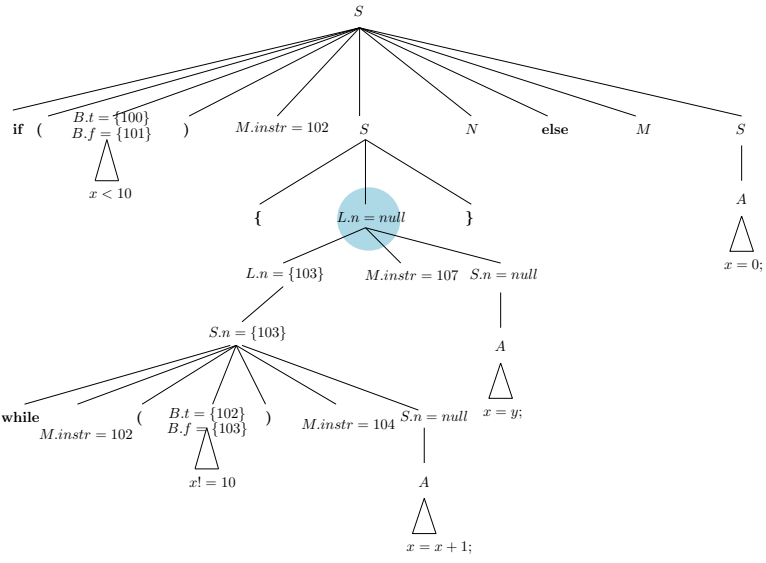

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto _
104: t1 = x + 1
105: x = t1
106: goto 102
107: t2 = y
108: x = t2
```

Παράδειγμα If-Else



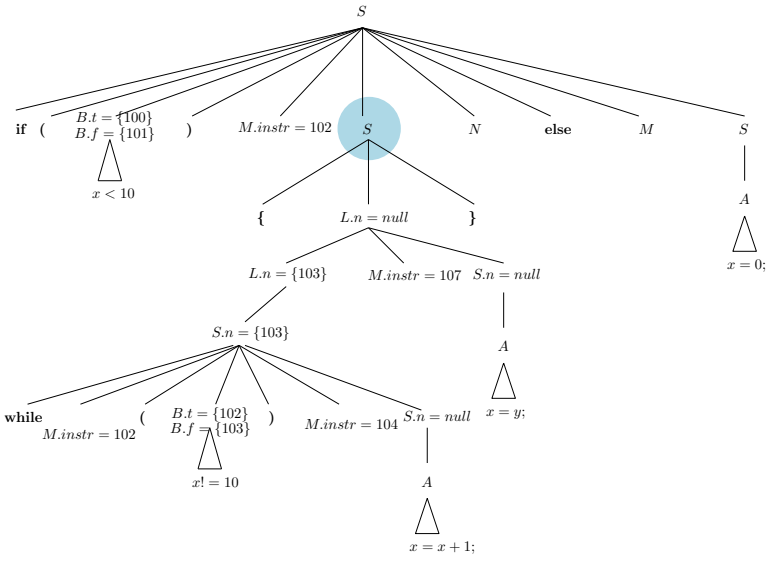

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto 107
104: t1 = x + 1
105: x = t1
106: goto 102
107: t2 = y
108: x = t2
```

