
 **HY340 : ΓΛΩΣΣΕΣ ΚΑΙ ΜΕΤΑΦΡΑΣΤΕΣ**  
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ,  
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ,  
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

---

```
VAR i:Integer;  
FUNCTION(Symbol) replicate  
  x = (function(x,y){return x*y;});  
  class DelFunctor: public std::unary_function<
```

**ΔΙΔΑΣΚΩΝ**  
Αντώνιος Σαββίδης

1

 **HY340 : ΓΛΩΣΣΕΣ ΚΑΙ ΜΕΤΑΦΡΑΣΤΕΣ**


---

**Διάλεξη 10η**  
**ΠΑΡΑΓΩΓΗ ΕΝΔΙΑΜΕΣΟΥ ΚΩΔΙΚΑ - II**

---

HY340      Α. Σαββίδης      Slide 2 / 38

2

 **Περιεχόμενα**


---

- **Δηλώσεις συναρτήσεων**
- Εκφράσεις αποθήκευσης
- Βασικές εκφράσεις
- Επαναχρησιμοποίηση κρυφών μεταβλητών

---

HY340      Α. Σαββίδης      Slide 3 / 38

3

 **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
  - 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      Slide 4 / 38

4



## Δηλώσεις συναρτήσεων (2/9)

Η διαχείριση του `scope` παραλείπεται καθώς είναι πολύ απλή. Παρατηρήστε ότι τεμαχίζουμε τον αρχικό γραμμικό κανόνα σε περισσότερους, καθώς θέλουμε επιμέρους σημασιολογικούς κανόνες με αντίστοιχο τύπο για το αριστερό τερματικό σύμβολο. Π.χ., πριν τα `formal arguments`, αυξάνουμε το `scope counter` με την κλήση της `enterscopespace` και μηδενίζουμε το `offset` για τα `formal arguments` με την `resetformalargsoffset`.

`funcname` → `id`  
{ `$funcname` = `id.value`; }

`funcname` → `ε`  
{ `$funcname` = `newtempfuncname()`; }

`funprefix` → `function funcname`

```
{
    $funprefix = newsymbol($funcname, function_s);
    $funprefix.iaddress = nextquadlabel();
    emit(funcstart, $funprefix, NULL, NULL);
    push(scopeoffsetstack, currscopeoffset()); ← Save current offset
    enterscopespace(); ← Entering function arguments scope space
    resetformalargsoffset(); ← Start formals from zero
}
```

Ο ενδιάμεσος κώδικας για μία συνάρτηση μπορεί να παραχθεί «στη μέση» άλλων εντολών, χωρίς να συνιστά εκτελέσιμη εντολή (θεωρούμε ότι παρακάμπτεται με κάποιο τρόπο).

HY340

A. Σαββίδης

Slide 5 / 38

5



## Δηλώσεις συναρτήσεων (3/9)

Και πάλι εισάγουμε ειδικό κανόνα για τα `formal arguments` καθώς θέλουμε να σηματοδοτήσουμε την είσοδο στο `scope space` των `function locals`, ενώ ταυτόχρονα θα μηδενίσουμε το αντίστοιχο `offset`. Επιπλέον, επειδή θα χρειαστεί να κρατήσουμε κάπου τον αριθμό των `locals` της συνάρτησης, χρησιμοποιούμε ένα γραμμικό κανόνα για το `block` της συνάρτησης (`funcbody`) και αποθηκεύουμε στο γνώρισμά του τον αριθμό αυτό. Επίσης, με την έξοδο από το σώμα μίας συνάρτησης πρέπει να μειώσουμε τον μετρητή για το παρόν `scope space` με την `exitscopespace`.

