ΗΥ340 : 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

Φροντιστήριο 3ο
Symbol Table & Scopes

Τι είναι ο Symbol Table

- Ο πίνακας συμβόλων (symbol table) είναι μία δομή, όπου αποθηκεύεται πληροφορία σχετικά με τα σύμβολα του προγράμματος
  - *Είδος συμβόλου* (μεταβλητή ή συνάρτηση)
  - *Scope* (εμβέλεια) του συμβόλου
- Ο πίνακας συμβόλων προσπελαύνεται κάθε φορά που ο compiler συναντά κάποιο σύμβολο
  - *Ορισμός νέων συμβόλων*
  - *Εύρεση υπαρχόντων συμβόλων*

Δεδομένα Symbol Table (1/2)

- Μεταβλητές
  - Όνομα
  - Τύπος
    - *Καθολική μεταβλητή*, *τυπικό όρισμα* ή *τοπική μεταβλητή*
  - Εμβέλεια (scope)
  - Γραμμή δήλωσης
- Συναρτήσεις
  - Όνομα
  - Τύπος
    - Συνάρτηση προγραμματιστή, ή βιβλιοθήκης
  - Τυπικά ορίσματα
  - Εμβέλεια (scope)
  - Γραμμή δήλωσης

Δεδομένα Symbol Table – Παράδειγμα (2/2)

```
1.  x = input();
2.  function g(x, y) {
3.      local z = x + y;
4.      print(z);
5.      return(function f(a, b) {
6.          return a + b;
7.      });
8.  }
```

- x: global variable at scope 0, line 1
- input: library function at scope 0, line 0
- g: user defined function with arguments x, y, at scope 0, line 2
- x, y: formal arguments at scope 1, line 2
- z: local variable at scope 1, line 3
- print: library function at scope 0, line 0
- f: user defined function with arguments a, b, at scope 1, line 5
- a, b: formal arguments at scope 2, line 5

Λειτουργίες Symbol Table (1/5)

- *Insert*
  - Εισαγωγή ενός νέου συμβόλου στον πίνακα
- *Lookup*
  - Αναζήτηση ενός συμβόλου στον πίνακα
- *Hide*
  - Απενεργοποίηση (όχι διαγραφή) όλων των συμβόλων ενός επιπέδου εμβέλειας

Λειτουργίες Symbol Table – Insert (2/5)

- Όποτε αναγνωρίζεται ένα σύμβολο δημιουργείται μία νέα εγγραφή για αυτό, εφόσον δεν υπάρχει ήδη στον symbol table
- Πότε γίνεται;
  - Κατά τον ορισμό νέας μεταβλητής, π.χ. *local x;*
    - Εκτός φυσικά αν αναφέρεται σε μεταβλητή ή συνάρτηση στο ίδιο scope
  - Κατά τον ορισμό νέας συνάρτησης, π.χ. *function foo(x, y) { return x+y;}*
  - Κατά την χρήση μη ορισμένου συμβόλου, πχ. *x = input();*

Λειτουργίες Symbol Table – Lookup (3/5)

- Αναζήτηση ενός συμβόλου στο τρέχον επίπεδο εμβέλειας ή σε περιέχουσα εμβέλεια με βάση τους κανόνες που ακολουθεί η γλώσσα
- Πότε γίνεται;
  - Κατά τον ορισμό ενός συμβόλου
    - Για να ελέγξουμε ότι δεν υπάρχει σύγκρουση με κάποιο ήδη υπάρχον σύμβολο
  - Κατά την χρήση ενός συμβόλου
    - Για να βρούμε το σύμβολο στο οποίο αναφερόμαστε
  - Ουσιαστικά λοιπόν γίνεται όποτε συναντήσουμε κάποιο σύμβολο

Λειτουργίες Symbol Table – Hide (4/5)

- Απενεργοποίηση των μεταβλητών κάποιου επιπέδου εμβέλειας (συνήθως του τρέχοντος)
- Πότε γίνεται;
  - Κατά την έξοδο από κάποιο block
  - Κατά την έξοδο από συνάρτηση