Παράδειγμα If-Else



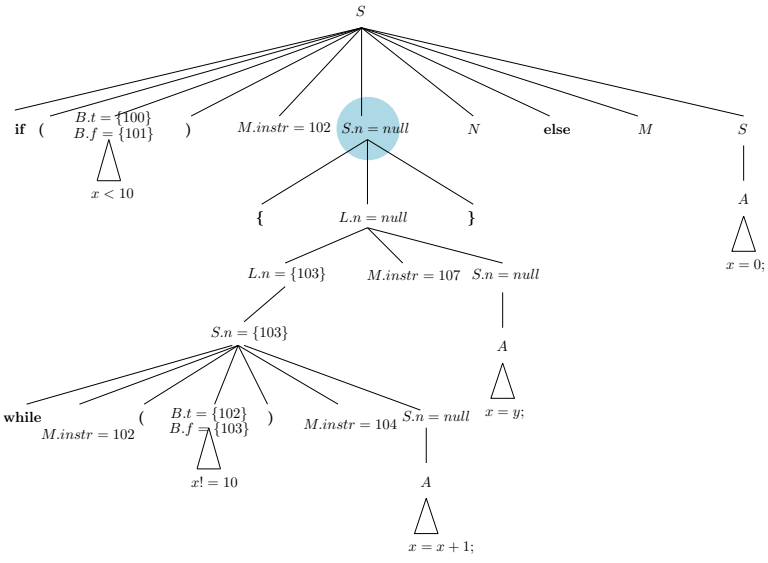

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto 107
104: t1 = x + 1
105: x = t1
106: goto 102
107: t2 = y
108: x = t2
```

Παράδειγμα If-Else



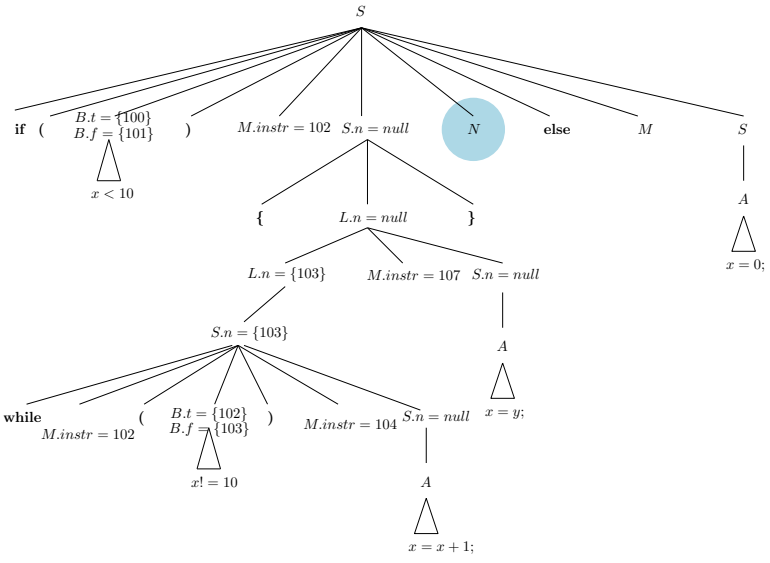

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto 107
104: t1 = x + 1
105: x = t1
106: goto 102
107: t2 = y
108: x = t2
```