`funcargs` → (`idlist` )

```
{
    enterscopespace(); ← Now entering function locals space
    resetfunctionlocalsoffset(); ← Start counting locals from zero
}
```

`funcbody` → `block`

```
{
    $funcbody = currscopeoffset(); ← Extract #total locals
    existscopespace(); ← Exiting function locals space
}
```

Προσοχή: η τιμή του `scope space counter` θα συμβαδίζει με την τιμή στο τρέχον `scope` μόνο εάν τα `formal arguments` είναι σε υψηλότερο `scope` (μικρότερη τιμή) κατά ένα από ότι τα `function locals`. Εάν δε συμβαίνει αυτό τότε χρειαζόμαστε ξεχωριστή μεταβλητή για `scope`.

HY340

A. Σαββίδης

Slide 6 / 38

6



## Δηλώσεις συναρτήσεων (4/9)

Ο ολοκληρωμένος συντακτικά οδηγούμενος ορισμός για τις συναρτήσεις, ο οποίος τελειώνει με την συμπλήρωση του αριθμού των `function locals` στο σύμβολο της συνάρτησης. Με το πέρας μίας συνάρτησης θα έχει ολοκληρωθεί και η παραγωγή ενδιάμεσου κώδικα για το `block`, και τελειώνουμε με την εντολή `FUNCEND`.

`funcdef` → `funprefix funcargs funcbody`

```
{
    existscopespace(); ← Exiting function definition space
    $funprefix.totallocals = $funcbody; ← Store #locals in symbol entry
    int offset = pop_and_top(scopeoffsetstack); ← pop and get pre scope offset
    restorecurrscopeoffset(offset); ← Restore previous scope offset
    $funcdef = $funprefix; ← The function definition returns the symbol
    emit(funcend, $funprefix, NULL, NULL);
}
```

`typeof` <`funcname`>: char\*  
`typeof` <`funcbody`>: unsigned  
`typeof` <`funprefix, funcdef`>: symbol\*

HY340

A. Σαββίδης

Slide 7 / 38

7



## Δηλώσεις συναρτήσεων (5/9)

...  
Εντολές ενδιάμεσων κώδικα πριν τη συνάρτηση

`funcstart` <`funcsymbol`>

...  
Εντολές ενδιάμεσων κώδικα λόγω του `block` της συνάρτησης

`funcend` <`funcsymbol`>

...  
Εντολές ενδιάμεσων κώδικα μετά τη συνάρτηση

Εδώ είναι απολύτως φυσιολογικό να έχουμε εσωτερικά παραγωγή ενδιάμεσου κώδικα για ορισμό συναρτήσεων

HY340

A. Σαββίδης

Slide 8 / 38

8

## Δηλώσεις συναρτήσεων (6/9)

```

z = x + y;
function f(a,b) { return a+b; }
function g() { function h(){} }
x = g;
x = (function() {});

```

```

1: ADD x y _t1
2: ASSIGN _t1 z
3: FUNCSTART f
4: ADD a b _t1
5: RETURN _t1
6: FUNCEND f
7: FUNCSTART g
8: FUNCSTART h
9: FUNCEND h
10: FUNCEND g
11: ASSIGN g x
12: FUNCSTART _f1
13: FUNCEND _f1
14: ASSIGN _f1 x

```

Τα δύο αυτά \_t1 είναι διαφορετικά (σε διαφορετική εμβέλεια)

HY340
A. Σαββίδης
Slide 9 / 38

9

## Δηλώσεις συναρτήσεων (7/9)

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

```

void resetformalargsoffset(void)
{ formalArgOffset = 0; }

void resetfunctionlocalsoffset(void)
{ functionLocalOffset = 0; }

void restorecurrscopeoffset( unsigned n ) {
  switch( currscopeSpace() ) {
    case programvar : programVarOffset = n; break;
    case functionlocal : functionLocalOffset = n; break;
    case formalarg : formalArgOffset = n; break;
    default : assert(0);
  }
}

unsigned nextquadlabel( void )
{ return currQuad; }

void patchlabel( unsigned quadNo, unsigned label ) {
  assert(quadNo < currQuad);
  quads[quadNo].label = label;
}

