How to Create a Programming Language

LECTURE 7B: SYMBOL MANAGEMENT

1

LECTURE OUTLINE

- ADDITIONAL CONCEPTS
- SYMBOL MANAGEMENT
- TABLE DESIGN AND STRUCTURES
- MANAGING SCOPE

2

ADDITIONAL CONCEPTS

- **ALIAS** – TWO DIFFERENT SYMBOLIC NAMES REFER TO SAME OBJECT
  - NO ADDITIONAL COMPLEXITY FOR SYMBOL MANAGEMENT
  - COMPLICATES CODE OPTIMIZATION
- **OVERLOADING** – A NAME CAN REFER TO MORE THAN ONE OBJECT AT SOME POINT
  - TYPICALLY APPLIES TO FUNCTIONS OR OPERATORS
  - ADDS COMPLEXITY TO SYMBOL MANAGEMENT

3

ADDITIONAL CONCEPTS

- **COERCION** – AUTOMATIC CONVERSION OF VARIABLE TYPE
- **POLYMORPHISM** – SINGLE FUNCTION OR OPERATION THAT CAN ACCEPT DIFFERENT ARGUMENT TYPES
  - **PARAMETRIC POLYMORPHISM**
  - **SUBTYPE POLYMORPHISM**

4

EXAMPLES

Alias:

```
#include <stdio.h>
void compute(int x[], int y[], int valcount) {
   int idx;
   for (idx = 0; idx < valcount; idx++) {
      y[idx] = x[idx] * y[idx];
   }
}
int main (int argc, char **argv) {
   int idx;
   int arr1[5] = {1, 3, 5, 7, 9};
   int *arr2;
   arr2 = &arr1[1];
   compute(arr1, arr2, 5);
   for (idx = 0; idx < 5; idx++) {
      printf ("idx=%d, val=%d\n", idx, arr2[idx]);
    }
}
```


5

EXAMPLES

Overloading:

complex.h:

```
class complex {
private:
  int r;
  int i;

public:
…
```

complex.cpp:

```
…
complex complex::operator+(complex b) {
  complex retval;
  retval.r = r + b.r;
  retval.i = i + b.i;
   return retval;
}
…
```


6

EXAMPLES

Overloading (continued):

Usecomplex.cpp:

```
  int ai = 5;
  int bi = 10;
  int ci;

  complex ac;
  complex bc;
  complex cc;

  ci = ai + bi;

  ac = complex(5, 10);
  bc = complex(10, 15);
  cc = ac + bc;
```


7

EXAMPLES

Coercion:

```
#include <stdio.h>

int main(int argc, char **argv)
{
  int a = 5;
  float b = 24.0;
  float c;

  c = b / a;  // Automatically 'converts' 'a' to be a float variable
}
```


8

EXAMPLES

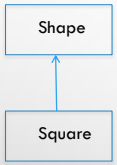
Parametric Polymorphism:

```
#include <iostream>
using namespace std;

template <class MyClass>
MyClass MaxOf(MyClass first, MyClass second) {
  if (first >= second) {
    return first;
  } else {
    return second;
  }
}
int main(int argc, char **argv) {
  int a = 5, b = 10, y;
  float c = 5.2, d = -2.13, z;

  y = MaxOf<int>(a,b);
  z = MaxOf<float>(c,d);
  cout << "Max integer: " << y << endl;
  cout << "Max float: " << z << endl;
}
```


9

EXAMPLES

Subtype Polymorphism:




```
void draw(Shape s);

...

Square mySquare;

draw(mySquare);  // This is valid since a square is a shape!
```


10

SYMBOL MANAGEMENT

- BASIC SYMBOL TABLE NEEDS:
  - A FUNCTION TO INSERT SYMBOLS AND ADD ATTRIBUTES TO SYMBOLS
  - A FUNCTION TO LOOKUP SYMBOLS
  - A FUNCTION TO 'ENTER' A NEW SCOPE
  - A FUNCTION TO 'EXIT' A SCOPE
- COMPLEXITY OF SYMBOL MANAGEMENT AFFECTED BY LANGUAGES:
  - BINDING RULES
  - SCOPE RULES

11

COMPLEXITY OF SYMBOL MANAGEMENT

- INNER SCOPE VARIABLE MAY 'HIDE' AN OUTER SCOPE VARIABLE WITH THE SAME NAME
- RECORDS CONTAIN FIELDS:
  - INVOKE A 'SCOPE' FOR FIELDS COMPRISING THE RECORD
  - 'WITH' STATEMENTS EXTEND RECORD SCOPE FOR MULTIPLE STATEMENTS
- FORWARD REFERENCES
- MAY NEED TO KEEP INFO FOR A DEBUGGER

