Μεταγλωττιστές

Σημασιολογική Ανάλυση

Δημήτρης Μιχαήλ

Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο

Σημασιολογική Ανάλυση

Υπάρχουν πολλές ιδιότητες που δεν μπορούν να περιγραφούν με γραμματικές χωρίς συμφραζόμενα.

π.χ θέλουμε να απαντάμε στα παρακάτω ερωτήματα

1. Η μεταβλητή **x** έχει δηλωθεί πριν την χρήση της;

Σημασιολογική Ανάλυση

Υπάρχουν πολλές ιδιότητες που δεν μπορούν να περιγραφούν με γραμματικές χωρίς συμφραζόμενα.

π.χ θέλουμε να απαντάμε στα παρακάτω ερωτήματα

1. Η μεταβλητή **x** έχει δηλωθεί πριν την χρήση της;
2. Όταν αναφερόμαστε στο προσδιοριστικό **x** εννοούμε συνάρτηση, πίνακα ή μεταβλητή;

Σημασιολογική Ανάλυση

Υπάρχουν πολλές ιδιότητες που δεν μπορούν να περιγραφούν με γραμματικές χωρίς συμφραζόμενα.

π.χ θέλουμε να απαντάμε στα παρακάτω ερωτήματα

1. Η μεταβλητή **x** έχει δηλωθεί πριν την χρήση της;
2. Όταν αναφερόμαστε στο προσδιοριστικό **x** εννοούμε συνάρτηση, πίνακα ή μεταβλητή;
3. Όταν αναφερόμαστε στο προσδιοριστικό **x** και υπάρχουν πολλές μεταβλητές με το ίδιο όνομα, σε ποια αναφερόμαστε;

Σημασιολογική Ανάλυση

Υπάρχουν πολλές ιδιότητες που δεν μπορούν να περιγραφούν με γραμματικές χωρίς συμφραζόμενα.

π.χ θέλουμε να απαντάμε στα παρακάτω ερωτήματα

1. Η μεταβλητή **x** έχει δηλωθεί πριν την χρήση της;
2. Όταν αναφερόμαστε στο προσδιοριστικό **x** εννοούμε συνάρτηση, πίνακα ή μεταβλητή;
3. Όταν αναφερόμαστε στο προσδιοριστικό **x** και υπάρχουν πολλές μεταβλητές με το ίδιο όνομα, σε ποια αναφερόμαστε;
4. Όταν κάνουμε ανάθεση στην μεταβλητή **x**, είναι οι τύποι συμβατοί;

Σημασιολογική Ανάλυση

Υπάρχουν πολλές ιδιότητες που δεν μπορούν να περιγραφούν με γραμματικές χωρίς συμφραζόμενα.

π.χ θέλουμε να απαντάμε στα παρακάτω ερωτήματα

1. Η μεταβλητή **x** έχει δηλωθεί πριν την χρήση της;
2. Όταν αναφερόμαστε στο προσδιοριστικό **x** εννοούμε συνάρτηση, πίνακα ή μεταβλητή;
3. Όταν αναφερόμαστε στο προσδιοριστικό **x** και υπάρχουν πολλές μεταβλητές με το ίδιο όνομα, σε ποια αναφερόμαστε;
4. Όταν κάνουμε ανάθεση στην μεταβλητή **x**, είναι οι τύποι συμβατοί;
5. Πόσες παραμέτρους δέχεται μια συνάρτηση με όνομα **foo**;
6. κ.τ.λ

Σημασιολογική Ανάλυση

Τέτοιου είδους έλεγχοι γίνονται σε έναν μεταγλωττιστή κατά την διάρκεια της σημασιολογικής ανάλυσης.

(i) Η σημασιολογική ανάλυση αφορά κυρίως τον **στατικό έλεγχο** (*static checking*) του προγράμματος.

(ii) Μπορεί να γίνει είτε ταυτόχρονα με την συντακτική ανάλυση είτε σε κάποιο πέρασμα στην συνέχεια.

Στατικός Έλεγχος
Static Checking

Γίνεται κατά την μεταγλώττιση και ελέγχει ιδιότητες που δεν καλύπτονται από την γραμματική όπως:

Συντακτικός Έλεγχος

- Κάθε αναγνωριστικό να έχει δηλωθεί πριν χρησιμοποιηθεί.
- Κάθε αναγνωριστικό να έχει δηλωθεί μόνο μια φορά σε κάθε εμβέλεια.
- Ένα **continue** πρέπει να είναι υποχρεωτικά μέσα σε μια επανάληψη.
- Ένα **break** πρέπει να είναι υποχρεωτικά μέσα σε μια επανάληψη ή σε ένα **switch**.

Στατικός Έλεγχος
Static Checking

Γίνεται κατά την μεταγλώττιση και ελέγχει ιδιότητες που δεν καλύπτονται από την γραμματική όπως:

Συντακτικός Έλεγχος

- Κάθε αναγνωριστικό να έχει δηλωθεί πριν χρησιμοποιηθεί.
- Κάθε αναγνωριστικό να έχει δηλωθεί μόνο μια φορά σε κάθε εμβέλεια.
- Ένα **continue** πρέπει να είναι υποχρεωτικά μέσα σε μια επανάληψη.
- Ένα **break** πρέπει να είναι υποχρεωτικά μέσα σε μια επανάληψη ή σε ένα **switch**.

Έλεγχος Τύπων

Οι κανόνες τύπων μιας γλώσσας φροντίζουν ένας τελεστής ή μια συνάρτηση να χρησιμοποιούνται με σωστό αριθμό και τύπο παραμέτρων.

- Επίσης γίνεται οποιαδήποτε μετατροπή τύπων είναι απαραίτητη.

Δυναμικός Έλεγχος

Υπάρχουν διάφοροι έλεγχοι που μπορεί να γίνουν μόνο κατά την διάρκεια της εκτέλεσης του προγράμματος.

π.χ

έλεγχος ορίων σε πίνακες

```
int foo() {
    // ...
}

int main() {
    int array[100];
    int x = foo();

    array[x] = 1;
}
```

Στην γλώσσα *C* δεν γίνεται τέτοιος έλεγχος, στην γλώσσα Java γίνεται.

Πίνακας Συμβόλων
Symbol Table

Δομή δεδομένων που διατηρεί πληροφορίες για το πρόγραμμα

- χτίζεται σιγά σιγά κατά την διάρκεια της ανάλυσης του προγράμματος
- βοηθάει για να γίνουν οι σημασιολογικοί έλεγχοι
- βοηθάει στην τελική παραγωγή κώδικα

Περιέχει πληροφορίες για προσδιοριστικά, τύπους, συναρτήσεις, κλάσεις, κ.τ.λ

Πίνακας Συμβόλων
Symbol Table

Τα διάφορα προσδιοριστικά (identifiers) που χρησιμοποιεί ο προγραμματιστής έχουν κάποιες ιδιότητες.

- Αυτές οι ιδιότητες περιγράφονται από μια εγγραφή στον πίνακα συμβόλων.
- Ένα στοιχείο του πίνακα συμβόλων περιέχει πληροφορίες για ένα προσδιοριστικό (identifier) όπως
  - το όνομα (lexeme)
  - τύπος (μεταβλητή, συνάρτηση, κ.τ.λ)
  - θέση αποθήκευσης
  - τιμή (άμα είναι σταθερά)
  - κ.τ.λ

Πίνακας Συμβόλων
Symbol Table

Ο πίνακας συμβόλων συνήθως δημιουργείται κατά την διάρκεια της ανάλυσης του προγράμματος και εμπλουτίζεται καθώς αποκτούμε περισσότερες γνώσεις για το πρόγραμμα.