```

HY340
A. Σαββίδης
Slide 10 / 38

10

## Δηλώσεις συναρτήσεων (8/9)

- Γιατί δεν κρατάμε πουθενά τον αριθμό των formal arguments των συναρτήσεων;
  - Γιατί ποτέ δεν απαιτείται στατικός έλεγχος 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
A. Σαββίδης
Slide 11 / 38

11

## Δηλώσεις συναρτήσεων (9/9)

```

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(){}),
  []
);


```

23  
TRUE  
hello

LibraryFunction("print")  
ProgramFunction("dumparguments", address n)  
ProgramFunction(anonymous, address x)  
Table[]

HY340
A. Σαββίδης
Slide 12 / 38

12




## Περιεχόμενα

- Δηλώσεις συναρτήσεων
- **Εκφράσεις αποθήκευσης**
- Βασικές εκφράσεις
- Επαναχρησιμοποίηση κρυφών μεταβλητών

HY340 A. Σαββίδης Slide 13 / 38

13




## Εκφράσεις αποθήκευσης (1/11)

- Πρόκειται για τα λεγόμενα *l-values*, δηλ. τις εκφράσεις που μπορούν να εμφανιστούν στο αριστερό τμήμα μίας εντολής εκχώρησης
  - αντιπροσωπεύοντας τον αποθηκευτικό χώρο του προγράμματος που έχει στη διάθεση του ο προγραμματιστής
  - γνωρίζουμε ότι υπάρχει και άλλος αποθηκευτικός χώρος, οι λεγόμενες κρυφές μεταβλητές, αλλά αυτές δεν είναι ποτέ στη διάθεση του προγραμματιστή
    - παρά το γεγονός ότι παράγουμε ενδιάμεσο κώδικα που τα χρησιμοποιεί ως *l-values* (δηλ. μπορούν να εμφανίζονται ως *result* για ASSIGN quads)

HY340 A. Σαββίδης Slide 14 / 38

14




## Εκφράσεις αποθήκευσης (2/11)

- Πριν μιλήσουμε για τα *l-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)
  - τιμές 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 A. Σαββίδης Slide 15 / 38

15



## 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)
    - *lvalue* → **id** | **local id** | **global id**
  - στοιχεία πίνακα
    - *lvalue* → *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 A. Σαββίδης Slide 16 / 38

16

### Εκφράσεις αποθήκευσης (4/11)

```

// Expression types. You use only the types you
// really need for i-code generation, so you may drop
// some entries
enum expr_t {
    var_e,
    tableitem_e,
    prograefunc_e,
    libraryfunc_e,
    arithexpr_e,
    boolexpr_e,
    assignexpr_e,
    newtable_e,
    costnum_e,
    constbool_e,
    conststring_e,
    nil_e
};

```

Γραμματικά σύμβολα με τύπο *expr*

- *lvalue*
- *member*
- *primary*
- *assignexpr*
- *call*
- *term*
- *objectdef*
- *const*

```

// For simplicity this is a superset type, but you may
// hack around with more clever storage (if you like).
struct expr {
    expr_t      type;
    symbol*     sym;
    int         index;
    double      numConst;
    char*       strConst;
    boolConst;
    unsigned char boolConst;
    expr*       next; // Just to make trivial e-lists.
};

```

HY340

A. Σαββίδης

Slide 17 / 38

17

### Εκφράσεις αποθήκευσης (5/11)

```

lvalue → id
{ ... $lvalue = lvalue_expr(sym); }
lvalue → local id
{ ... $lvalue = lvalue_expr(sym); }
lvalue → global id
{ ... $lvalue = lvalue_expr(sym); }

