CSE304/504 Compiler Design Implementing a Symbol Table

Lecture Outline

Additional Concepts

Symbol Management

Table Design and Structures

Managing Scope

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

Additional Concepts

Coercion – Automatic conversion of variable type

Polymorphism – Single function or operation that can accept different argument types

- **Parametric Polymorphism**
- **Subtype Polymorphism**

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]);
     }
}
```

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;
}
…
```

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;
```

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
}
```

Examples

Parametric Polymorphism:

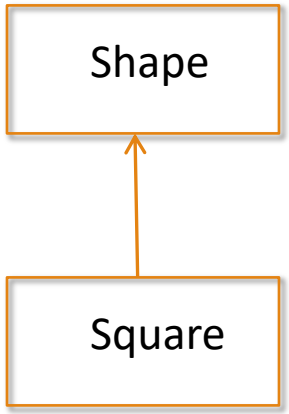
```
#include <iostream>
using namespace std;

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;
}
```

Examples

Subtype Polymorphism:




```
void draw(Shape s);

…

Square mySquare;

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

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

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

Table Design and Structures

Static scope management

- Leblanc-Cook

Dynamic scope management

- Association Lists
- Central Reference Table

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)

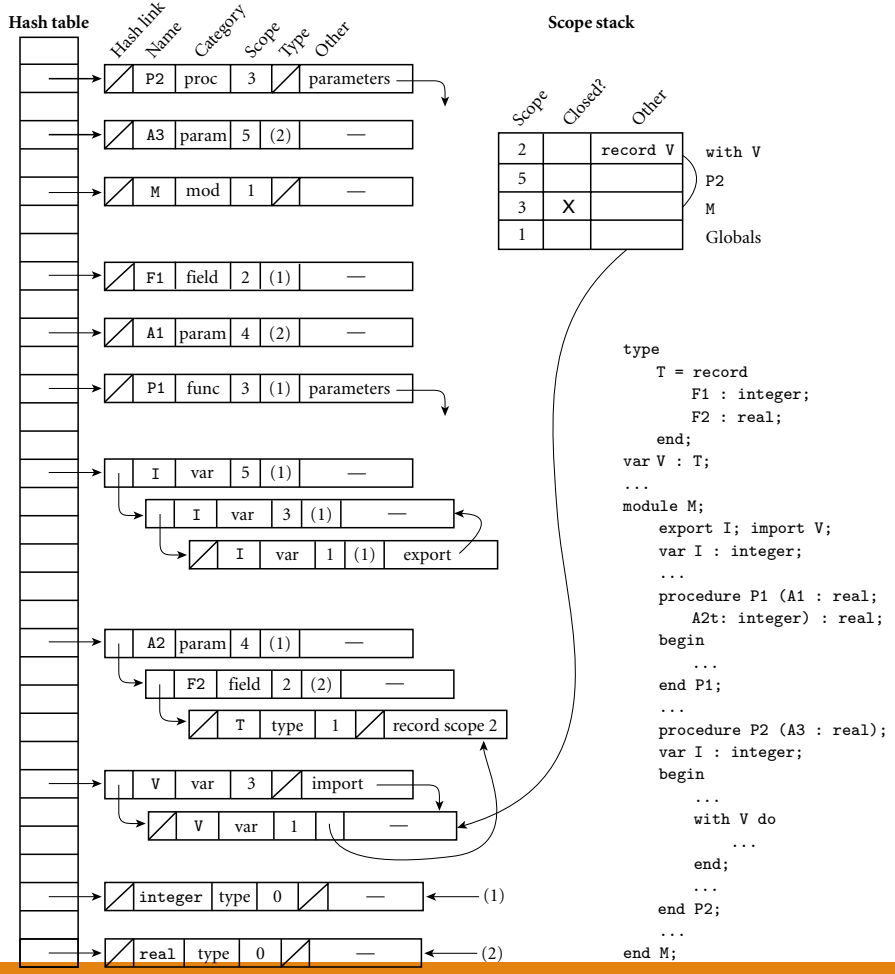
Hash table keyed by symbol name

- Entry contains name, scope, category, type, etc
- Scope stack indicates which scopes are active

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
```