Λειτουργίες Symbol Table – Παράδειγμα (5/5)

```
x = input();
g = 12.4;
print(typeof(x));
function foo(x, y) {
    print(x + y);
    local p = y;
    ::print(p);
    function h(a) {
        return a + x + y;
    }
    y = h(::x);
}
```

lookup(*x*), ins(*x*), lookup(*input*)

lookup(*g*), ins(*g*)

lookup(*print*), lookup(*typeof*), lookup(*x*)

lookup(*foo*), ins(*foo*), lookup(*x*), ins(*x*), lookup(*y*), ins(*y*)

lookup(*print*), lookup(*x*), lookup(*y*)

lookup(*p*), ins(*p*), lookup(*y*)

lookup(*::print*), lookup(*p*)

lookup(*h*), ins(*h*), lookup(*a*), ins(*a*)

lookup(*a*) , lookup(*x*), ***error***

hide(*a*)

lookup(*y*), lookup(*h*), lookup(*::x*)

hide(*foo::x*, *foo::y*, *foo::p*, *foo::h*)

Symbol Definitions (1/2)

- Symbol redefinition:
  - Όταν υπάρχει ήδη μια μεταβλητή με id ίδιο με το id της μεταβλητής που πάμε να ορίσουμε και βρίσκεται στο ίδιο scope
    - Μεταβλητές ίδιου scope με ίδιο όνομα (ok)
    - Συναρτήσεις ίδιου scope με ίδιο όνομα **(error)**
    - Συνάρτηση και μεταβλητή ίδιου scope με ίδιο όνομα **(error)**
    - Οποιοδήποτε σύμβολο με όνομα ίδιο με συνάρτηση βιβλιοθήκης **(error)**
- Undefined symbol:
  - Όταν μια μεταβλητή που χρησιμοποιείται δεν είναι στον πίνακα συμβόλων
    - Οι μεταβλητές ορίζονται με την πρώτη εμφάνισή τους (εισάγονται στον symbol table) (ok)
    - Όταν ζητάμε global μεταβλητή η οποία δεν έχει οριστεί και άρα δεν υπάρχει στον symbol table **(error)**

Symbol Definitions – Παράδειγμα (2/2)

```
read(a);
a = input();
function foo(x) {
    local print = “hello”;
    print(x + y);
    local hello = a;
    local hello = 35;
    function hello() {
        print(::a + ::x);
    }
    function cos(){}
}
function foo(){}
```