```

```

// Making an l-value expression out of a symbol is simple,
// since the expression inherits the symbol type. Also,
// getting information like 'library function' name or the
// 'program function' address (after code generation), is
// straightforward through the symbol 'sym' field.
expr* lvalue_expr (symbol* sym) {
    assert(sym);
    expr* e = (expr*) malloc(sizeof(expr));
    memset(e, 0, sizeof(expr));
    e->next = (expr*) 0;
    e->sym = sym;
    switch (sym->type) {
        case var_e:      e->type = var_e; break;
        case programfunc_e: e->type = programfunc_e; break;
        case libraryfunc_e: e->type = libraryfunc_e; break;
        default: assert(0);
    }
    return e;
}

```

HY340

A. Σαββίδης

Slide 18 / 38

18

### Εκφράσεις αποθήκευσης (6/11)

- Εκτός από τις μεταβλητές, χώρο αποθήκευσης λαμβάνουν και τα στοιχεία των δυναμικών πινάκων.
- Καθώς δεν αποτελούν «δηλωμένες» μεταβλητές, αλλά εκφράσεις που δυναμικά αντιστοιχούν σε χώρο αποθήκευσης, θέλουν ειδική μεταχείριση. Αυτό φαίνεται καλύτερα με δύο παραδείγματα

```

...
x = t.a.b.c;

```

|                    |     |    |                     |
|--------------------|-----|----|---------------------|
| 1: TABLEGETELEM t  | "a" | t1 | // i["a"]           |
| 2: TABLEGETELEM t1 | "b" | t2 | // i["a"]["b"]      |
| 3: TABLEGETELEM t2 | "c" | t3 | // i["a"]["b"]["c"] |
| 4: ASSIGN          | t3  | x  |                     |

Όταν είναι r-value, μπορεί να παράγω TABLEGETELEM εντολές για το κάθε index και να χρησιμοποιώ το αποτέλεσμα σαν το νέο στοιχείο.

```

...
t.a.b = x;

```

|                    |     |    |                |
|--------------------|-----|----|----------------|
| 1: TABLEGETELEM t  | "a" | t1 | // i["a"]      |
| 2: TABLESETELEM t1 | "b" | x  | // i["a"]["b"] |

Όταν όμως είναι l-value σε εκχώρηση, πρέπει να παράγω μία TABLESETELEM εντολή για το τελευταίο index.

HY340

A. Σαββίδης

Slide 19 / 38

19

### Εκφράσεις αποθήκευσης (7/11)

- *Η λύση*
  - Όταν έχω στοιχείο πίνακα, δεν παράγω αμέσως κώδικα για το συγκεκριμένο στοιχείο
  - αλλά το καταγράφω ως νέο expression «στοιχείο πίνακα» - `tableitem_e`, αποθηκεύοντας τόσο το `table` όσο και το `index` expression
  - Όταν χρησιμοποιείται ένα l-value που είναι τύπου «στοιχείο πίνακα»
    - ως επιμέρους έκφραση σε άλλο κανόνα (π.χ. `lvalue.id` ή `lvalue[expr]`) ή για αναγωγή σε r-value (δηλ. `primary`) τότε παράγουμε μία εντολή `TABLEGETELEM`
    - ως το αριστερό τμήμα μίας εκχώρησης (`lvalue=expr`), τότε παράγουμε μία εντολή `TABLESETELEM`

HY340

A. Σαββίδης

Slide 20 / 38

20



## Εκφράσεις αποθήκευσης (8/11)

Στην περίπτωση του έχω *hvalue.id*, και το *hvalue* είναι στοιχείο πίνακα, πρέπει να παράγουμε την εντολή που λαμβάνει αυτό το στοιχείο, καθώς το *hvalue* δεν χρησιμοποιείται το ίδιο για αποθήκευση. Αυτή η δουλειά γίνεται από τη συνάρτηση `emit_iftableitem` που θα δούμε αργότερα.

```
lvalue → tableitem
{ $lvalue = $tableitem; }

tableitem → lvalue . id
{ $tableitem = member_item($lvalue, id.name); }

expr* member_item (expr* lv, char* name) {
    lv = emit_iftableitem(lv); // Emit code if r-value use of table item
    expr* ti = newexpr(tableitem_e); // Make a new expression
    ti->sym = lv->sym;
    ti->index = newexpr_conststring(name); // Const string index
    return ti;
}
```