1. Παραμένει διαρκώς στην μνήμη και πρέπει να επιτρέπει την ταχεία αναζήτηση συμβόλων.
2. Υλοποιείται συνήθως με κατακερματισμό (hashing).

Χρησιμοποιείται στην συνέχεια από την σημασιολογική ανάλυση, αλλά και από την φάση δημιουργίας κώδικα.

Πίνακας Συμβόλων

Symbol Table

Δεν είναι πάντα σαφές σε ποια φάση της μεταγλώττισης μπορούμε να προσθέσουμε πληροφορία στον πίνακα συμβόλων.

Σε μερικές περιπτώσεις μπορεί να προσθέσουμε ένα όνομα στον πίνακα συμβόλων ακόμη και στην φάση της λεκτικής ανάλυσης.

Άλλες φορές όπως π.χ στην γλώσσα C

```
int main() {
    int x;
    struct x {
        float y, z;
    };
}
```

πρέπει να περιμένουμε μέχρι να ανακαλυφθεί ο ρόλος ενός ονόματος μέχρι να εισαχθεί στον πίνακα συμβόλων.

Στην παραπάνω περίπτωση πρέπει να προσθέσουμε δύο στοιχεία με όνομα `x` στον πίνακα συμβόλων.

Πολλαπλές Δηλώσεις Προσδιοριστικών

Στις σύγχρονες γλώσσες προγραμματισμού επιτρέπεται η χρήση των ίδιων ονομάτων για διαφορετικά προσδιοριστικά.

Αυτό επιτυγχάνεται με την έννοια της **εμβέλειας (scope)**.

```
{    /* outer scope */
     int x, y;

     x = 1;

     {    /* inner scope */
          int x;

          x = 2;
          y = 2;
     }
}
```

Πολλαπλές Δηλώσεις Προσδιοριστικών

Για να υποστηρίξουμε την έννοια της εμβέλειας, θα χρησιμοποιήσουμε έναν πίνακα συμβόλων ανά εμβέλεια.

- Ένα μπλόκ κώδικα (program block) με δηλώσεις μεταβλητών θα έχει τον δικό του πίνακα συμβόλων, με ένα στοιχείο για κάθε δήλωση.

Πολλαπλές Δηλώσεις Προσδιοριστικών

Για να υποστηρίξουμε την έννοια της εμβέλειας, θα χρησιμοποιήσουμε έναν πίνακα συμβόλων ανά εμβέλεια.

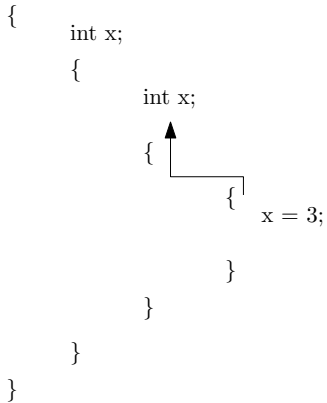
- Ένα μπλόκ κώδικα (program block) με δηλώσεις μεταβλητών θα έχει τον δικό του πίνακα συμβόλων, με ένα στοιχείο για κάθε δήλωση.

- Αυτή η τεχνική λειτουργεί και για άλλες κατασκευές όπως οι κλάσεις (class) που υπάρχουν σε αντικειμενοστραφής γλώσσες προγραμματισμού.

  Κάθε κλάση έχει τον δικό της πίνακα συμβόλων με στοιχεία που αφορούν τις μεθόδους και τις μεταβλητές που ανήκουν στην κλάση.

Κανόνας του Κοντυνότερου Μπλόκ

Για να βρούμε την εμβέλεια που ανήκει μια μεταβλητή αρκεί να ψάξουμε από τα μέσα προς τα έξω.



Στην περίπτωση αυτή η αλυσίδα των πινάκων συμβόλων είναι μια στοίβα.

Παράδειγμα

Καθώς διαβάζουμε την εντολή printf() στο παρακάτω παράδειγμα ο πίνακας συμβόλων έχει την εξής μορφή.

```c
#include <stdio.h>

int p = 17;

void foo() {
    /* ... */
}

int main() {
    int j;

    for( int i = 0; i < p; i++ ) {
        printf("%d %d\n",i,j++);
    }
}
```

S_2	i	variable	int	
S_1	j	variable	int	
S_0	p	variable	int	
	foo	function	void	no params
	main	function	int	no params

Παράδειγμα Υλοποίησης Πίνακα Συμβόλων

Java με Generics

```
import java.util.*;

public class SymbolTable<E> {

    private Map<String, E> table = new HashMap<String, E>();
    protected SymbolTable<E> parent;

    public SymbolTable() {
        parent = null;
    }

    public SymbolTable(SymbolTable<E> parent) {
        this.parent = parent;
    }

    public void put(String s, E symbol) {
        table.put(s, symbol);
    }
```

Παράδειγμα Υλοποίησης Πίνακα Συμβόλων

Java με Generics

```
    public E get(String s) {
        for(SymbolTable<E> e = this; e != null; e = e.parent) {
            E found = e.table.get(s);
            if ( found != null )
                return found;
        }
        return null;
    }

    public E getOnlyInTop(String s) {
        return table.get(s);
    }

}
```

Χρήση Πίνακα Συμβόλων

Σε περίπτωση που ο μεταγλωττιστής δουλεύει με ένα μόνο πέρασμα, μπορούμε μέσω των σημασιολογικών ρουτινών να κατασκευάσουμε τον πίνακα συμβόλων.

- Κατά την δήλωση μιας μεταβλητής βάζουμε πληροφορίες για την x στον πίνακα συμβόλων.
- Αντίστοιχα κατά την δήλωση μιας συνάρτησης προσθέτουμε μια εγγραφή στον πίνακα συμβόλων μαζί με πληροφορίες για τον τύπο επιστροφής της συνάρτησης, τον αριθμό των παραμέτρων, τους τύπους των παραμέτρων, κ.τ.λ.

Χρήση Πίνακα Συμβόλων

Σε περίπτωση που ο μεταγλωττιστής δουλεύει με ένα μόνο πέρασμα, μπορούμε μέσω των σημασιολογικών ρουτινών να κατασκευάσουμε τον πίνακα συμβόλων.

- Κατά την δήλωση μιας μεταβλητής βάζουμε πληροφορίες για την *x* στον πίνακα συμβόλων.
- Αντίστοιχα κατά την δήλωση μιας συνάρτησης προσθέτουμε μια εγγραφή στον πίνακα συμβόλων μαζί με πληροφορίες για τον τύπο επιστροφής της συνάρτησης, τον αριθμό των παραμέτρων, τους τύπους των παραμέτρων, κ.τ.λ.

- Όταν χρησιμοποιείται ένα προσδιοριστικό ψάχνουμε για πληροφορίες στον πίνακα συμβόλων.