Symbol Table Example




```
type
    T = record
        F1 : integer;
        F2 : real;
    end;
var V : T;
...
module M;
    export I; import V;
    var I : integer;
    ...
    procedure P1 (A1 : real;
        A2t: integer) : real;
    begin
        ...
    end P1;
    ...
    procedure P2 (A3 : real);
    var I : integer;
    begin
        ...
        with V do
            ...
        end;
        ...
    end P2;
    ...
end M;
```

Association Lists

Simple

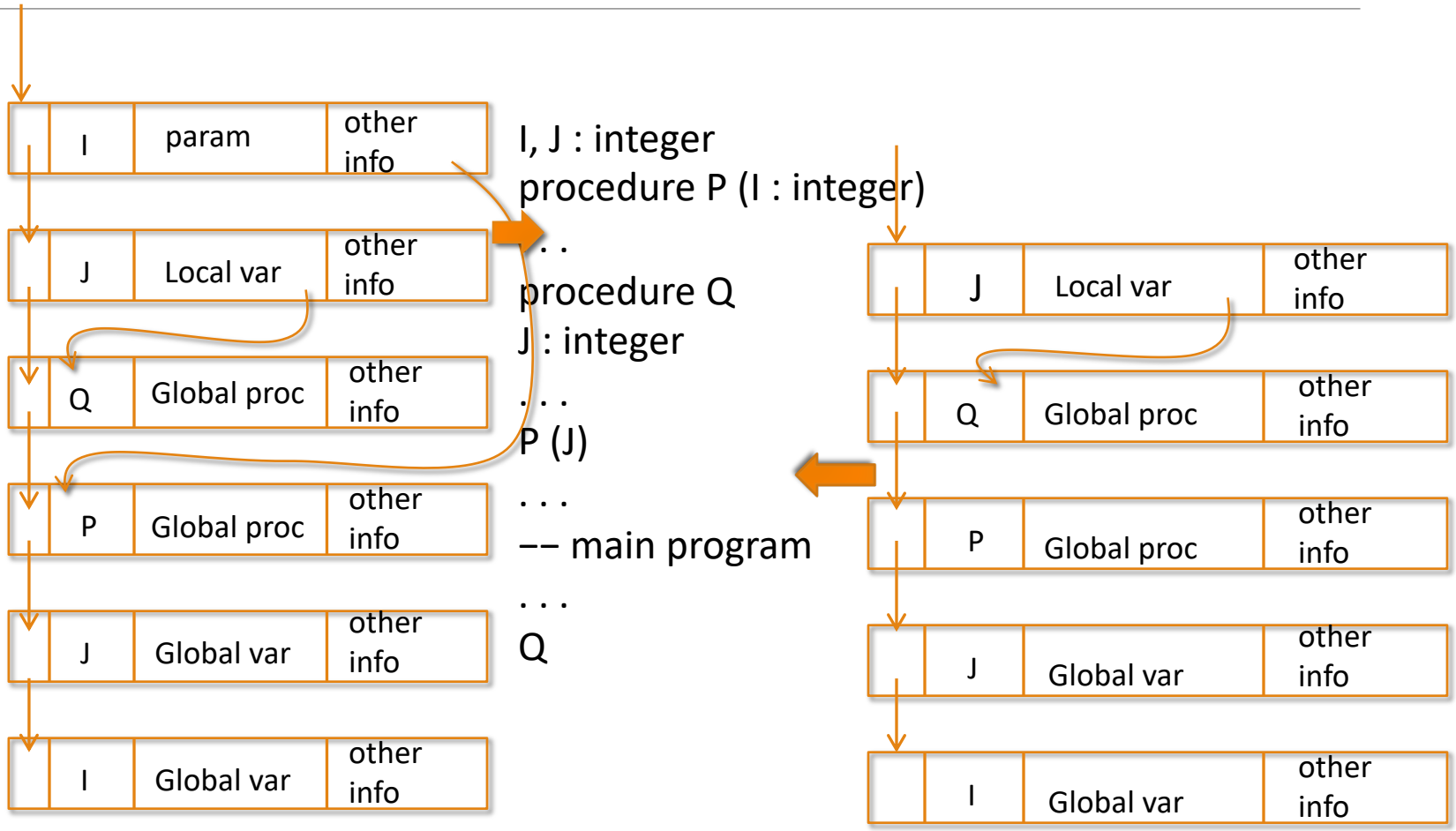
Inefficient (for lookups)