HY340

A. Σαββίδης

Slide 21 / 38

21



## Εκφράσεις αποθήκευσης (9/11)

```
tableitem → lvalue [ expr ]
{
    $lvalue = emit_iftableitem($lvalue);
    $tableitem = newexpr(tableitem_e);
    $tableitem->sym = $lvalue->sym;
    $tableitem->index = $expr; ← The index is the expression
}

primary → lvalue
{ $primary = emit_iftableitem($lvalue); }
```

HY340

A. Σαββίδης

Slide 22 / 38

22



## Εκφράσεις αποθήκευσης (10/11)

```
assignexpr → lvalue = expr {
    if $lvalue->type == tableitem_e then {
        emit( // lvalue[index] = expr
            tablesetelem,
            $lvalue,
            $lvalue->index,
            $expr ← Use result operand for the assigned value
        );
        $assignexpr = emit_iftableitem($lvalue); ← Will always emit
        $assignexpr->type = assignexpr_e;
    } else {
        emit( // that is: lvalue = expr
            assign,
            $expr,
            NULL,
            $lvalue
        );
        $assignexpr = newexpr(assignexpr_e);
        $assignexpr->sym = newtemp();
        emit(assign, $lvalue, NULL, $assignexpr);
    }
}
```

Actual argument values are not dependent on the involved variables.

Χρεώσουμε νέα κρυφή μεταβλητή καθώς το lvalue μπορεί να αλλάξει

To assignment αποθηκεύει το assigned value, ανεξάρτητα από την το expr

HY340

A. Σαββίδης

Slide 23 / 38

23



## Εκφράσεις αποθήκευσης (11/11)

```
expr* newexpr (expr* t) {
    expr* e = (expr*) malloc(sizeof(expr));
    memset(e, 0, sizeof(expr));
    e->type = t;
    return e;
}

expr* newexpr_conststring (char* s) {
    expr* e = newexpr(conststring_e);
    e->strconst = strdup(s);
    return e;
}

expr* emit_iftableitem (expr* e) {
    if (e->type != tableitem_e)
        return e;
    else {
        expr* result = newexpr(var_e);
        result->sym = newtemp();
        emit(
            tablegetelem,
            e,
            e->index,
            result
        );
        return result;
    }
}
```

|                           |                                                                                          |
|---------------------------|------------------------------------------------------------------------------------------|
| <code>x = p[a+b];</code>  | 1: ADD a b _t1<br>2: TABLEGETELEM p _t1 _t2<br>3: ASSIGN _t2 x                           |
| <code>y = q.a = x;</code> | 4: TABLESETELEM q "a" x<br>5: TABLEGETELEM q "a" t3<br>6: ASSIGN _t3 y<br>7: ASSIGN y t4 |


HY340

A. Σαββίδης

Slide 24 / 38

24






## Περιεχόμενα

- Δηλώσεις συναρτήσεων
- Εκφράσεις αποθήκευσης
- **Βασικές εκφράσεις**
- Επαναχρησιμοποίηση κρυφών μεταβλητών


HY340
A. Σαββίδης
Slide 25 / 38



## Βασικές εκφράσεις (1/9)

- Κλήση συναρτήσεων
- Δημιουργία πινάκων
- Ορισμός συνάρτησης σε παρένθεση
- Μοναδιαίοι τελεστές