1. εαν δεν βρεθεί καλούμε την απαραίτητη ρουτίνα λάθους
2. αν βρεθεί
   - χρησιμοποιούμε την εκάστωτε πληροφορία
   - εμπλουτίζουμε την εγγραφή με επιπλέον πληροφορίες που μπορεί να έχουμε

Χρήση Πίνακα Συμβόλων

Παράδειγμα

program	→	*block*	{	`top = null;` }
block	→	'{'	{	`saved.push(top);` `top = new SymbolTable<Symbol>(top);` }
		decls		
		stmts		
		'}'	{	`top=saved.pop();` }
decls	→	*decls decl*		
	|	ϵ		
decl	→	**int id ;**	{	`s = new Symbol();` `s.type=int;` `top.put(id.lexeme,s);` }
stmts	→	*stmts stmt*		
	|	ϵ		
stmt	→	*block*		
	|	**id = LIT_INT ;**	{	`if (top.get(id.lexeme)==null)` `error("undeclared");` }

Πίνακας Συμβόλων και Αφηρημένο Συντακτικό Δέντρο

Εαν ο μεταγλωττιστής κατασκευάζει ένα αφηρημένο συντακτικό δέντρο, μπορούμε να κάνουμε διάσχιση του δέντρου και να κατασκευάσουμε τον πίνακα συμβόλων.

- Στην συνέχεια σε ένα δεύτερο πέρασμα (δεύτερη διάσχιση) μπορούμε να κάνουμε τους απαραίτητους ελέγχους.
- Τα δύο περάσματα είναι ευκολότερη λύση σε γλώσσες όπως η Java όπου επιτρέπονται τα ”forward references”.
- Αυτή είναι πλέον η πιο διαδεδομένη τεχνική, αφού το κόστος της μνήμης δεν είναι πλέον απαγορευτικό.

Συστήματα Τύπων

- Για να κάνει έλεγχο τύπων ο μεταγλωττιστής πρέπει να αναθέσει ένα τύπο σε κάθε μέρος του προγράμματος.

Συστήματα Τύπων

- Για να κάνει έλεγχο τύπων ο μεταγλωττιστής πρέπει να αναθέσει ένα τύπο σε κάθε μέρος του προγράμματος.

- Στην συνέχεια μπορεί να ελέγξει πως αυτοί οι τύποι ακολουθούν μια συλλογή από λογικούς κανόνες που ονομάζεται το **σύστημα τύπων** της γλώσσας.

Συστήματα Τύπων

Το σύστημα τύπων μιας γλώσσας προκύπτει από την περιγραφή της γλώσσας.

Για παράδειγμα η εκδόσεις της γλώσσας C είναι οι εξής:

1. **Ansi C** ή **C89**
2. **C90**
3. **C99**
4. **C1X**

Ο σύνδεσμος `http://www.open-std.org/jtc1/sc22/wg14/www/standards` περιέχει περισσότερες πληροφορίες.

DRAFT Ansi C Standard (ANSI X3J11/88-090) http://flash-gordon.me.uk/ansi.c.txt

```
3.3.5 Multiplicative operators

Syntax

          multiplicative-expression:
                  cast-expression
                  multiplicative-expression *  cast-expression
                  multiplicative-expression /  cast-expression
                  multiplicative-expression %  cast-expression

Constraints

   Each of the operands shall have arithmetic type.  The operands of
the % operator shall have integral type.
```

3.3.5 Multiplicative operators

Semantics

The usual arithmetic conversions are performed on the operands.

The result of the binary * operator is the product of the operands.

The result of the / operator is the quotient from the division of the first operand by the second; the result of the % operator is the remainder. In both operations, if the value of the second operand is zero, the behavior is undefined.

When integers are divided and the division is inexact, if both operands are positive the result of the / operator is the largest integer less than the algebraic quotient and the result of the % operator is positive. If either operand is negative, whether the result of the / operator is the largest integer less than the algebraic quotient or the smallest integer greater than the algebraic quotient is implementation-defined, as is the sign of the result of the % operator. If the quotient a/b is representable, the expression (a/b)*b + a%b shall equal a .

Έλεγχος Τύπων

Η διαδικασία ελέγχου τύπων μας βοηθάει να ανακαλύψουμε λάθη στα προγράμματα μας.

1. **στατικός**: ο μεταγλωττιστής φροντίζει να ελέγξει πως δεν θα συμβούν λάθη τύπων κατά την εκτέλεση.
2. **δυναμικός**: ο μεταγλωττιστής εμπλουτίζει τον παραγόμενο κώδικα με πληροφορίες για τους τύπους ώστε ο έλεγχος να γίνεται δυναμικά.

Έλεγχος Τύπων

Ο έλεγχος τύπων μπορεί να πάρει δύο μορφές:

1. σύνθεση (synthesis)
   - φτιάχνει τον τύπο μίας έκφρασης από τους τύπους των υποεκφράσεων
2. συμπερασμός (inference)
   - καθορίζει τον τύπο από τον τρόπο χρήσης
   - χρησιμοποιείται σε γλώσσες που δεν πρέπει να δηλώνουμε μεταβλητές πριν από την χρήση τους

Εκφράσεις Τύπων
Type Expressions

Αναπαριστούμε έναν τύπο με μία *έκφραση τύπου*

1. που είναι ένας βασικός τύπος (*basic type*),
2. φτιάχνεται χρησιμοποιώντας έναν *κατασκευαστή τύπων* (*type constructor*) επάνω σε μια έκφραση τύπου.

Εκφράσεις Τύπων
Type Expressions

- Μια έκφραση τύπου είναι ένας βασικός τύπος όπως boolean, char, integer, float και void.

  π.χ στην C

  ```
  int a;
  ```

  θα είχαμε πως ο τύπος του `a` είναι integer

Εκφράσεις Τύπων
Type Expressions

- Μια έκφραση τύπου είναι ένας βασικός τύπος όπως boolean, char, integer, float και void.

  π.χ στην C

  ```
  int a;
  ```

  θα είχαμε πως ο τύπος του `a` είναι integer

- Μια έκφραση τύπου μπορεί να φτιαχτεί με τον κατασκευαστή τύπων πίνακα *array* που πέρνει παραμέτρους έναν αριθμό και μια έκφραση τύπου π.χ *array(3, integer)*

  ```
  float x[10];
  ```

  ο τύπος του `x` είναι array(10, float)

  ```
  float y[10][20];
  ```

  ο τύπος του `y` είναι array(10, array(20, float))

Εκφράσεις Τύπων
Type Expressions

- Εαν s και t είναι εκφράσεις τύπων, τότε το καρτεσιανό γινόμενο τους $s \times t$ είναι έκφραση τύπου. Ο τελεστής $\times$ είναι αριστερά προσεταιριστικός. Με γινόμενα μπορούμε να φτιάξουμε λίστες ή ζευγάρια τύπων π.χ για παράμετρους σε συναρτήσεις.

Εκφράσεις Τύπων
Type Expressions

- Εαν s και t είναι εκφράσεις τύπων, τότε το καρτεσιανό γινόμενο τους $s \times t$ είναι έκφραση τύπου. Ο τελεστής $\times$ είναι αριστερά προσεταιριστικός. Με γινόμενα μπορούμε να φτιάξουμε λίστες ή ζευγάρια τύπων π.χ για παράμετρους σε συναρτήσεις.

- Ένα όνομα τύπου είναι μια έκφραση τύπου. Θα δούμε την χρήση του παρακάτω.

Εκφράσεις Τύπων
Type Expressions

- Εαν s και t είναι εκφράσεις τύπων, τότε το καρτεσιανό γινόμενο τους $s \times t$ είναι έκφραση τύπου. Ο τελεστής $\times$ είναι αριστερά προσεταιριστικός. Με γινόμενα μπορούμε να φτιάξουμε λίστες ή ζευγάρια τύπων π.χ για παράμετρους σε συναρτήσεις.

- Ένα όνομα τύπου είναι μια έκφραση τύπου. Θα δούμε την χρήση του παρακάτω.