Παράδειγμα If-Else



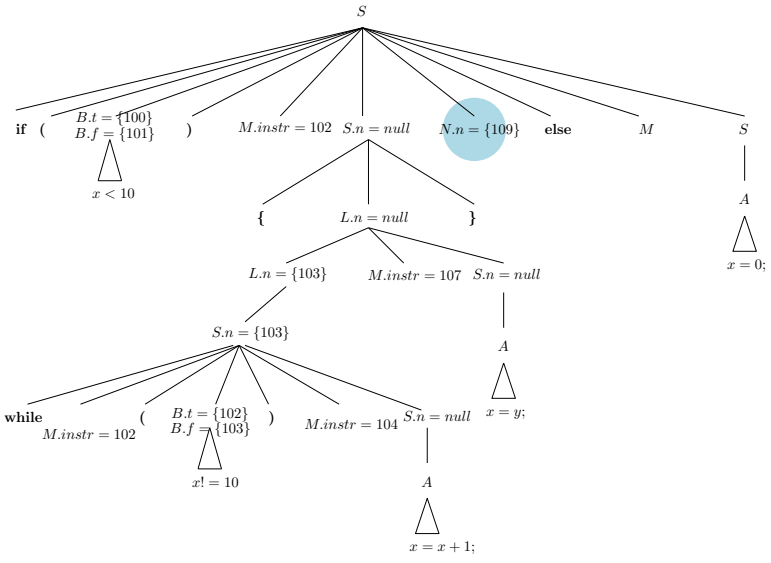

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto 107
104: t1 = x + 1
105: x = t1
106: goto 102
107: t2 = y
108: x = t2
```

Παράδειγμα If-Else



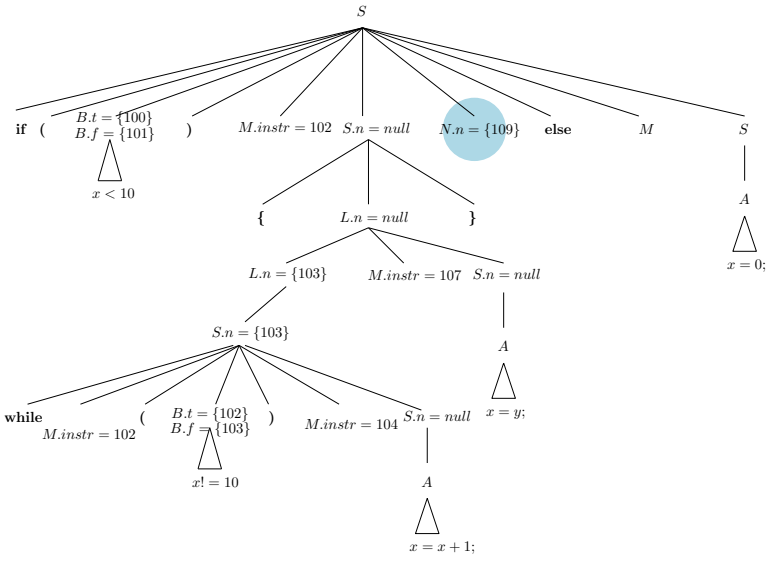

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto 107
104: t1 = x + 1
105: x = t1
106: goto 102
107: t2 = y
108: x = t2
```

Παράδειγμα If-Else



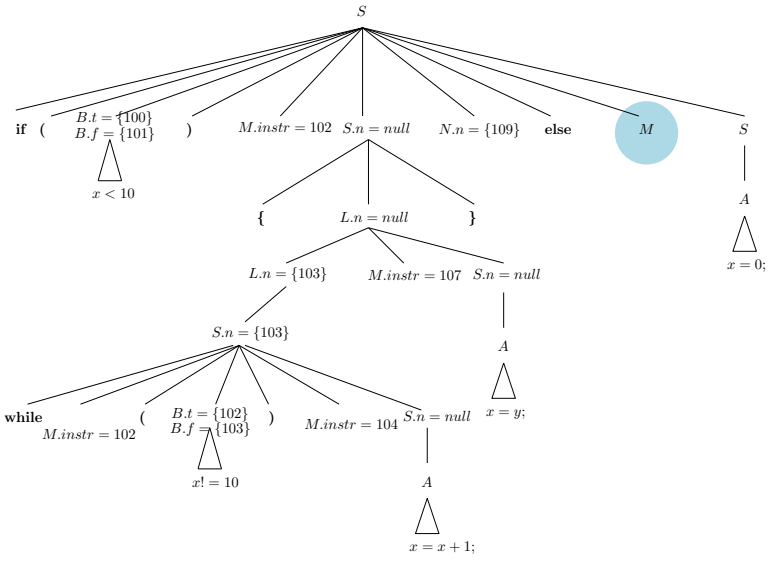

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto 107
104: t1 = x + 1
105: x = t1
106: goto 102
107: t2 = y
108: x = t2
```

Παράδειγμα If-Else



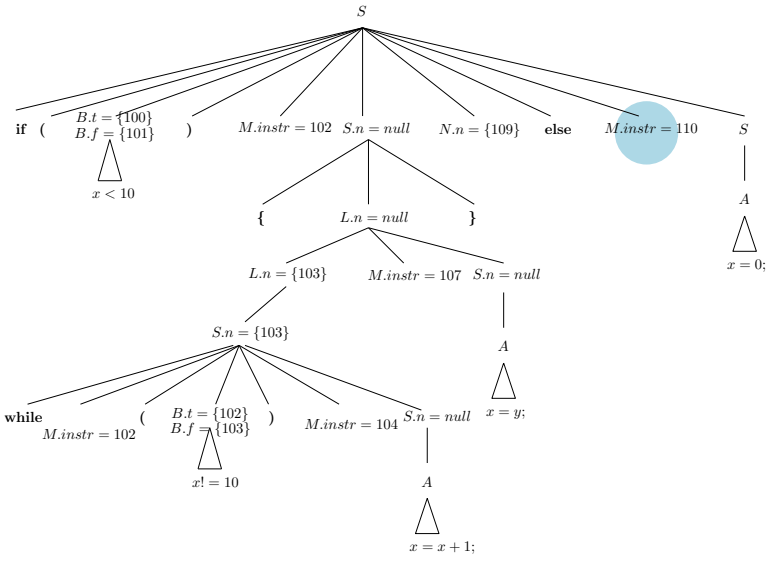

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto 107
104: t1 = x + 1
105: x = t1
106: goto 102
107: t2 = y
108: x = t2
109: goto _
```

