TDDD16 Compilers and interpreters
TDDB44 Compiler Construction

# Data Structures for Symbol Tables

**Complement to Lecture 2 (4), pages 74–78 + 80-88**

Christoph Kessler, IDA,
Linköpings universitet, 2008.

## Symbol table

- **Set of symbol table items**
  - searchable by name + scope
- **Data** stored for each entry:
  - name
  - attributes
    - type (int, bool, array, ptr, function)
    - address (block, offset)
    - declared or not, used or not
    - ...
- **Operations**
  - lookup ( name )
  - insert ( name )
  - put ( name, attribute, value )
  - get ( name, attribute)
  - enterscope ()
  - exitscope()

ADT Dictionary + Scoping Control



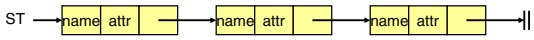
## Data Structures for Symbol Tables

**For flat symbol tables:**
(one block of scope)
- Linear lists
- Hash tables
- ...
(see data structures for ADT Dictionary)

**For nested scopes:**
- Trees of flat symbol tables
- Linear lists with scope control
  - Only for 1-pass-compilers
- Hash tables with scope control (see following slides)
  - Only for 1-pass-compilers



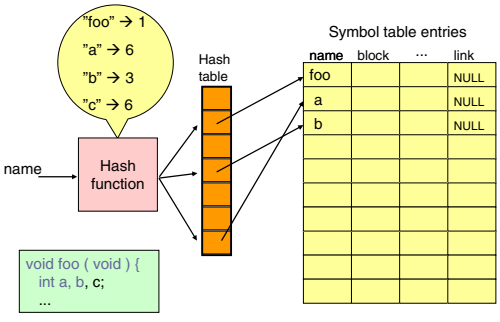
## Linear lists




- Unsorted linear lists
  - ☺ Easy to implement
  - ☺ Space efficient
  - ☺ Insertion itself is fast
    but needs lookup to check if the name was already in
  - ☹ Lookup is slow
    Inserting *n* identifiers and doing *m* lookups requires $O(n(n+m))$ string comparisons

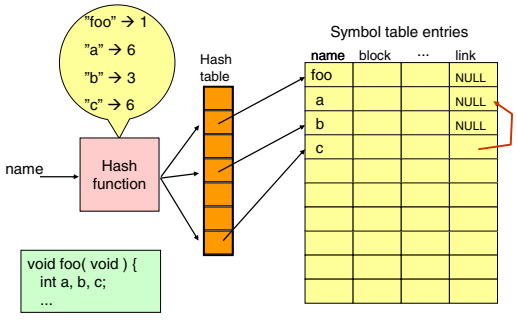


## Hash table with chaining (1)






## Hash table with chaining (2)




☺ Much faster lookup on average

☹ Degenerates towards linear list for bad hash functions

TDDD16 Compilers and interpreters
TDDB44 Compiler Construction

# Hierarchical Symbol Tables

**For nested scope blocks**

Christoph Kessler, IDA,
Linköpings universitet, 2008.

## Tree-based Symbol Table

File/module scope:

```
class Bar {
   int x;
   void foo1( … ) { … }
   void foo2( … ) {
        int inner21( … ) {
           float x;
           …
        }
        int inner22( … ) {
           double x, y;
           …
           foo1( x );
        }
        …
   }
   …
}
…
```

- enterscope(), exitscope()
- insert(), lookup()

Global symbol table

| name | attr | … | link |
|---|---|---|---|
| Bar | | | |

Symbol table for Bar

| name | attr | … | link |
|---|---|---|---|
| x | int | | |
| foo1 | funct | | |
| foo2 | funct | | |

Symbol table for foo1

| name | attr | … | link |
|---|---|---|---|

Symbol table for foo2

| name | attr | … | link |
|---|---|---|---|
| inner21 | funct | | |
| inner22 | funct | | |

Symbol table for inner21

| name | attr | … | link |
|---|---|---|---|
| x | float | | |

Symbol table for inner22

| name | attr | … | link |
|---|---|---|---|
| x | double | | |
| y | double | | |



## For One-Pass Compilers?

File/module scope:

```
class Bar {
   int x;
   void foo1( … ) { … }
   void foo2( … ) {
        int inner21( … ) {
           float x;
           …
        }
        int inner22( … ) {
           double x, y;
           …
           foo1( x );
        }
        …
   }
   …
}
…
```

- enterscope(), exitscope()
- insert(), lookup()