HY340
A. Σαββίδης
Slide 26 / 38



## Βασικές εκφράσεις (2/9)

```
methodcall → ..id ( elist )
{
  $methodcall.elist = $elist;
  $methodcall.method = 1;
  $methodcall.name = $id.val;
}
```


```
call → call ( elist )
{
  $call = make_call($call, $elist);
  call → ( funcdef ) ( elist )
  {
    expr* func = newexpr(programfunc_e);
    func->sym = $funcdef;
    $call = make_call(func, $elist);
  }
}
```

```
expr* make_call (expr* lv, expr* reversed_elist) {
  expr* func = emit_iftableitem(lv);
  while (reversed_elist) {
    emit(param, reversed_elist, NULL, NULL);
    reversed_elist = reversed_elist->next;
  }
  emit(call, func, NULL, NULL);
  expr* result = newexpr(var_e);
  result->sym = newtemp();
  emit(getretval, NULL, NULL, result);
  return result;
}
```

**Κλήση συναρτήσεων (1/2)**

```
x = f(y) (a,b);
1: PARAM y
2: CALL f
3: GETRETVAL t1
4: PARAM b
5: PARAM a
6: CALL t1
7: GETRETVAL t2
8: ASSIGN t2 x
```

HY340
A. Σαββίδης
Slide 27 / 38



## Βασικές εκφράσεις (3/9)

```
call → lvalue callsuffix
{
  $lvalue = emit_iftableitem($lvalue); ←in case it was a table item too
  if ($callsuffix.method) {
    expr* t = $lvalue;
    $lvalue = emit_iftableitem(member_item(t, $callsuffix.name));
    $callsuffix.elist->next = t; ←insert as first argument (reversed, so last)
  }
  $call = make_call($lvalue, $callsuffix.elist);
}
```

```
callsuffix → normcall
{ $callsuffix = $normcall; }
callsuffix → methodcall
{ $callsuffix = $methodcall; }
normcall → ( elist )
{
  $normcall.elist = $elist;
  $normcall.method = 0;
  $normcall.name = NULL;
}
```

**Κλήση συναρτήσεων (2/2)**

```
sprite.move(dx,dy);
1: TABLEGETLEM sprite "move" t1
2: PARAM dy
3: PARAM dx
4: PARAM sprite
5: CALL t1
6: GETRETVAL t2
```

```
typeof:callsuffix> : struct call
typeof:normcall> : struct call
typeof:methodcall> : struct call
struct call {
  expr* elist;
  unsigned char method;
  char* name;
};
```

HY340
A. Σαββίδης
Slide 28 / 38

28

7



## Βασικές εκφράσεις (4/9)

### Δημιουργία πινάκων (1/2)

```
tablemake → [ elist ]
{
    expr* t = newexpr(newtable_e);
    t->sym = newtemp();
    emit(tablecreate, t, NULL, NULL);
    for (int i = 0; $elist; $elist = $elist->next)
        emit(tablesetelem, t, newexpr_constnum(i++), $elist);
    $tablemake = t;
}
```

```
x = [-4, 13, 12.34, a+b];
1: UMINUS 4 _t1
2: ADD a b _t2
3: TABLECREATE _t3
4: TABLESETELEM _t3 0 _t1
5: TABLESETELEM _t3 1 13
6: TABLESETELEM _t3 2 12.34
7: TABLESETELEM _t3 3 _t2
8: ASSIGN _t3 x
9: ASSIGN x _t4
```

HY340

A. Σαββίδης

Slide 29 / 38

29



## Βασικές εκφράσεις (5/9)

### Δημιουργία πινάκων (2/2)

```
tablemake → [ indexed ]
{
    expr* t = newexpr(newtable_e);
    t->sym = newtemp();
    emit(tablecreate, t, NULL, NULL);
    foreach <index, value> in $indexed do
        emit(tablesetelem, t, index, value);
    $tablemake = t;
}
```

Το γραμματικό σύμβολο `indexed` αντιστοιχεί σε τύπο δεδομένων πίνακα στο ζεύγος εκφράσεων.

```
t = [ ("x": 0), ("y": a+b), (-1: 0) ];
1: ADD a b _t1
2: UMINUS 1 _t2
3: TABLECREATE _t3
4: TABLESETELEM _t3 "x" 0
5: TABLESETELEM _t3 "y" _t1
6: TABLESETELEM _t3 _t2 0
7: ASSIGN _t3 t
```