New variables: *read*, *a*

Function definition: *foo,* New variable: *x*

Collision with library function: print **(error)**

New variable: *y*

New variable: *hello*

Same variable hello

Redeclaration of *hello* as function **(error)**

Global *a* found but global *x* not found **(error)**

Collision with library function: cos **(error)**

Redeclaration of function: *foo* **(error)**

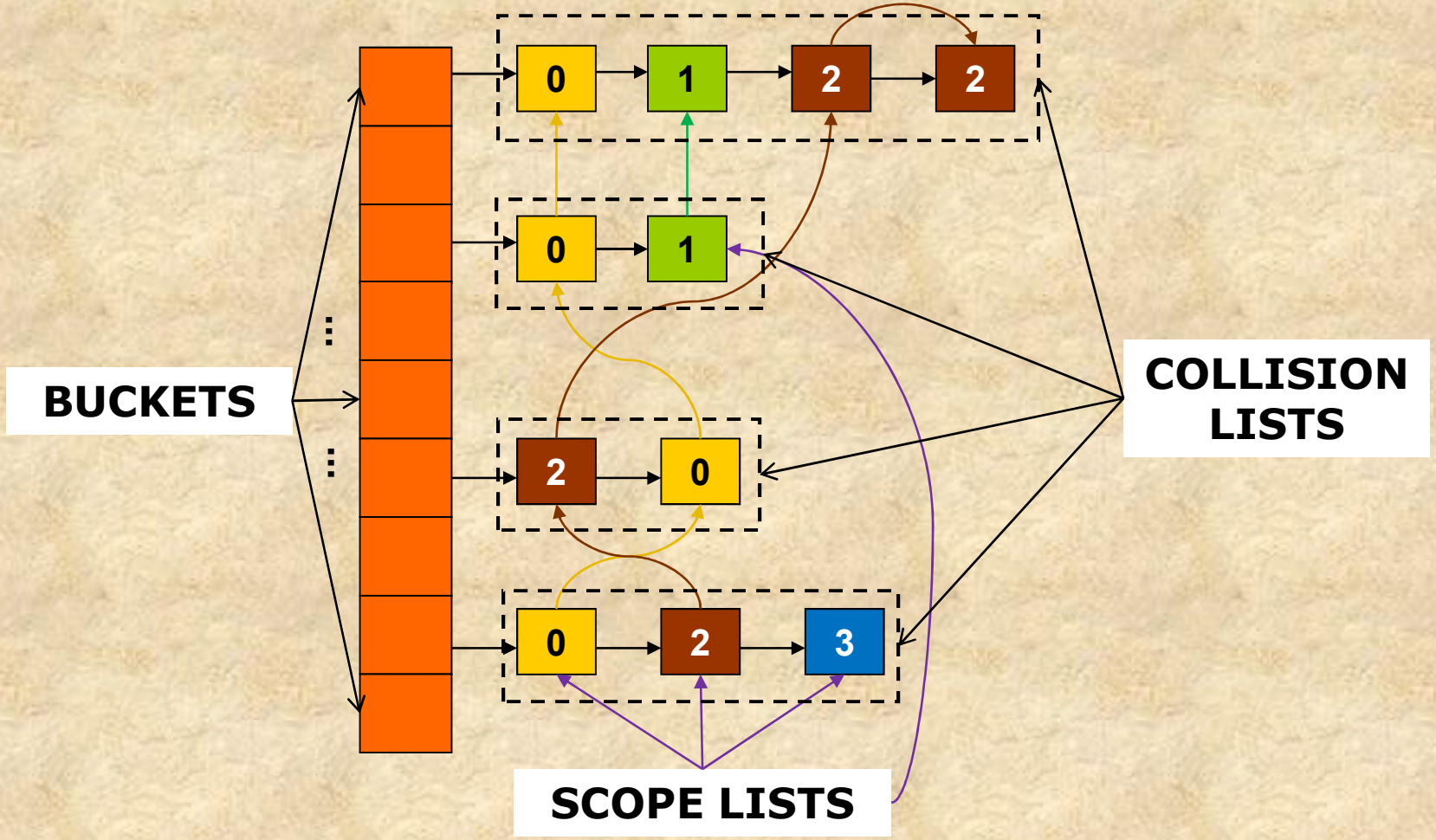
Υλοποίηση του Symbol Table

- Υπάρχουν κυρίως δύο τρόποι υλοποίησης του πίνακα συμβόλων:
  - Με Linked List
  - Με Hash Table
- Προτιμάμε την υλοποίηση με Hashtable κυρίως λόγω της επίδοσης του Lookup

Υλοποίηση Symbol Table με Hash Table (1/2)

- Ένας Hashtable αποτελείται από buckets
- Κάθε σύμβολο κατανέμεται σε ένα από αυτά τα buckets με βάση μια hash συνάρτηση
- Σύμβολα που τυχαίνει να πέσουν στο ίδιο bucket δημιουργούν μια linked list
- Επιπλέον είναι χρήσιμο να υπάρχει και ένα scope link που δημιουργεί μία λίστα συνδέοντας όλα τα σύμβολα που ανήκουν στο ίδιο scope
  - Αυτή η λίστα μπορεί να είναι μία δομή πάνω στην ήδη υπάρχουσα ή και ξεχωριστή (απλά πρέπει να ανανεώνεται παράλληλα με την κύρια δομή)

Υλοποίηση Symbol Table με Hash Table (2/2)



Υλοποίηση λειτουργιών Symbol Table

- Οι συναρτήσεις Insert και Lookup είναι όμοιες με αυτές που έχετε υλοποιήσει σε προηγούμενα μαθήματα για Hashtables
  - Πρέπει όμως να γίνει και ο χειρισμός του scope link
  - Επιπλέον, είναι πιθανό να χρειαστείτε δύο εκδόσεις για τη lookup
    - Αναζήτηση ενός συμβόλου σε οποιοδήποτε scope
    - Αναζήτηση ενός συμβόλου σε συγκεκριμένο scope
- Η συνάρτηση Hide θα «ακολουθεί» το scope link για μία συγκεκριμένη εμβέλεια και θα ακυρώνει αυτά τα σύμβολα