- Με τον κατασκευαστή record φτιάχνουμε *εγγραφές*. Η διαφορά του record από το γινόμενο είναι πως στην εγγραφή τα πεδία έχουν ονόματα.

Εκφράσεις Τύπων
Type Expressions

```
typedef struct {
    integer AM;
    char name[40];
} student;

student S[10];
```

Το παραπάνω πρόγραμμα ορίζει έναν καινούριο τύπο που αναπαρίστατε από την έκφραση τύπων

$$\text{record((}\mathbf{AM} \times \text{integer)} \times (\mathbf{name} \times \text{array(40,char)))}$$

ενώ η μεταβλητή `S` στην συνέχεια έχει τύπο

$$\text{array(10,record((}\mathbf{AM} \times \text{integer)} \times (\mathbf{name} \times \text{array(40,char))))}$$

Εκφράσεις Τύπων
Type Expressions

- Μια έκφραση τύπου φτιάχνεται με τον κατασκευαστή τύπων *pointer*. Εάν *t* είναι μια έκφραση τύπου, τότε pointer(t) είναι η έκφραση τύπου για ”δείκτη σε αντικείμενο τύπου t”.

  π.χ

  ```
  int *ptr;
  ```

  η μεταβλητή `ptr` έχει τύπο pointer(integer)

  π.χ

  ```
  char **aptr;
  ```

  η μεταβλητή `aptr` έχει τύπο pointer(pointer(char))

Εκφράσεις Τύπων
Type Expressions

- Μια έκφραση τύπου φτιάχνεται με τον κατασκευαστή τύπων $\rightarrow$ για τύπους συνάρτησης. Γράφουμε $s \rightarrow t$ για "συνάρτηση από τον τύπο s στον τύπο t".

  π.χ

  ```
  char* strcpy(char *dest, char *src);
  ```

  η παραπάνω συνάρτητη με όνομα `strcpy` έχει τύπο

  $$\text{pointer(char)} \times \text{pointer(char)} \rightarrow \text{pointer(char)}$$

  θεωρούμε πως ο τελεστής $\times$ έχει μεγαλύτερη προτεραιότητα από τον τελεστή $\rightarrow$ που σημαίνει πως $\text{int} \times \text{int} \rightarrow \text{int}$ είναι ισοδύναμο με $(\text{int} \times \text{int}) \rightarrow \text{int}$.

  Ο τελεστής $\rightarrow$ είναι δεξιά προσεταιριστικός.

Εκφράσεις Τύπων
Type Expressions

- Μια έκφραση τύπου μπορεί να περιέχει μεταβλητές η τιμή των οποίων είναι έκφραση τύπου.

  Οι μεταβλητές είναι χρήσιμες για να μιλάμε για άγνωστους τύπους.

  Σε πολλές γλώσσες δεν είναι απαραίτητη η δήλωση ενός τύπου την ώρα δήλωσης μιας μεταβλητής.

  Ο μεταγλωττιστής όμως θέλει να κάνει διάφορους ελέγχους, π.χ εαν ο προγραμματιστής χρησιμοποιεί μια μεταβλητή ως ακέραιο σε μια πρόταση και ως πίνακα σε άλλη.

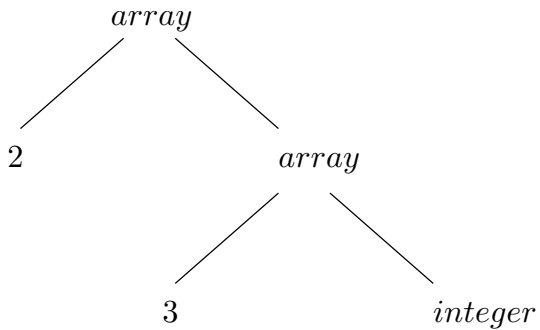
Ο συμπερασμός τύπων (type inference) είναι η διαδικασία κατανόησης μιας κατασκευής μιας γλώσσας από τον τρόπο χρήσης της.

Εκφράσεις Τύπων

Αναπαράσταση

Μια βολική αναπαράσταση εκφράσεων τύπων είναι η χρήση γραφημάτων.

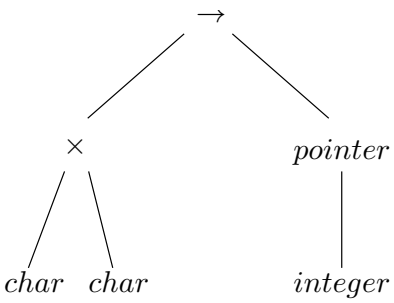
Παράδειγμα έκφρασης τύπου για `int`[2][3].

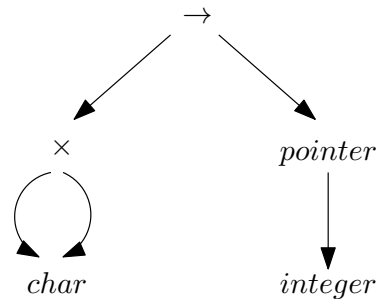


Παράδειγμα έκφρασης τύπου για

$$\text{char} \times \text{char} \rightarrow \text{pointer(integer)}$$

αριστερά ως δέντρο και δεξιά ως DAG (κατευθυνόμενο ακυκλικό γράφημα).





Αποθήκευση Τοπικών Ονομάτων

- Από τον τύπο των αναγνωριστικών μπορούμε να βρούμε και τον χώρο που θα χρειαστούμε για την αποθήκευση κατά την διάρκεια της εκτέλεσης του προγράμματος.

Αποθήκευση Τοπικών Ονομάτων

(i) Από τον τύπο των αναγνωριστικών μπορούμε να βρούμε και τον χώρο που θα χρειαστούμε για την αποθήκευση κατά την διάρκεια της εκτέλεσης του προγράμματος.

(ii) Κατά την μεταγλώττιση μπορούμε να χρησιμοποιήσουμε την πληροφορία αυτή για να δώσουμε σε κάθε αναγνωριστικό μία σχετική διεύθυνση.

Αποθήκευση Τοπικών Ονομάτων

(i) Από τον τύπο των αναγνωριστικών μπορούμε να βρούμε και τον χώρο που θα χρειαστούμε για την αποθήκευση κατά την διάρκεια της εκτέλεσης του προγράμματος.

(ii) Κατά την μεταγλώττιση μπορούμε να χρησιμοποιήσουμε την πληροφορία αυτή για να δώσουμε σε κάθε αναγνωριστικό μία σχετική διεύθυνση.

(iii) Ο τύπος και η σχετική διεύθυνση αποθηκεύονται στον πίνακα συμβόλων για το αναγνωριστικό.

Αποθήκευση Τοπικών Ονομάτων

(i) Από τον τύπο των αναγνωριστικών μπορούμε να βρούμε και τον χώρο που θα χρειαστούμε για την αποθήκευση κατά την διάρκεια της εκτέλεσης του προγράμματος.

(ii) Κατά την μεταγλώττιση μπορούμε να χρησιμοποιήσουμε την πληροφορία αυτή για να δώσουμε σε κάθε αναγνωριστικό μία σχετική διεύθυνση.

(iii) Ο τύπος και η σχετική διεύθυνση αποθηκεύονται στον πίνακα συμβόλων για το αναγνωριστικό.

(iv) Πληροφορία που έχει μέγεθος που δεν μπορεί να προσδιοριστεί κατά την μεταγλώττιση όπως δυναμικοί πίνακες, υλοποιούνται με την δέσμευση ενός σταθερού μεγέθους χώρου που να μπορεί να αποθηκεύσει ένα δείκτη στην πληροφορία.