Global symbol table

| name | attr | … | link |
|---|---|---|---|
| Bar | | | |

Symbol table for Bar

| name | attr | … | link |
|---|---|---|---|
| x | int | | |
| foo1 | funct | | |
| foo2 | funct | | |

After code was emitted for foo1 resp. for inner21, could release its symbol table

Symbol table for foo1

| name | attr | … | link |
|---|---|---|---|

Symbol table for foo2

| name | attr | … | link |
|---|---|---|---|
| inner21 | funct | | |
| inner22 | funct | | |

Symbol table for inner21

| name | attr | … | link |
|---|---|---|---|
| x | | | |

Symbol table for inner22

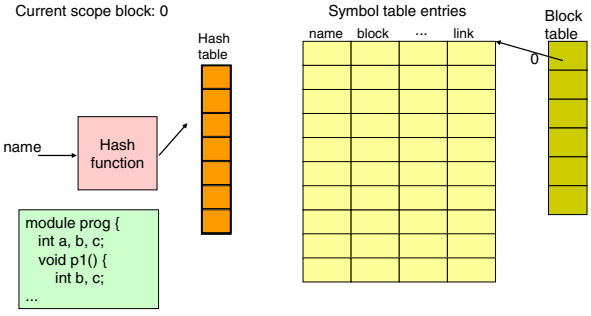
| name | attr | … | link |
|---|---|---|---|
| x | double | | |
| y | double | | |



## Hash tables with chaining + scoping

**(For One-Pass Compilers Only)**

Current scope block: 0

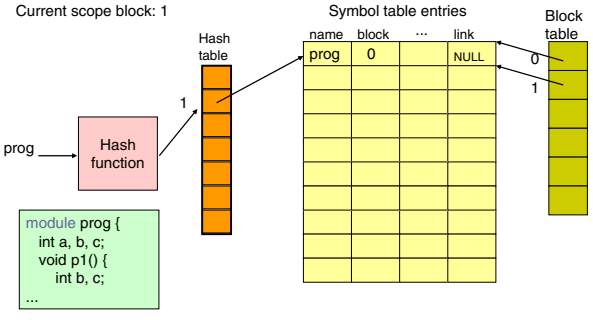



insert p1 and enter a new scope block (2)



## Hash tables with chaining + scoping

Current scope block: 1

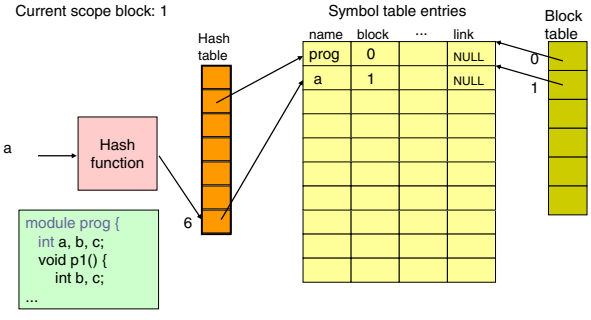



insert prog and enter a new scope block (1)