HY340

A. Σαββίδης

Slide 30 / 38

30



## Βασικές εκφράσεις (6/9)

### Ορισμός συνάρτησης σε παρένθεση

```
primary → ( funcdef )
{
    $primary = newexpr(programfunc_e);
    $primary->sym = $funcdef;
}
```

```
x = (function(a,b) { return a+b; });
Πως μπορούμε να υποστηρίξουμε κάτι τέτοιο? Με τους κανόνες που ήδη έχουμε παρουσιάσει για call!
```

```
(function() {} ) ();
6: FUNCSTART _f1
7: FUNCEND _f1
8: CALL _f1
9: GETRETV _t1
```

HY340

A. Σαββίδης

Slide 31 / 38

31



## Βασικές εκφράσεις (7/9)

### Μοναδιαίοι τελεστές (1/3)

```
term → ( expr ) { $term = $expr; }
term → primary { $term = $expr; }
```

```
term → - expr
{
    check_arith($expr);
    $term = newexpr(arithexpr_e);
    $term->sym = newtemp();
    emit(uminus, $expr, NULL, $term);
}
```

```
term → not expr ← ολική αποτίμηση bool exprs
{
    $term = newexpr(boolexpr_e);
    $term->sym = newtemp();
    emit(not, $expr, NULL, $term);
}
```

```
void comperror (char* format, ...);
// Use this function to check correct use of
// of expression in arithmetic
void check_arith (expr* e, const char* context) {
    if ( e->type == constbool_e ||
        e->type == conststring_e ||
        e->type == nil_e ||
        e->type == newtable_e ||
        e->type == programfunc_e ||
        e->type == libraryfunc_e ||
        e->type == boolexpr_e )
        comperror("Illegal expr used in %s!", context);
}
```

HY340

A. Σαββίδης

Slide 32 / 38

32



## Βασικές εκφράσεις (8/9)

**Μοναδιαίοι τελεστές (2/3)**

```

term → lvalue++
{
    check_arith($lvalue);
    $term = newexpr(var_e);
    $term->sym = newtemp();
    if ($lvalue->type == tableitem_e) {
        $expr* val = emit_iftableitem($lvalue);
        emit(assign, val, NULL, $term);
        emit(add, val, newexpr_constnum(1), val);
        emit(tablesetelem, $lvalue, $lvalue->index, val);
    }
    else {
        emit(assign, $lvalue, NULL, $term);
        emit(add, $lvalue, newexpr_constnum(1), $lvalue);
    }
}

```

|                         |                                                                                                               |
|-------------------------|---------------------------------------------------------------------------------------------------------------|
| <code>a = x--;</code>   | 1: ASSIGN x _t0<br>2: SUB x 1 x<br>3: ASSIGN _t0 a                                                            |
| <code>a = t.x++;</code> | 4: TABLETELEM t "x" _t2<br>5: ASSIGN _t2 t1<br>3: ADD _t2 1 _t2<br>4: TABLETELEM t "x" _t2<br>5: ASSIGN _t1 a |

HY340
A. Σαββίδης
Slide 33 / 38

33

## Βασικές εκφράσεις (9/9)

**Μοναδιαίοι τελεστές (3/3)**

```

term → ++lvalue
{
    check_arith($lvalue);
    if ($lvalue->type == tableitem_e) {
        $term = emit_iftableitem($lvalue);
        emit(add, $term, newexpr_constnum(1), $term);
        emit(tablesetelem, $lvalue, $lvalue->index, $term);
    }
    else {
        emit(add, $lvalue, newexpr_constnum(1), $lvalue);
        $term = newexpr(arithexpr_e);
        $term->sym = newtemp();
        emit(assign, $lvalue, NULL, $term);
    }
}
term → --lvalue { Σημειώστε το! }
term → lvalue-- { Σημειώστε το! }