Παράδειγμα If-Else



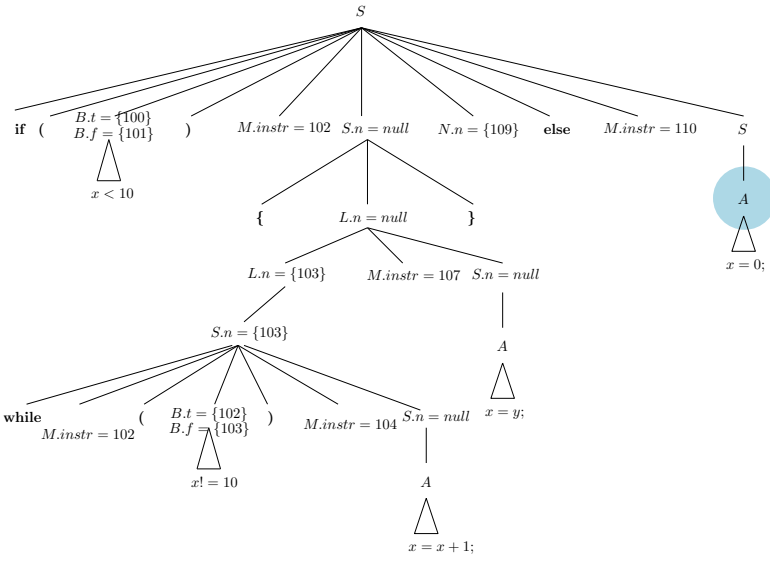

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto 107
104: t1 = x + 1
105: x = t1
106: goto 102
107: t2 = y
108: x = t2
109: goto _
```

Παράδειγμα If-Else



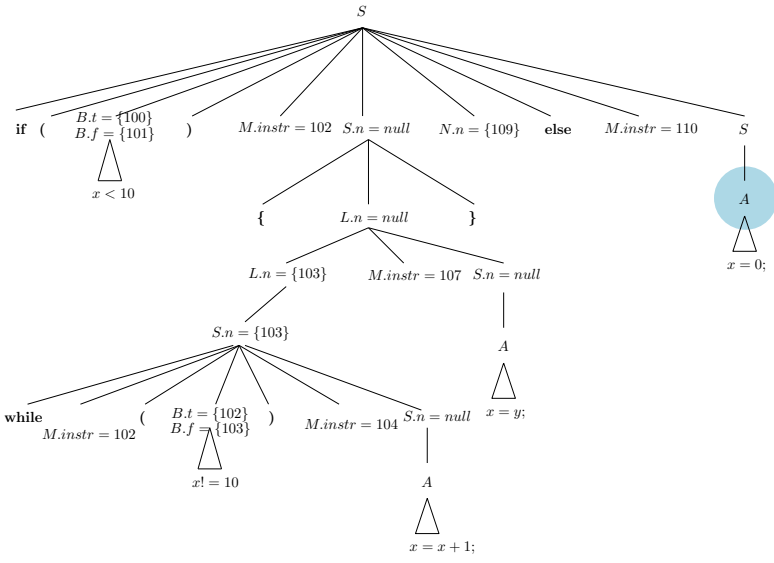

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto 107
104: t1 = x + 1
105: x = t1
106: goto 102
107: t2 = y
108: x = t2
109: goto _
```