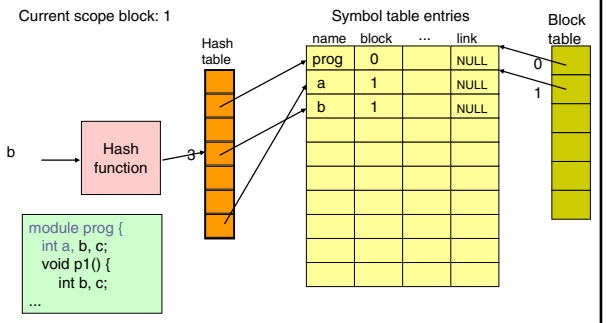


## Hash tables with chaining + scoping

Current scope block: 1

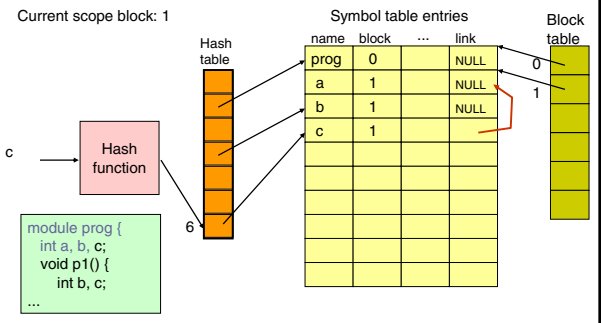


## Hash tables with chaining + scoping





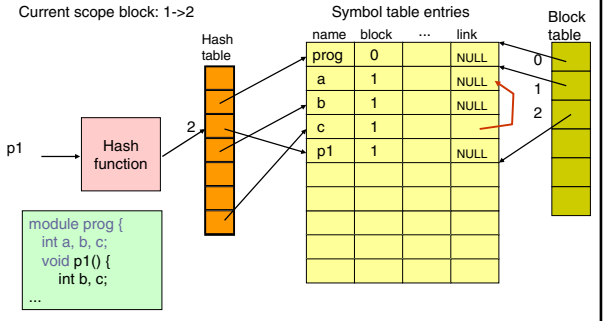

## Hash tables with chaining + scoping




a and c hash to the same hash value (6) – use chaining



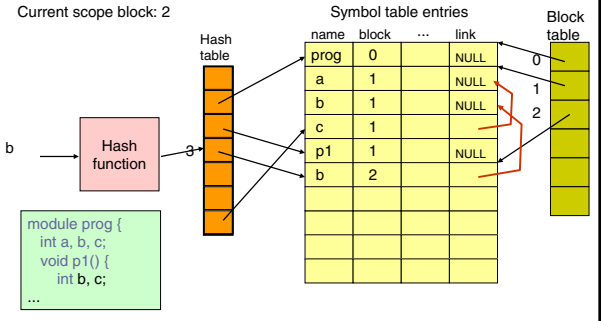
## Hash tables with chaining + scoping




insert p1 and enter a new scope block (2)



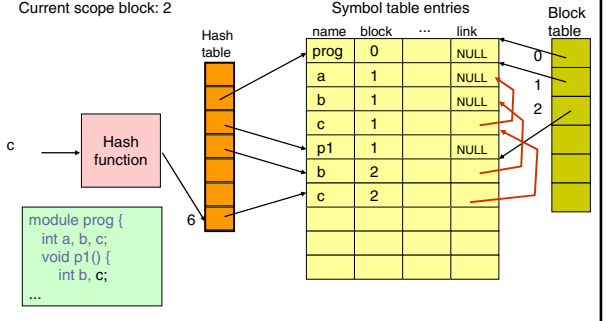
## Hash tables with chaining + scoping




make hash table point to (statically) closest b – will later find this one first in chain

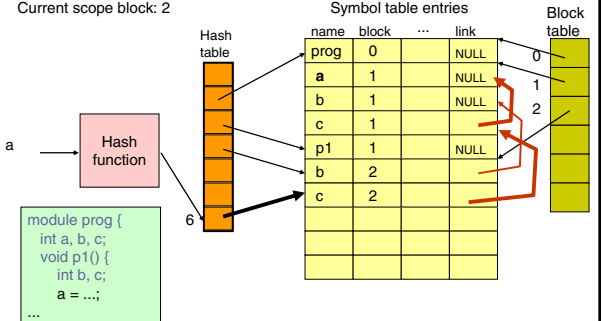


## Hash tables with chaining + scoping






## Hash tables with chaining + scoping




lookup(a): follow chain links ...

TDDD16 Compilers and interpreters
TDDB44 Compiler Construction

# More on Symbol Tables

**Compiler representation of names**

**String space for identifiers**

**Arrays (see old slide set p.78, 79)**

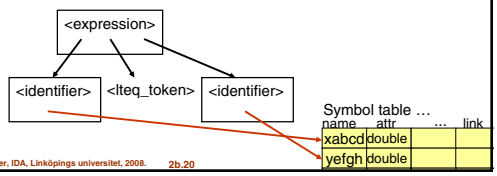
Christoph Kessler, IDA,
Linköpings universitet, 2008.

## Compiler representation of names

- A unique and compact internal representation for a name is the index (address in compiler address space) of its symbol table entry.
- Used instead of full name (string) in the internal representation of a program

☺ Time and space efficient

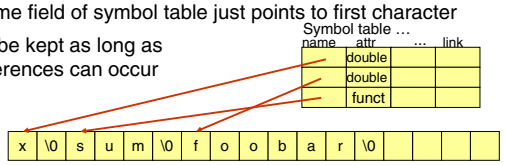
Example: Parse-tree for expression xabcd <= yefgh;






## String Space for Identifiers

- Identifiers can vary in length
- Must be stored in token table
- Name field of symbol table just points to first character
- To be kept as long as references can occur




- Usually, full names kept only during compilation
  - Exception:
    Added to the program's constant pool in the .data segment if symbolic debugging or reflection should be enabled (e.g., gcc –g file1.c to prepare for symbolic debugging)