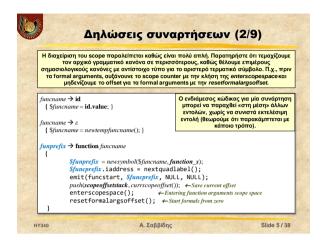
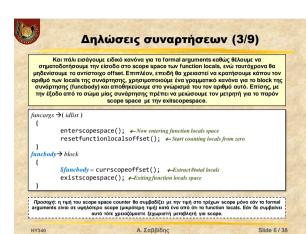


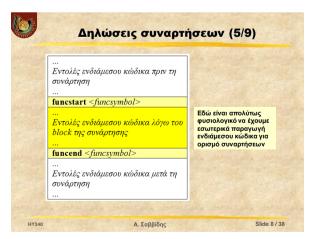
Δηλώσεις συναρτήσεων (1/9) Ο ορισμός συναρτήσεων επιτρέπεται σε οποιοδήποτε σημείο του κώδικα • είτε ως μη εκτελέσιμη εντολή • ή ως έκφραση σε παρενθετική μορφή που φέρει τη διεύθυνση της συνάρτησης Κάθε τέτοιος σωστός ορισμός οδηγεί και στη δήλωση ενός νέου συμβόλου κατηγορίας συνάρτησης • είτε με εξωτερικό αναγνωριστικό όνομα • ή με κρυφό όνομα συνάρτησης Για κάθε ορισμένη συνάρτηση πρέπει να αποθηκεύεται • ο αριθμός των συνολικών τοπικών μεταβλητών ο η διεύθυνση της (αριθμός) στον ενδιάμεσο αλλά και τελικό κώδικα (η τελευταία θα λάβει τιμή μόνο όταν εφαρμοστεί η παραγωγή τελικού κώδικα) HY340 Α. Σαββίδης

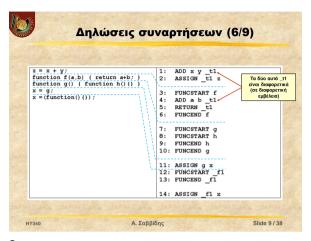
3











```
Συμπλήρωση μερικών από τις βοηθητικές συναρτήσεις. Προσθέτουμε και μία συνάρτηση patchlabel/γα να συμπληρώνουμε ένα αρχικά undefined label για κάποιο quad, αρκεί να είχαμε «σώσει» κάπου το index του.

| Void reset forma largeof feet (void) { function.Local of feet (void) } { function.Local of feet (void) } { function.Local of feet (void) } { void reset function.local if function.local of feet (void) } { case function.local if function.local fiet (void) } { case function.local if function.local fiet (void) } { default: assert(0); } } { unsigned next quad label (void) } { void patchlabel (unsigned quadNo, unsigned label) { assert (quadNo currOuad); quads (quadNo).label = label; } } { void patchlabel (unsigned quadNo, unsigned label) } { void patchlabel (unsigned quadNo, unsigned label) } { void patchlabel (unsigned quadNo, unsigned label) } { void patchlabel (unsigned void patchlabel (unsigned void patchlabel) } { void patchlabel (unsigned void patchlabel (unsigned void patchlabel (unsigned void patchlabe
```

```
Δηλώσεις συναρτήσεων (8/9)
■ Γιατί δεν κρατάμε πουθενά τον αριθμό των formal
  arguments των συναρτήσεων;
   • Γιατί ποτέ δεν απαιτείται στατικός έλεγχο ως προς τον
      αριθμό των πραγματικών ορισμάτων στη κλήση
      συνάρτησης καθώς:
       • Δεν είναι πάντα εφικτό, αφού οποιαδήποτε μεταβλητή μπορεί
         να φέρει διεύθυνση συνάρτησης, ανάλογα με τη λογική
         εκτέλεσης
       • Επιτρέπουμε να υπάρχει διαφορετικός αριθμός πραγματικών
         ορισμάτων από ότι τα τυπικά ορίσματα
       • Τα τυπικά ορίσματα στην alpha είναι προαιρετικά, ως
         συντακτική ευκολία στον προγραμματιστή της συνάρτησης,
        καθώς μπορεί να «βλέπει» όλα τα actual arguments μέσω
των συναρτήσεων βιβλιοθήκης totalarguments και argument
HY340
                          Α. Σαββίδης
```

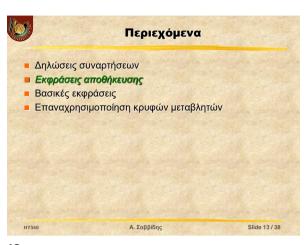
```
function dumparguments() {
    for (local i = 0, local n = totalarguments(); i < n; ++i)
    {       print(argument(i)); print("\n"); }
}

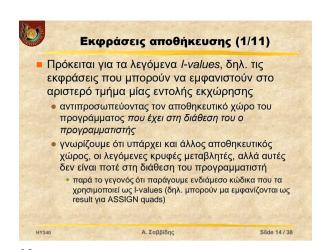
dumparguments(23, true, "hello");
dumparguments();
dumparguments,
print,
dumparguments,
(function(){}),
[]
);

13
TRUE
hello
LibraryFunction("print")
ProgramFunction("dumparguments", address n)
ProgramFunction(in(anonymous, address x)
Table[]</pre>

A Σαββίδης
Slide 12/38
```

11 12







Εκφράσεις αποθήκευσης (2/11)