Παράδειγμα If-Else



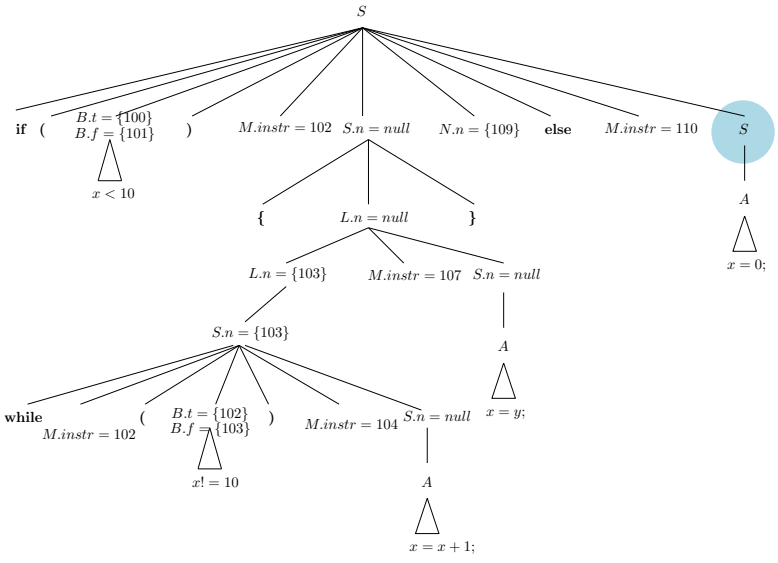

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto 107
104: t1 = x + 1
105: x = t1
106: goto 102
107: t2 = y
108: x = t2
109: goto _
```

Παράδειγμα If-Else



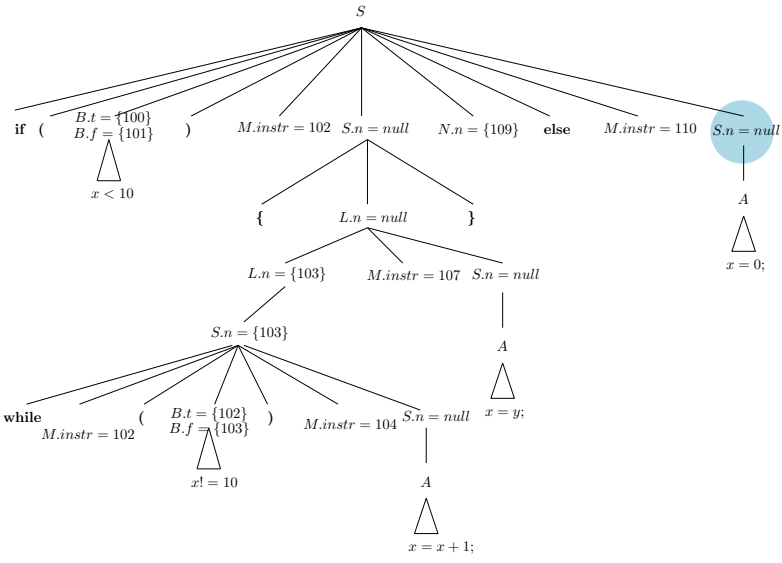

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto 107
104: t1 = x + 1
105: x = t1
106: goto 102
107: t2 = y
108: x = t2
109: goto _
110: t3 = 0
111: x = t3
```