Symbol Table Entries

```
typedef struct Variable {
    const char *name;
    unsigned int scope;
    unsigned int line;
} Variable;

typedef struct Function {
    const char *name;
    //List of arguments
    unsigned int scope;
    unsigned int line;
} Function;
```

```
enum SymbolType {
    GLOBAL,  LOCAL, FORMAL,
    USERFUNC, LIBFUNC
};

typedef struct SymbolTableEntry {
    bool isActive;
    union {
        Variable *varVal;
        Function *funcVal;
    } value;
    enum SymbolType type;
} SymbolTableEntry;
```

Χώροι Εμβέλειας

- Ένας τρόπος χειρισμού των χώρων εμβέλειας (scopes) είναι να υπάρχει μία καθολική μεταβλητή που να δείχνει την εμβέλεια στην οποία βρισκόμαστε.
- Η μεταβλητή αυτή θα πρέπει να αυξάνεται ή να μειώνεται στα actions των κατάλληλων γραμματικών κανόνων
  - Π.χ. *block : '**{**' {++scope;} stmts '**}**' { Hide(scope--); }*
  - *Function definitions*
  - *Όχι όμως στα indexed elements ενός πίνακα*

Χώροι Εμβέλειας – Μεταβλητές (1/2)

- ***local x***
  - Κάνουμε lookup στο SymbolTable στο **τρέχον** scope
  - Αν βρεθεί κάτι (είτε μεταβλητή είτε συνάρτηση) αναφερόμαστε εκεί
  - Αν δε βρεθεί κάτι και δεν υπάρχει collision με library function προσθέτουμε στο SymbolTable νέα μεταβλητή στο τρέχον scope
    - Αν είμαστε σε scope 0, η μεταβλητή δηλώνεται global (αγνοείται το local)
- ***::x (global x)***
  - Κάνουμε lookup στο SymbolTable στο **global** scope
  - Αν βρεθεί κάτι (είτε μεταβλητή είτε συνάρτηση) αναφερόμαστε εκεί, αλλιώς είναι error
  - Το ::x δεν κάνει **ποτέ** εισαγωγή στο SymbolTable

```
function f() {}
local f;       //ok, found locally
local print; //ok, found in scope 0
local x; //new global var in scope 0
{
   local f; //new local var in scope 1
   local print; //error: trying to
                //shadow libfunc
}
function g(a, b) { local a = local b; }
//ok, a and b found locally
```

```
x; //new global var x
print(::x); //ok, x found
print(::y); //error: no y
function f() { return ::x} //ok
{ print(::f()); } //ok, f found
```

Χώροι Εμβέλειας – Μεταβλητές (2/2)

- ***x (variable x)***
  - Κάνουμε lookup στο SymbolTable ξεκινώντας από το τρέχον scope και πηγαίνωντας μέχρι το global
    - Η αναζήτηση γίνεται από μέσα προς τα έξω
  - Αν δε βρεθεί τίποτα σε κανένα scope, τότε προσθέτουμε στο SymbolTable νέα μεταβλητή στο τρέχον scope
  - Αν βρεθεί σε κάποιο scope κάτι (είτε μεταβλητή είτε συνάρτηση) αναφερόμαστε εκεί και μένει να δούμε αν έχουμε πρόσβαση στο σύμβολο
- *Προσβασιμότητα συμβόλων*
  - Όλα τα ορατά σύμβολα που ορίζονται είτε μέσα στο τρέχον scope είτε σε κάποιο εξωτερικότερο scope ***χωρίς όμως να μεσολαβεί συνάρτηση*** είναι προσβάσιμα
    - Π.χ. αν βρισκόμαστε σε συνάρτηση και αναφερόμαστε σε τοπική μεταβλητή ή τυπικό όρισμα κάποιας περιέχουσας συνάρτησης δεν έχουμε πρόσβαση
  - Όλες οι ορατές συναρτήσεις, τόσο στο τρέχον scope όσο και σε εξωτερικότερα είναι ορατές (άσχετα με το αν μεσολαβεί συνάρτηση ή όχι)
  - Όλες οι καθολικές μεταβλητές (σε scope 0) και οι συναρτήσεις βιβλιοθήκης είναι προσβάσιμες

```
x = y = 1;  //new variables x, y
{
  x = 2; //ok, refers to global x
  a = 3; //new var a in scope 1
  function f (z) {
    x = 4; //ok, refers to global x
    a = 5; //error: cannot access
           //a in scope 1
    y = 6; //ok, refers to global y
    {
      z = 7; //ok, refers to formal
      function g() { return z; }
      //error: cannot access z
    }
  }
}
```