12

TABLE DESIGN AND STRUCTURES

- STATIC SCOPE MANAGEMENT
  - LEBLANC-COOK
- DYNAMIC SCOPE MANAGEMENT
  - ASSOCIATION LISTS
  - CENTRAL REFERENCE TABLE

13

LEBLANC-COOK SYMBOL MANAGEMENT

- SCOPES ARE NUMBERED:
  - 0 – LANGUAGE PRE-DEFINED OPERATIONS/FUNCTIONS
  - 1 – USER DEFINED GLOBALS
  - 2-> - FUNCTION/RECORD SCOPES
- EACH SCOPE HAS UNIQUE NUMBER (NUMBERS ARE NOT LEXICAL LEVEL)
- HASH TABLE KEYED BY SYMBOL NAME
  - ENTRY CONTAINS NAME, SCOPE, CATEGORY, TYPE, ETC
  - SCOPE STACK INDICATES WHICH SCOPES ARE ACTIVE

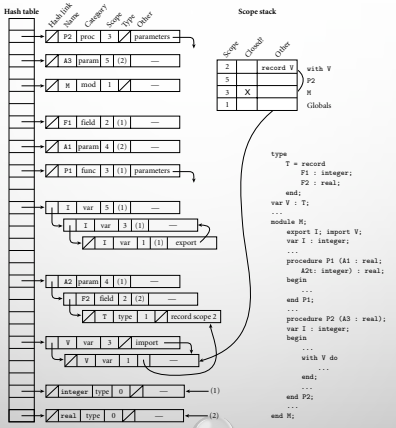
14

LEBLANC-COOK SYMBOL TABLE LOOKUP

```
procedure lookup(name)
  pervasive := best := null
  apply hash function to name to find appropriate chain
  foreach entry e on chain
    if e.name = name -- not something else with same hash value
      if e.scope = 0
        pervasive := e
      else
        foreach scope s on scope stack, top first
          if s.scope = e.scope
            best := e  -- closer instance
            exit inner loop
          elsif best != null and then s.scope = best.scope
            exit inner loop      -- won't find better
          if s.closed
            exit inner loop      -- can't see farther
  if best != null while best is an import or export entry
    best := best.real entry
  return best
  elsif pervasive != null
    return pervasive
  else
    return null  -- name not found
```


15

SYMBOL TABLE EXAMPLE



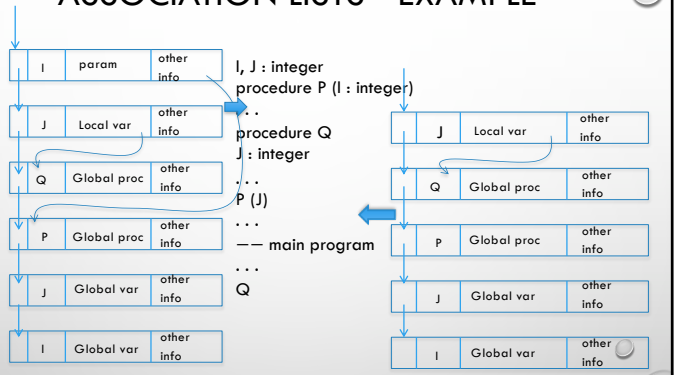
16

ASSOCIATION LISTS

- SIMPLE
- INEFFICIENT (FOR LOOKUPS)
- METHOD:
  - SYMBOLS LINKED ON A SINGLE LIST
  - INNER SCOPES ARE CLOSER TO FRONT OF LIST
  - LIST SEARCHED FRONT TO BACK (INNER TO OUTER)
- SINGLE LIST CAN GET LONG AFFECTING SEARCH TIME FOR SYMBOLS AT OUTER SCOPE(S)

17

ASSOCIATION LISTS - EXAMPLE



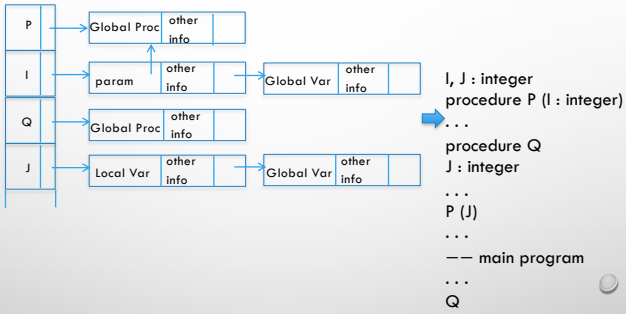
18

CENTRAL REFERENCE TABLES

- SIMILAR TO LEBLANC-COOK BUT WITHOUT SCOPE STACK
- EACH SYMBOL NAME HAS ITS OWN 'CHAIN'
- INNER SCOPES AT THE FRONT OF EACH CHAIN
- MORE EXPENSIVE FOR SCOPE ENTRY/EXIT
- LOOKUPS ARE FASTER SINCE CHAIN ONLY HOLDS SINGLE SYMBOL NAME

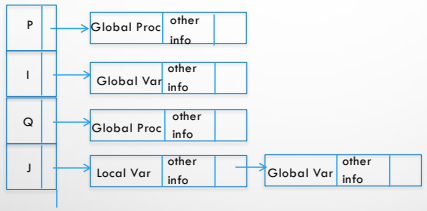
19

CENTRAL REFERENCE TABLES - EXAMPLE



20

CENTRAL REFERENCE TABLES - EXAMPLE




```
I, J : integer
procedure P (I : integer)
. . .
procedure Q
J : integer
. . .
P (J)
. . .
-- main program
. . .
Q
```


21

QUESTIONS

22