- Πριν μιλήσουμε για τα I-values είναι σημαντικό να ξεκαθαρίσουμε τι είδους ορίσματα δίνουμε σε κάθε quad:
 - σταθερές τιμές (constants) που είναι τιμές συγκεκριμένου τύπου (string, number, boolean)
 - τιμή συνάρτησης βιβλιοθήκης (όνομα)
 - τιμής συνάρτησης προγράμματος (διεύθυνση)
 - κρυφή μεταβλητή (μπορεί να είναι result)
 - μεταβλητή προγράμματος (*μπορεί* να είναι result)
- Άρα το result field ενός quad μπορεί να είναι μόνο μεταβλητή (καθώς πρέπει να προσφέρει αποθηκευτικό χώρο για αποτέλεσμα).
 - Με βάση αυτή την παρατήρηση συνεχίζουμε

HY340 Α. Σαββίδης Slide 15/38

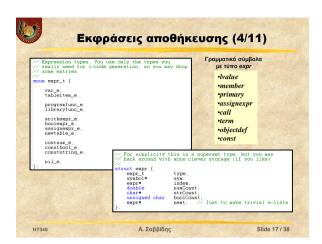


Εκφράσεις αποθήκευσης (3/11)

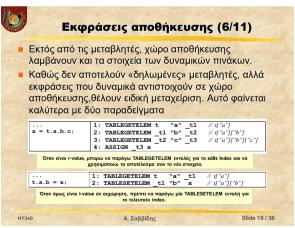
- Τα είδη των βασικών αποθηκευτικών εκφράσεων στη γραμματική μας είναι δύο:
 - ανεξάρτητες «δηλωμένες» μεταβλητές (ενδέχεται να μην καταλήξουν πάντα σε μεταβλητές, εάν το id ή το global id κάνουν resolve σε κάτι άλλο, π.χ. συνάρτηση)
 - Ivalue → id | local id | global id
 - στοιχεία πίνακα
 - Ivalue → tableitem
- Καθώς όλα αυτά αντιστοιχούν σε τύπο έκφρασης, μένει να δούμε τον τύπο δεδομένων για τις εκφράσεις, καθώς και σε ποια γραμματικά σύμβολα συνολικά αναφέρεται

HY340 Α. Σαββίδης Slide 16 / 38

15 16

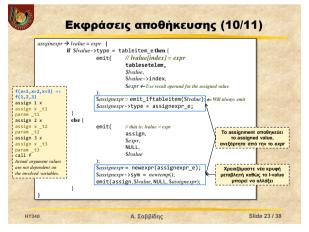




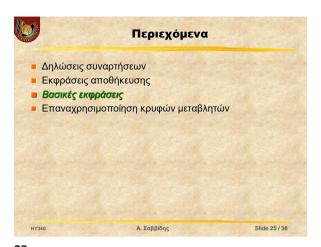


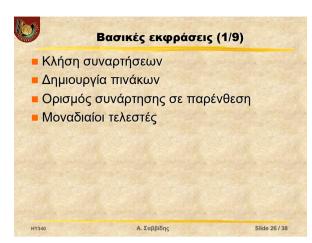


19 20



23 24





27 28

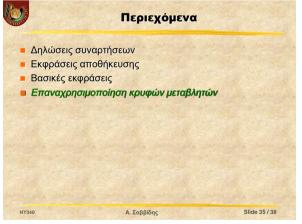
```
Anμουργία πνάκων (1/2)

| mablemake → [ elist ] |
| expr* t = newexpr(newtable_e);
| t->sym = newtemp();
| emit(tablecreate, t, NULL, NULL);
| for (int i = 0; $elist; $elist > selist > next)
| emit(tablecreate, t, NULL, NULL);
| for (int i = 0; $elist; $elist > selist > next)
| emit(tablecreate, t, NULL, NULL);
| for (int i = 0; $elist; $elist > selist > next)
| emit(tablecreate, t, newexpr_constnum(i++), $elist);
| Stablemake = t;
| x = [-4, 13, 12.34, a+b]; | 1: UMINUS 4 _ t1
| 2: ADD a b _ t2
| 3: TableSERTELEM _ t3 0 _ t1
| 5: TableSERTELEM _ t3 113
| 6: TableSERTELEM _ t3 12.34
| 7: TableSERTELEM _ t3 3 _ t2
| 8: ASSIGN x _ t4

HY340

| A Σαββίδης | Slide 29/38
```

31 32



Επαναχρησιμοποίηση κρυφών μεταβλητών (1/3) Έχουμε ήδη αναφέρει την τακτική επαναφοράς του μετρητή ονομασίας κρυφών μεταβλητών στην τιμή μηδέν μετά από κάθε stmt Ωστόσο υπάρχουν και επιπλέον περιθώρια βελτίωσης, εάν παρατηρήσει κανείς ότι μετά τη χρησιμοποίηση μίας κρυφής μεταβλητής σε μία εντολή ενδιάμεσου κώδικα ως όρισμα (αφού την πρώτη φορά είναι πάντα το αποτέλεσμα), αυτή ουσιαστικά καθίσταται «διαθέσιμη». Αυτό σημαίνει εμβόλιμο κώδικα στους σημασιολογικούς κανόνες μετάφρασης για επαναχρησιμοποίηση ανά περίπτωση HY340 Α. Σαββίδης Slide 36 / 38

35 36