Χώροι Εμβέλειας – Συναρτήσεις (1/2)

- ***funcdef – function f(…) {}***
  - Αν η συνάρτηση δεν έχει όνομα της δίνουμε εμείς ένα μοναδικό όνομα που δε μπορεί να δοθεί από το χρήστη (π.χ. _f1, $f2, κλπ)
    - Κατά τα άλλα μπαίνει κανονικά στο SymbolTable
  - Κάνουμε lookup στο **τρέχον** scope
  - Αν βρεθεί μεταβλητή ή συνάρτηση είναι error
  - Αν υπάρχει collision με libfunc είναι error
  - Αλλιώς εισάγουμε νέα συνάρτηση στο τρέχον scope
  - ***Μόνο*** αυτός ο κανόνας δηλώνει συναρτήσεις
    - Προσοχή: αν έχουμε *x(1);* το x ***δεν*** είναι συνάρτηση αλλά μεταβλητή!
- ***formal arguments – function f(x) {}***
  - Κάνουμε lookup στο **τρέχον** scope (αυτό της συνάρτησης που δηλώνεται)
  - Αν βρεθεί κάτι είναι error
  - Αν υπάρχει collision με libfunc είναι error
  - Αλλιώς εισάγουμε νέο τυπικό όρισμα στο τρέχον scope

```
function f() { // f in scope 0
  function f() {}// f in scope 1
  function f() {}//error: f exists
}
x = 1;
function x() {} //error: x is var
function sin() {} //error: func
//shadows libfunc
y(1, 2); //new variable y, not
         //function y
```

```
function f(x, y, z) {}
//ok, f with formals x, y, z
function g(x, y, x) {} //error:
//formal redeclaration
function h(x, cos) {} //error:
//formal shadows libfunc
```

Χώροι Εμβέλειας – Συναρτήσεις (2/2)

- Οι συναρτήσεις (είτε χρήστη είτε βιβλιοθήκης) είναι σταθερές και δε μπορούν να αλλάξουν τιμή
- Συνεπώς, για μια συνάρτηση f απαγορεύονται ενέργειες όπως
  - f = 1, ++f, f--
- Στον κανόνα του lvalue όμως (lvalue→id) που κάνουμε το lookup και βρίσκουμε π.χ. ότι αναφερόμαστε σε συνάρτηση δεν ξέρουμε ακόμα τον τρόπο χρήσης
  - Μπορεί να είναι το λάθος f = 1
  - Αλλά και το σωστό f(1)
- Ο έλεγχος γίνεται στους *επιμέρους κανόνες* και όχι στο lvalue
  - Στον κανόνα lvalue→id απλά αποθηκεύουμε την πληροφορία του συμβόλου που βρήκαμε στο $$
    - Χρειάζεται να προσθέσουμε στο union του yacc τον κατάλληλο τύπο
      - **%union { SymbolTableEntry* exprNode; … } %type <exprNode>** lvalue
  - Αργότερα, στη χρήση του lvalue χρησιμοποιούμε το αποθηκευμένο σύμβολο για να ελέγξουμε τον τρόπο χρήσης
    - Π.χ. στο assignexpr → lvalue '=' expr αν το σύμβολο που περιέχεται στο $1 είναι συνάρτηση βγάζουμε το κατάλληλο μήνυμα λάθους