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)

Association Lists - Example



Central Reference Tables

Similar to LeBlanc-Cook but without scope stack

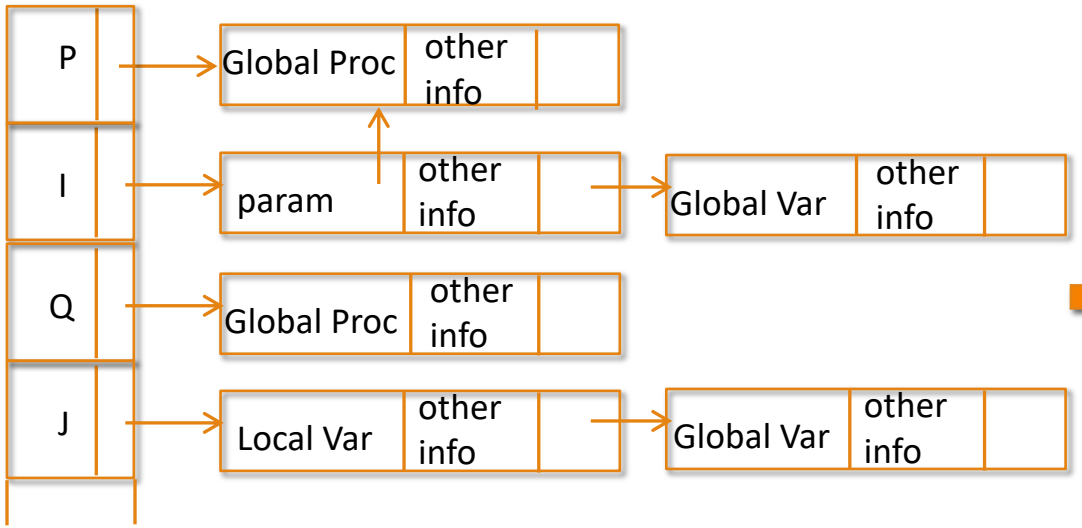
Each symbol name has its own ‘chain’

Inner scopes at the front of each chain

More expensive for scope entry/exit

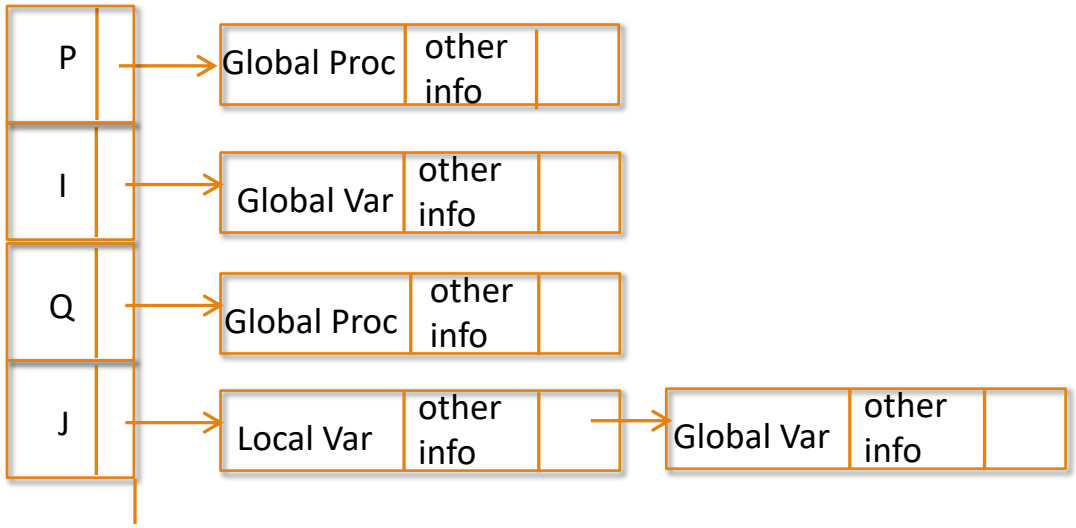
Lookups are faster since chain only holds single symbol name

Central Reference Tables - Example



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

Central Reference Tables - Example




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

Questions