Δηλώσεις
Declarations

- Θα δούμε με λεπτομέρεια πως μπορούμε μέσα από το μεταφραστικό τμήμα να μαζέψουμε την απαραίτητη πληροφορία για τα αναγνωριστικά.

Δηλώσεις
Declarations

(i) Θα δούμε με λεπτομέρεια πως μπορούμε μέσα από το μεταφραστικό τμήμα να μαζέψουμε την απαραίτητη πληροφορία για τα αναγνωριστικά.

(ii) Εκτός από τον ακριβή τύπο κάθε προσδιοριστικού πρέπει να ξέρουμε και το ”πλάτος” του (width), δηλαδή το μέγεθος σε bytes που χρειαζόμαστε για την αποθήκευση του.

Πλάτος Τύπων

```
1   typedef struct {
2       int real, img;
3   } complex;
4
5   int main() {
6
7       complex a[100];
8
9       a[13].real = 1;
10      a[13].img = 2;
11
12      /* ... */
13  }
```

(i) Στην γραμμή 10 για να μπορέσει ο μεταγλωττιστής να παράξει κώδικα πρέπει να ξέρει που ξεκινά στην μνήμη η μεταβλητή $a[13].img$

Πλάτος Τύπων

```
1   typedef struct {
2       int real, img;
3   } complex;
4
5   int main() {
6
7       complex a[100];
8
9       a[13].real = 1;
10      a[13].img = 2;
11
12      /* ... */
13  }
```

(i) Στην γραμμή 10 για να μπορέσει ο μεταγλωττιστής να παράξει κώδικα πρέπει να ξέρει που ξεκινά στην μνήμη η μεταβλητή $a[13].img$

(ii) Συναρτήσει της θέσης μνήμης a όπου ξεκινά ο πίνακας, η μεταβλητή αυτή βρίσκεται στην θέση

$$a + 13 * width(complex) + offset(img)$$

Πλάτος Τύπων

```
1   typedef struct {
2       int real, img;
3   } complex;
4
5   int main() {
6
7       complex a[100];
8
9       a[13].real = 1;
10      a[13].img = 2;
11
12      /* ... */
13  }
```

(i) Στην γραμμή 10 για να μπορέσει ο μεταγλωττιστής να παράξει κώδικα πρέπει να ξέρει που ξεκινά στην μνήμη η μεταβλητή $a[13].img$

(ii) Συναρτήσει της θέσης μνήμης a όπου ξεκινά ο πίνακας, η μεταβλητή αυτή βρίσκεται στην θέση

$$a + 13 * width(complex) + offset(img)$$

(iii) Πρέπει δηλαδή να έχει πληροφορίες για το πλάτος κάθε τύπου (μέγεθος σε bytes) καθώς και την θέση μίας μεταβλητής μέσα σε μια εγγραφή.

Πλάτος Τύπων

```
1   typedef struct {
2       int real, img;
3   } complex;
4
5   int main() {
6
7       complex a[100];
8
9       a[13].real = 1;
10      a[13].img = 2;
11
12      /* ... */
13  }
```

(i) Στην γραμμή 10 για να μπορέσει ο μεταγλωττιστής να παράξει κώδικα πρέπει να ξέρει που ξεκινά στην μνήμη η μεταβλητή $a[13].img$

(ii) Συναρτήσει της θέσης μνήμης a όπου ξεκινά ο πίνακας, η μεταβλητή αυτή βρίσκεται στην θέση

$$a + 13 * width(complex) + offset(img)$$

(iii) Πρέπει δηλαδή να έχει πληροφορίες για το πλάτος κάθε τύπου (μέγεθος σε bytes) καθώς και την θέση μίας μεταβλητής μέσα σε μια εγγραφή.

(iv) Το $offset(img)$ εδώ είναι ουσιαστικά ίσο με το πλάτος όλων των τύπων που υπάρχουν στην ίδια εγγραφή και είναι πριν από το img.

Πλάτος Τύπων

```
1   typedef struct {
2       int real, img;
3   } complex;
4
5   int main() {
6
7       complex a[100];
8
9       a[13].real = 1;
10      a[13].img = 2;
11
12      /* ... */
13  }
```

(i) Στην γραμμή 10 για να μπορέσει ο μεταγλωττιστής να παράξει κώδικα πρέπει να ξέρει που ξεκινά στην μνήμη η μεταβλητή $a[13].img$

(ii) Συναρτήσει της θέσης μνήμης a όπου ξεκινά ο πίνακας, η μεταβλητή αυτή βρίσκεται στην θέση

$$a + 13 * width(complex) + offset(img)$$

(iii) Πρέπει δηλαδή να έχει πληροφορίες για το πλάτος κάθε τύπου (μέγεθος σε bytes) καθώς και την θέση μίας μεταβλητής μέσα σε μια εγγραφή.

(iv) Το $offset(img)$ εδώ είναι ουσιαστικά ίσο με το πλάτος όλων των τύπων που υπάρχουν στην ίδια εγγραφή και είναι πριν από το img.

(v) θεωρώντας πως έχουν ακέραιους με πλάτος 8 bytes, η μεταβλητή βρίσκεται στην θέση $a + 13 * 16 + 8$.

Πλάτος Τύπων

- Κατά την διάρκεια της σημασιολογικής ανάλυσης θα μαζέψουμε ταυτόχρονα πληροφορίες για τον τύπο και για το πλάτος των προσδιοριστικών.

Πλάτος Τύπων

(i) Κατά την διάρκεια της σημασιολογικής ανάλυσης θα μαζέψουμε ταυτόχρονα πληροφορίες για τον τύπο και για το πλάτος των προσδιοριστικών.

(ii) Η πληροφορίες αυτές θα βοηθήσουν και για τον έλεγχο ορθότητας, αλλά θα μας είναι απαραίτητες και στις επόμενες φάσεις της μεταγλώττισης όπου θα πρέπει να κατασκευάσουμε κώδικα μηχανής.

Δηλώσεις
Declarations

$$
\begin{array}{lcl}
P & \rightarrow & D \\
D & \rightarrow & T\ L\ \mathbf{;}\ D \\
D & \rightarrow & \epsilon \\
L & \rightarrow & \mathbf{id} \\
L & \rightarrow & L, \mathbf{id} \\
T & \rightarrow & B\ C \\
T & \rightarrow & \mathbf{struct\ \{}\ \mathrm{D}\ \mathbf{\}} \\
B & \rightarrow & \mathbf{char} \\
B & \rightarrow & \mathbf{int} \\
B & \rightarrow & \mathbf{double} \\
C & \rightarrow & \epsilon \\
C & \rightarrow & \mathbf{[\ number\ ]}\ C
\end{array}
$$

Η παραπάνω γραμματική περιγράφει δηλώσεις μεταβλητών, υποστηρίζοντας πίνακες και εγγραφές.