Παράδειγμα If-Else



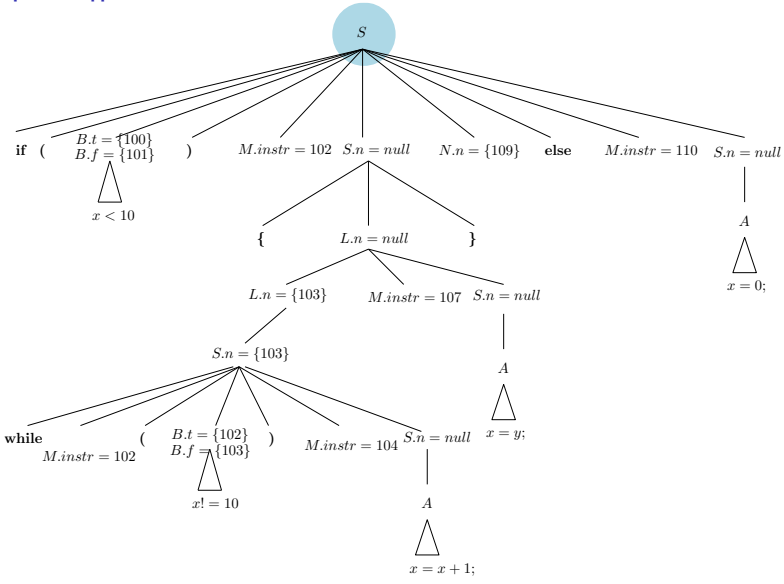

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto 107
104: t1 = x + 1
105: x = t1
106: goto 102
107: t2 = y
108: x = t2
109: goto _
110: t3 = 0
111: x = t3
```

Παράδειγμα If-Else



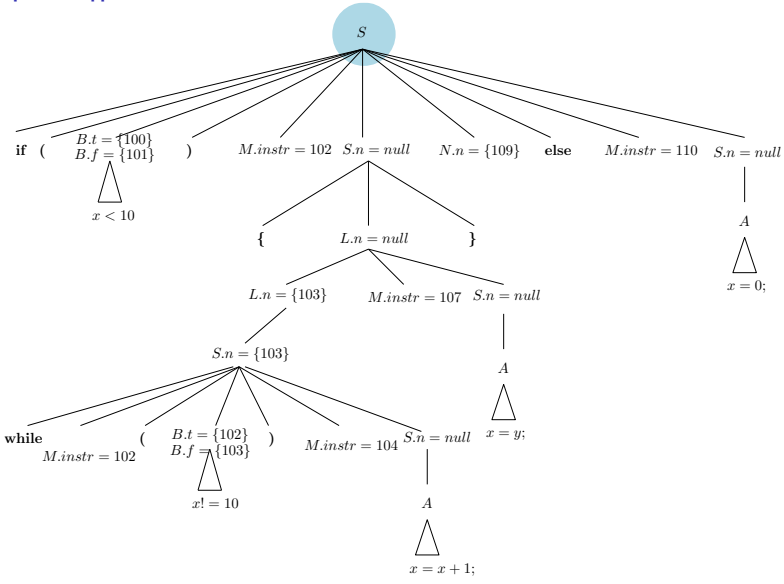

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto 107
104: t1 = x + 1
105: x = t1
106: goto 102
107: t2 = y
108: x = t2
109: goto _
110: t3 = 0
111: x = t3
```

Παράδειγμα If-Else



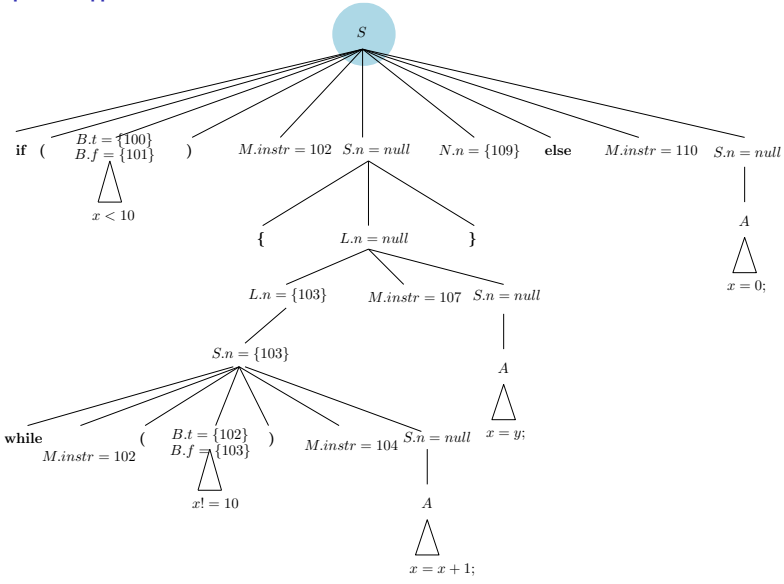

```
100: if x < 10 goto _
101: goto _
102: if x != 10 goto 104
103: goto 107
104: t1 = x + 1
105: x = t1
106: goto 102
107: t2 = y
108: x = t2
109: goto _
110: t3 = 0
111: x = t3
```