```

|                           |                                                                                                                             |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <code>++x;</code>         | 1: ADD x 1 x<br>2: ASSIGN x _t0                                                                                             |
| <code>a = ++t.x;</code>   | 2: TABLETELEM t "x" _t0<br>3: ADD _t0 1 _t0<br>4: TABLESETELEM t "x" _t0<br>5: ASSIGN _t0 a                                 |
| <code>a = --t.x.y;</code> | 6: TABLETELEM t "x" _t1<br>7: TABLESETELEM _t1 "y" _t2<br>8: SUB _t2 1 _t2<br>9: TABLETELEM _t1 "y" _t2<br>10: ASSIGN _t2 a |

Χρησιμοποιήστε νέα κρυφή μεταβλητή καθώς πρόκειται για assignment

HY340
A. Σαββίδης
Slide 34 / 38

34

## Περιεχόμενα

- Δηλώσεις συναρτήσεων
- Εκφράσεις αποθήκευσης
- Βασικές εκφράσεις
- **Επαναχρησιμοποίηση κρυφών μεταβλητών**

HY340
A. Σαββίδης
Slide 35 / 38

35

## 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
A. Σαββίδης
Slide 36 / 38

36



## Επαναχρησιμοποίηση κρυφών μεταβλητών (2/3)

- Η δυνατότητα επαναχρησιμοποίησης φαίνεται και στο παρακάτω παράδειγμα
  - Όποτε έχω χρήση κρυφών μεταβλητών ως r-values, τότε μπορώ να χρησιμοποιήσω αυτή τη μεταβλητή και πάλι για προσωρινή αποθήκευση
- Η αναγνώριση των κρυφών μεταβλητών είναι εύκολη βάσει του ονόματος

```
unsigned int istempname (char* s) {  
    return *s == '_';  
}  
unsigned int istempexpr (expr* e) {  
    return e->sym && istempname(e->sym->name);  
}
```

|                                                                                             |                                                       |
|---------------------------------------------------------------------------------------------|-------------------------------------------------------|
| <b>a = x+y+z;</b><br><i>(όχι r-value)</i>                                                   | 1: ADD x y _t1<br>2: ADD _t1 z _t2<br>3: ASSIGN _t2 a |
| Στο 1: το _t1 είναι αποτέλεσμα, ενώ στο 2: είναι όρισμα, άρα μπορεί να επαναχρησιμοποιηθεί. | 1: ADD x y _t1<br>2: ADD _t1 z _t1<br>3: ASSIGN _t1 a |

HY340

A. Σαββίδης

Slide 37 / 38

37



## Επαναχρησιμοποίηση κρυφών μεταβλητών (3/3)

### Σημεία επαναχρησιμοποίησης (προαιρετικά)

- Κλήση συνάρτησης. Από το lvalue, call, ή κάποιο από τα elist
- Κατασκευή πίνακα. Από κάποιο από τα elist ή από κάποιο <index, expr>
- Μοναδιαίο μείον. Από το expr.
- Λογική άρνηση. Από το expr.
- Αριθμητικές εκφράσεις (δυαδικοί τελεστές). Από οποιαδήποτε εκ των δύο expr.
- Λογικές εκφράσεις (δυαδικοί τελεστές). Από οποιαδήποτε εκ των δύο expr.

### Η επέκταση ενός σημειολογικού κανόνα (unary minus) για επαναχρησιμοποίηση κρυφής μεταβλητής

```
term → - expr  
{  
    check_arith($expr);  
    $term = newexpr(arithexpr_e);  
    $term->sym = istempexpr($expr) ? $expr->sym : newtemp();  
    emit(uminus,$expr, NULL, $term);  
}
```

*Μόνο αυτό προσθέτουμε*

HY340

A. Σαββίδης

Slide 38 / 38

38