Δηλώσεις

Ας δούμε πρώτα το μεταφραστικό σχήμα για τους τύπους χωρίς τις εγγραφές.

$$
\begin{array}{lcl}
T & \rightarrow & B\ C \\
B & \rightarrow & \textbf{char} \\
B & \rightarrow & \textbf{int} \\
B & \rightarrow & \textbf{double} \\
C & \rightarrow & \epsilon \\
C & \rightarrow & \textbf{[number]}\ C
\end{array}
$$

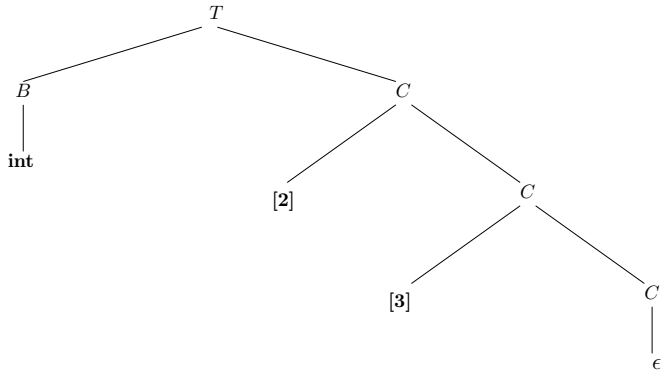
Δηλώσεις

Ας δούμε πρώτα το μεταφραστικό σχήμα για τους τύπους χωρίς τις εγγραφές.

T	$\rightarrow$	B	{	curType=B.type; curWidth=B.width;	}
		C	{	T.type=C.type; T.width=C.width;	}
B	$\rightarrow$	**char**	{	B.type=char; B.width=1;	}
B	$\rightarrow$	**int**	{	B.type=int; B.width=4;	}
B	$\rightarrow$	**double**	{	B.type=double; B.width=8;	}
C	$\rightarrow$	ϵ	{	C.type=curType; C.width=curWidth;	}
C	$\rightarrow$	**[number]** C_1	{	C.type=array(num.yylval, C_1.type) C.width=num.yylval $\times$ C_1.width	}

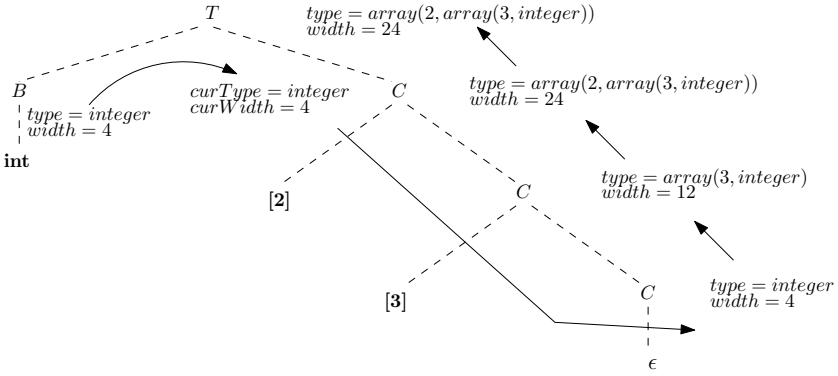
Παράδειγμα

Το συντακτικό δέντρο για τον τύπο *int*$[2][3]$ φαίνεται παρακάτω:



Παράδειγμα

Το συντακτικό δέντρο για τον τύπο *int*[2][3] φαίνεται παρακάτω:



Μαζί με την μετάφραση από το μεταφραστικό σχήμα που παρουσιάσαμε.

Ακολουθίες Δηλώσεων

- Γλώσσες όπως η C, η C++ και η Java μας επιτρέπουν να διαχειριστούμε όλες τις δηλώσεις σε μια συνάρτηση ως μια ομάδα.

Ακολουθίες Δηλώσεων

(i) Γλώσσες όπως η C, η C++ και η Java μας επιτρέπουν να διαχειριστούμε όλες τις δηλώσεις σε μια συνάρτηση ως μια ομάδα.

(ii) Αντίστοιχα και όλες τις δηλώσεις μέσα σε μία εγγραφή ή κλάση.

Ακολουθίες Δηλώσεων

(i) Γλώσσες όπως η C, η C++ και η Java μας επιτρέπουν να διαχειριστούμε όλες τις δηλώσεις σε μια συνάρτηση ως μια ομάδα.

(ii) Αντίστοιχα και όλες τις δηλώσεις μέσα σε μία εγγραφή ή κλάση.

(iii) Οι δηλώσεις στην C++ και στην Java μπορεί να βρίσκονται σε όλο το μήκος μιας συνάρτησης ή μιας κλάσης αλλά και πάλι μπορούμε να τις διαχειριστούμε ως μια ομάδα.

Ακολουθίες Δηλώσεων

(i) Γλώσσες όπως η C, η C++ και η Java μας επιτρέπουν να διαχειριστούμε όλες τις δηλώσεις σε μια συνάρτηση ως μια ομάδα.

(ii) Αντίστοιχα και όλες τις δηλώσεις μέσα σε μία εγγραφή ή κλάση.

(iii) Οι δηλώσεις στην C++ και στην Java μπορεί να βρίσκονται σε όλο το μήκος μιας συνάρτησης ή μιας κλάσης αλλά και πάλι μπορούμε να τις διαχειριστούμε ως μια ομάδα.

(iv) Για το μεταφραστικό σχήμα, θα χρησιμοποιήσουμε μια μεταβλητή, *offset*, η οποία θα μας βοηθήσει ώστε να υπολογίσουμε την σχετική θέση της μεταβλητής ως προς την αρχή της ομάδας στην οποία ανήκει.

Ακολουθίες Δηλώσεων

(i) Γλώσσες όπως η C, η C++ και η Java μας επιτρέπουν να διαχειριστούμε όλες τις δηλώσεις σε μια συνάρτηση ως μια ομάδα.

(ii) Αντίστοιχα και όλες τις δηλώσεις μέσα σε μία εγγραφή ή κλάση.

(iii) Οι δηλώσεις στην C++ και στην Java μπορεί να βρίσκονται σε όλο το μήκος μιας συνάρτησης ή μιας κλάσης αλλά και πάλι μπορούμε να τις διαχειριστούμε ως μια ομάδα.

(iv) Για το μεταφραστικό σχήμα, θα χρησιμοποιήσουμε μια μεταβλητή, *offset*, η οποία θα μας βοηθήσει ώστε να υπολογίσουμε την σχετική θέση της μεταβλητής ως προς την αρχή της ομάδας στην οποία ανήκει.

(v) Θα προσθέτουμε στον πίνακα συμβόλων αυτή την σχετική θέση μαζί με το αναγνωριστικό της μεταβλητής.

Παράδειγμα

Προσθέτοντας στον πίνακα συμβόλων την σχετική θέση μιας μεταβλητής με βάση την αρχή της ομάδας (π.χ εγγραφής) μπορούμε να υποστηρίξουμε και κώδικα όπως φαίνεται στο παράδειγμα.

```
1   #include <stdio.h>
2
3   int main() {
4
5       struct {
6           int x;
7
8           struct {
9               int a;
10              int x;
11              int y;
12          } y;
13      } p;
14
15      p.x = 1;
16      p.y.x = 2;
17      p.y.y = 3;
18
19      return 0;
20  }
```

Παράδειγμα

Για να υπολογίσει ο μεταγλωττιστής σε ποια θέση μνήμης βρίσκεται η μεταβλητή *p.y.y* πρέπει να

1. βρει την θέση μνήμης της *p*
2. επειδή η *p* είναι εγγραφή, να βρει τον πίνακα συμβόλων της
3. να βρει πως η μεταβλητή *y* είναι σε offset 4 από την αρχή της εγγραφής
4. επειδή η *p.y* είναι εγγραφή να βρει τον πίνακα συμβόλων της
5. να βρει πως η μεταβλητή *y* είναι σε offset 8 από την αρχή της εγγραφής