Παράδειγμα If-Else



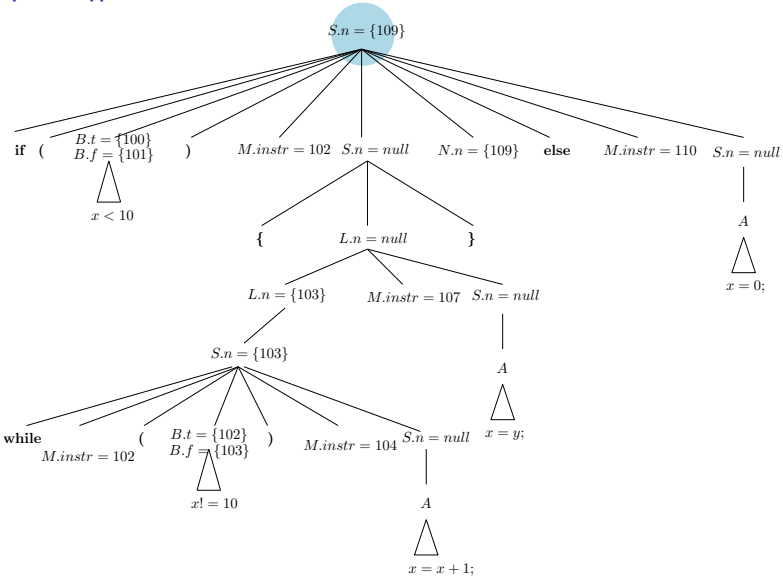

```
100: if x < 10 goto 102
101: goto _
102: if x != 10 goto 104
103: goto 107
104: t1 = x + 1
105: x = t1
106: goto 102
107: t2 = y
108: x = t2
109: goto _
110: t3 = 0
111: x = t3
```

Παράδειγμα If-Else




```
100: if x < 10 goto 102
101: goto 110
102: if x != 10 goto 104
103: goto 107
104: t1 = x + 1
105: x = t1
106: goto 102
107: t2 = y
108: x = t2
109: goto _
110: t3 = 0
111: x = t3
```

Παράδειγμα If-Else




```
100: if x < 10 goto 102
101: goto 110
102: if x != 10 goto 104
103: goto 107
104: t1 = x + 1
105: x = t1
106: goto 102
107: t2 = y
108: x = t2
109: goto _
110: t3 = 0
111: x = t3
```

Break και Continue

```
for( ; ; readch() ) {
    if ( peek = ' ' || peek == '\t' )
        continue;
    else if ( peek == '\n' )
        line = line + 1;
    else
        break;
}
```

Εαν S είναι η επανάληψη που περιέχει ένα break, τότε το break στέλνει τον κώδικα στην επόμενη γραμμή μετά το S.

Για την παραγωγή ενδιάμεσου κώδικα για το break πρέπει

1. να διατηρούμε πρόσβαση στον βρόγχο S (π.χ με ένα ειδικό entry στο symbol-table που το ανανεώνουμε κάθε φορά που ξεκινά ή τελειώνει ένας βρόγχος)
2. να γράψουμε ένα κενό 'goto _'
3. να προσθέσουμε την γραμμή του goto στο S.nextlist

Το continue μπορεί να υλοποιηθεί με τον ίδιο τρόπο, μόνο ο στόχος αλλάζει.

Συναρτήσεις

Οι παρακάτω προσθήκες στην γραμματική επιτρέπουν την χρήση συναρτήσεων:

$$
\begin{array}{lcl}
D & \rightarrow & T\ \textbf{id}\ (\ FP\ )\ \{\ \mathrm{S}\ \} \\
S & \rightarrow & \textbf{return}\ E\ ; \\
E & \rightarrow & \textbf{id}\ (\ AP\ ) \\
AP & \rightarrow & E \\
 & \rightarrow & AP\ ,\ E
\end{array}
$$

D	Declaration
T	Type
FP	Format Parameters
E	Expression
S	Statement
AP	Actual Parameters

Κλήση Συναρτήσεων

Ο κώδικας

```
n = f(a[i], b[i+1], c);
```

μπορεί να μεταφραστεί σε

```
t1 = i * 4
t2 = a[t1]
t3 = i + 1
t4 = t3 * 4
t5 = b[t4]
param t2
param t5
param c
t6 = call f, 3
n = t6
```

Για να εμφανίζονται όλα τα param συνεχόμενα (όπως στο παραπάνω παράδειγμα) μπορούμε να αποθηκεύουμε τα *E.addr* σε μια ουρά, και στο τέλος να παράγουμε όλες τις παραμέτρους συνεχόμενες.

Μέσω του πίνακα συμβόλων πρέπει να μπορούμε να βρούμε τον αριθμό των παραμέτρων που περιμένει η συνάρτηση.