Η μεταβλητή *p.y.y* λοιπόν είναι στην διεύθυνση μνήμης $\&p + 4 + 8$ θεωρώντας πως *sizeof*(*int*) $= 4$.

```
1  #include <stdio.h>
2
3  int main() {
4
5      struct {
6          int x;
7
8          struct {
9              int a;
10             int x;
11             int y;
12         } y;
13     } p;
14
15     p.x = 1;
16     p.y.x = 2;
17     p.y.y = 3;
18
19     return 0;
20 }
```

Παράδειγμα

Για να υποστηρίξουμε τέτοιες λειτουργίες, όταν προσθέτουμε μια μεταβλητή τύπου εγγραφής ή κλάσης σε έναν πίνακα συμβόλων (π.χ της συνάρτησης main), αποθηκεύουμε ως τύπο της (π.χ record) μεταβλητής τον τύπο της μαζί με ένα δείκτη στον πίνακα συμβόλων της.

```c
#include <stdio.h>

int main() {

    struct {
        int x;

        struct {
            int a;
            int x;
            int y;
        } y;
    } p;

    p.x = 1;
    p.y.x = 2;
    p.y.y = 3;

    return 0;
}
```

Ακολουθίες Δηλώσεων

Έστω πως η μεταβλητή *top* μας δείχνει τον πίνακα συμβόλων που είναι ενεργοποιημένος στο σημείο που βρίσκονται οι δηλώσεις.

- Τον ονομάζουμε *top* γιατί είναι η κορυφή μιας στοίβας πινάκων συμβόλων.

Ακολουθίες Δηλώσεων

Έστω πως η μεταβλητή *top* μας δείχνει τον πίνακα συμβόλων που είναι ενεργοποιημένος στο σημείο που βρίσκονται οι δηλώσεις.

- Τον ονομάζουμε *top* γιατί είναι η κορυφή μιας στοίβας πινάκων συμβόλων.

Θα γράψουμε το μεταφραστικό σχήμα για τις δηλώσεις μεταβλητών που αφορά το παρακάτω κομμάτι της γραμματικής μας.

$$
\begin{array}{lcl}
P & \rightarrow & D \\
D & \rightarrow & T\ L\ \mathbf{;}\ D \\
D & \rightarrow & \epsilon \\
L & \rightarrow & \mathbf{id} \\
L & \rightarrow & L, \mathbf{id}
\end{array}
$$

Ακολουθίες Δηλώσεων

P	$\rightarrow$	D	{ offset=0;	}
D	$\rightarrow$	T L **;** D	{ curType=T.type; curWidth=T.width;	}
D	$\rightarrow$	ϵ		
L	$\rightarrow$	**id**	{ top.put(id.yylval, curType, offset); offset = offset + curWidth;	}
L	$\rightarrow$	L, **id**	{ top.put(id.yylval, curType, offset); offset = offset + curWidth;	}

Ακολουθίες Δηλώσεων

Μπορούμε να προσθέσουμε και κανόνες ώστε να ελέγξουμε πως δεν υπάρχουν δύο φορές τα ίδια αναγνωριστικά.

P	$\rightarrow$	D	{ offset=0;	}
D	$\rightarrow$	T L ; D	{ curType=T.type; curWidth=T.width;	}
D	$\rightarrow$	ϵ		
L	$\rightarrow$	**id**	{ if (top.getOnlyInTop(id.yylval) != NULL) throw ”error”; top.put(id.yylval, curType, offset); offset = offset + curWidth;	}
L	$\rightarrow$	L, **id**	{ if (top.getOnlyInTop(id.yylval) != NULL) throw ”error”; top.put(id.yylval, curType, offset); offset = offset + curWidth;	}

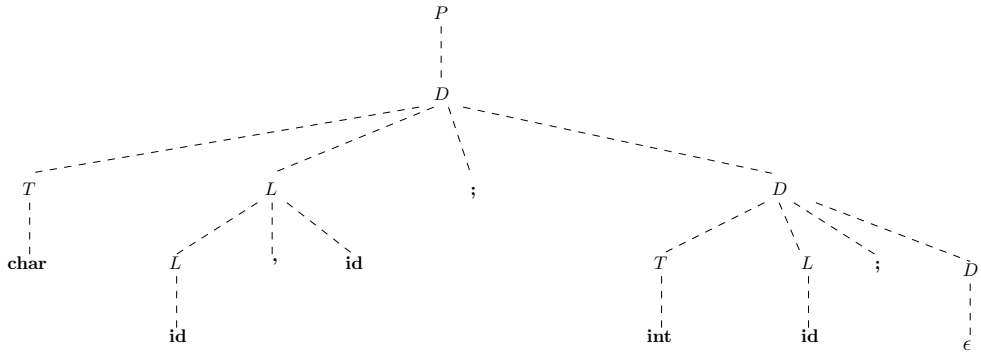
Η συνάρτηση top.getOnlyInTop() ελέγχει μόνο στον πίνακα συμβόλων που είναι ενεργός. Όταν όμως χρησιμοποιούμε σε εκφράσεις ένα αναγνωριστικό κάνουμε αναζήτηση του σε όλη την στοίβα των πινάκων συμβόλων.

Ακολουθίες Δηλώσεων

Ας δούμε πως λειτουργεί αυτό το μεταφραστικό σχήμα στην είσοδο

```
int foo() {
    char x, y;
    int f;

    /* ... */
}
```

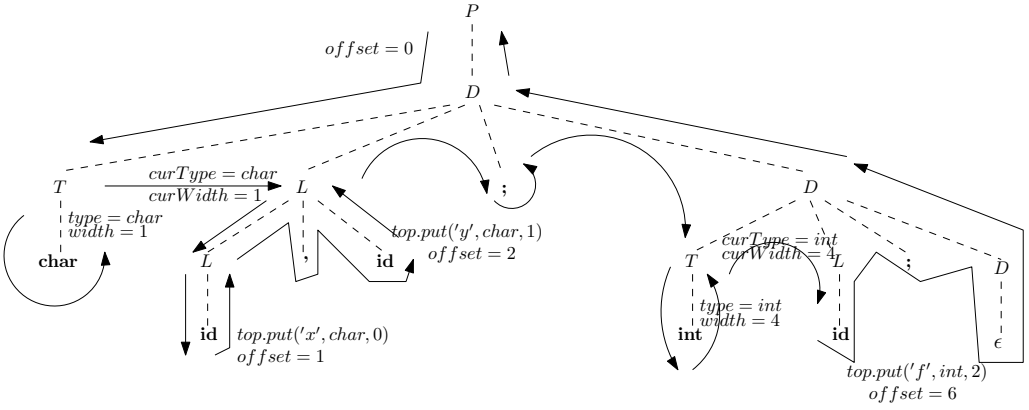


Ακολουθίες Δηλώσεων

Ας δούμε πως λειτουργεί αυτό το μεταφραστικό σχήμα στην είσοδο

```
int foo() {
    char x, y;
    int f;

    /* ... */
}
```



Πεδία σε Εγγραφές

Οι ιδέες που χρησιμοποιήσαμε ως τώρα λειτουργούν και για εγγραφές αρκεί να προσέξουμε δύο λεπτομέρειες:

(i) Τα ονόματα των πεδίων μέσα σε μια εγγραφή πρέπει να είναι μοναδικά.
(ii) Η σχετική θέση ενός πεδίου (offset) είναι με βάση την αρχή της εγγραφής.

Για ευκολία ο τύπος μια εγγραφής έχει την μορφή *record*(*t*) όπου *record*() είναι ένας κατασκευαστής τύπου και *t* είναι ένα αντικείμενο πίνακα συμβόλων που περιέχει πληροφορίες για τα πεδία αυτής της εγγραφής.

Πεδία σε Εγγραφές

$$
\begin{array}{lcl}
P & \rightarrow & D \\
D & \rightarrow & T\ L\ \mathbf{;}\ D \\
D & \rightarrow & \epsilon \\
L & \rightarrow & \mathbf{id} \\
L & \rightarrow & L, \mathbf{id} \\
T & \rightarrow & B\ C \\
 & & \\
T & \rightarrow & \mathbf{struct\ \{}\ \mathrm{D}\ \mathbf{\}} \\
 & & \\
B & \rightarrow & \mathbf{char} \\
B & \rightarrow & \mathbf{int} \\
B & \rightarrow & \mathbf{double} \\
C & \rightarrow & \epsilon \\
C & \rightarrow & \mathbf{[\ number\ ]}\ C
\end{array}
$$

Πεδία σε Εγγραφές

T	$\rightarrow$	**struct {**	{	Stack.push(top);
				top = new SymbolTable(top);
				Stack.push(offset);
				offset = 0; }
		D **}**	{	T.type = record(top);
				T.width = offset;
				offset = Stack.pop();
				top = Stack.pop(); }

Θεωρώντας πως η κλάση SymbolTable υλοποιεί μια στοίβα από πίνακες συμβόλων. Αναζήτηση συμβόλων σε έναν τέτοιο πίνακα συμβόλων γίνεται πρώτα από την κορυφή της στοίβας και συνεχίζει προς τα κάτω.

Έλεγχος Τύπων

Για να κάνει έλεγχο τύπων ο μεταγλωττιστής πρέπει να αναθέσει μία έκφραση τύπου σε κάθε μέρος του πηγαίου προγράμματος όπως:

- εκφράσεις (expressions)
- προτάσεις (statements)

Έλεγχος Τύπων

Για να κάνει έλεγχο τύπων ο μεταγλωττιστής πρέπει να αναθέσει μία έκφραση τύπου σε κάθε μέρος του πηγαίου προγράμματος όπως:

- εκφράσεις (expressions)
- προτάσεις (statements)

Στην συνέχεια πρέπει να ελέγξει πως οι τύποι αυτοί ακολουθούν μία συλλογή από λογικούς κανόνες της γλώσσας που ονομάζονται το **σύστημα τύπων** της γλώσσας.

Στην περίπτωση των γλωσσών προγραμματισμού που υποχρεώνουν τον προγραμματιστή να δηλώσει τον τύπο των μεταβλητών πριν από την χρήση τους, ο τύπος της έκφρασης

$$E_1 + E_2$$

προκύπτει μέσω **σύνθεσης** από τους τύπους των εκφράσεων

$$E_1$$

και

$$E_2.$$

Εκφράσεις, Προτάσεις και Έλεγχος Τύπων

Ένας τυπικός κανόνας σύνθεσης είναι:

Εάν
- f έχει τύπο $s \to t$
- x έχει τύπο s

τότε
- η έκφραση $f(x)$ έχει τύπο t

όπου x και f είναι εκφράσεις και $s \to t$ είναι συνάρτηση από το s στο t.

Εκφράσεις, Προτάσεις και Έλεγχος Τύπων

Ένας τυπικός κανόνας σύνθεσης είναι:

Εάν
- f έχει τύπο $s \to t$
- x έχει τύπο s

τότε
- η έκφραση $f(x)$ έχει τύπο t

όπου x και f είναι εκφράσεις και $s \to t$ είναι συνάρτηση από το s στο t.

- Ο κανόνας αυτός λειτουργεί και για συναρτήσεις με πολλές παραμέτρους.
- Επίσης λειτουργεί και για εκφράσεις $E_1 + E_2$ εαν δούμε την έκφραση αυτή ως την εφαρμογή μίας συνάρτησης $add(E_1, E_2)$.

Εκφράσεις, Προτάσεις και Έλεγχος Τύπων

Ο τρόπος αντιμετώπισης των προτάσεων (statements) είναι παρόμοιος με τον τρόπο για τις εκφράσεις.

Για παράδειγμα αντιμετωπίζουμε μία πρόταση

```
if ( E ) S;
```

ως την χρήση μίας συνάρτησης

- με όνομα *if*
- που πέρνει παραμέτρους *E* και *S*.

Ο τύπος της συνάρτησης είναι

$$\mathit{boolean} \times \mathit{void} \rightarrow \mathit{void}.$$

Μετατροπές Τύπων

Έστω η έκφραση

$$x + i$$

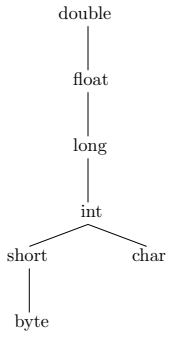
όπου x είναι τύπου *float* και i είναι τύπου *integer*.

- Η αναπαράσταση των ακεραίων και των αριθμών κινητής υποδιαστολής είναι διαφορετική στη μηχανή και διαφορετικές εντολές μηχανής πρέπει να χρησιμοποιηθούν για να προσθέσουν ακέραιους ή floats.
- Ο μεταγλωττιστής πρέπει να μετατρέψει ένα από τα δύο ορίσματα για να γίνει πρόσθεση με τελεστέους ίδιου τύπου.

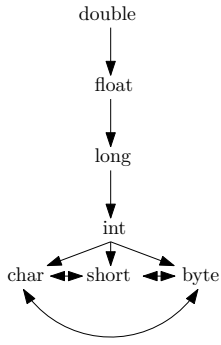
Μετατροπές Τύπων

Οι μετατροπές τύπων διαφέρουν από γλώσσα σε γλώσσα.

π.χ στην γλώσσα Java μπορούν να γίνουν οι παρακάτω μετατροπές βασικών τύπων.



- διεύρυνση (widening)
- δεν χάνεται πληροφορία



- στένωση (narrowing)
- μπορεί να χαθεί πληροφορία

Μετατροπές Τύπων

Οι μετατροπές τύπων διαφέρουν από γλώσσα σε γλώσσα.

Έμμεση Μετατροπή

Μία μετατροπή τύπων ονομάζεται **έμμεση** (implicit ή coersion) εάν γίνεται αυτόματα από τον μεταγλωττιστή.

Στις περισσότερες γλώσσες, οι έμμεσες μετατροπές είναι μόνο μετατροπές διεύρυνσης.

Άμεση Μετατροπή

Μία μετατροπή τύπων ονομάζεται **άμεση** (explicit ή cast) εάν ο προγραμματιστής πρέπει να γράψει κάτι για να γίνει η μετατροπή.

Εκφράσεις και Έλεγχος Τύπων

Expressions

$E \rightarrow E_1 + E_2$ { if (!TypeCompatible(E_1.type, E_2.type))
error("Type error");
E.type = max(E_1.type, E_2.type); }

$E \rightarrow E_1 \,\%\, E_2$ { if (E_1.type != integer || E_2.type != integer)
error("Type error"); }

$E \rightarrow$ **id** { Symbol s = symbolTable.find(id.yylval);
if (s==null) error("Unresolved symbol!");
E.type = s.type; }

Μέσω της συνάρτησης *max* κάνουμε *type conversion* ώστε το αποτέλεσμα να χωράει στον τύπο.
π.χ $max(char, int) = int$ ή $max(double, int) = double$.