NTNU – Trondheim
Norwegian University of
Science and Technology

Recitation lecture: problem set 4

Intro to PS 4: Symbol tables

- Organize identifiers and strings so that we can resolve them to memory locations in the finished program
- Variable names and function names are text strings, we need to index a table based on those
- Skeleton includes a **hash table** implementation, **tlhash**

Hash tables in C: tlhash

- The standard library has a hash table implementation, but its reception has been mixed.
- The provided **tlhash.[h|c]** (typeless hash) is a simple implementation

  (You can make your own if you don’t like this implementation, but it is not required; implementing hash tables is a topic for another course)

Hash tables in C: tlhash

- tlhash interface to handle **tlhash_t** struct
  - Initialize
  - Finalize
  - Insert
  - Lookup
  - Remove
  - Obtain all keys
  - Obtain all values
- Keys and values are just void pointers, managing what type they point to is for the calling program to care about

Hash tables in C: tlhash

(Library) function(s) of the week™

- tlhash interface to handle **tlhash_t** struct
  - Initialize
  - Finalize
  - Insert
  - Lookup
  - Remove
  - Obtain all keys
  - Obtain all values
- Keys and values are just void pointers, managing what type they point to is for the calling program to care about
- Example usage in **ir.c**. OBS: Not relevant for the actual solving of the assignment, only intended to show usage

symbol_t struct

- Entry in hash table, and links to the **entry** field in AST nodes.

```
typedef struct s {
    char *name;          // Symbol name
    symtype_t type;      // Type
    node_t *node;        // Pointer to node in AST
    size_t seq;          // Sequence number
    size_t nparms;       // Number of parameters (function)
    tlhash_t *locals;    // Local symbol table (function)
} symbol_t;
```

symbol_t struct

- Name : Name of symbol
- Type : Function, global var, parameter or local var
- Node : Root node of function
- Seq : Sequencing number (not for global vars)
- Param : Parameter count for functions
- Locals : Local symbol table for functions

TODO: Globals and functions

- Main function calls **create_symbol_table**, your origin for the following tasks
- Skeleton declares a global symbol table: **global_names**
- Fill with symbol_t structs for functions and gobal vars (implement **find_globals**)
- Functions need their own name table, it can already be filled with parameter names
- Functions also link to their tree node (so we can traverse a function’s subtree, given its name)
- Number functions and parameters (seq)

TODO: Locals

- Traverse each function's subtree, resolve names and string within its scope (implement **bind_names**)
- Both entering declared names into its local table, and linking used names to the symbol they represent
- Look up used identifiers first locally, then globally
- Create global index of string literals
- Sequence numbers should be assigned **by the order of appearance**, parameters coming first

TODO: Print and destroy symtab

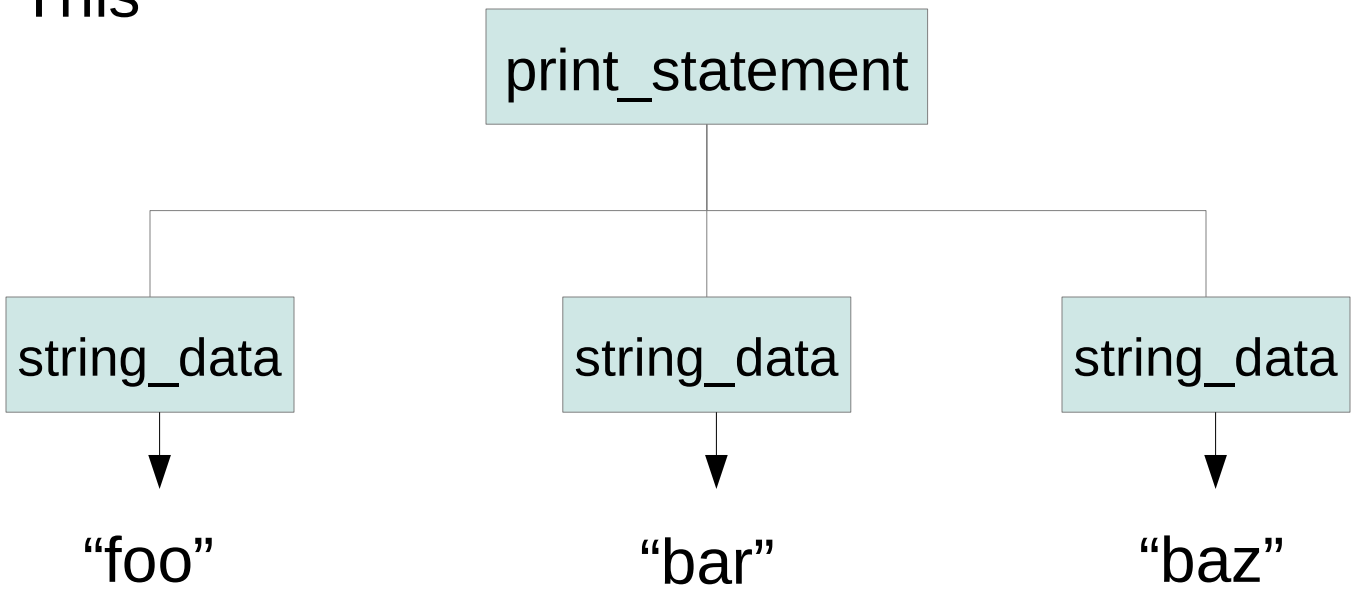
- Implement **print_symbol_table** to display your table
- Lastly, free up all allocated memory (implement **destroy_symbol_table**)

Global index of string literals

- String literals are static data and are used only once, in the node representing them
- The node currently contains a pointer to the string at the data element
- In code generation we want to write out all strings at once, so
  - Add the pointer ti the global **string_list**
  - Keep a count of strings: **stringc**
  - Remember to grow the table as needed
  - Replace the node's data element with the list index of the string it used to hold

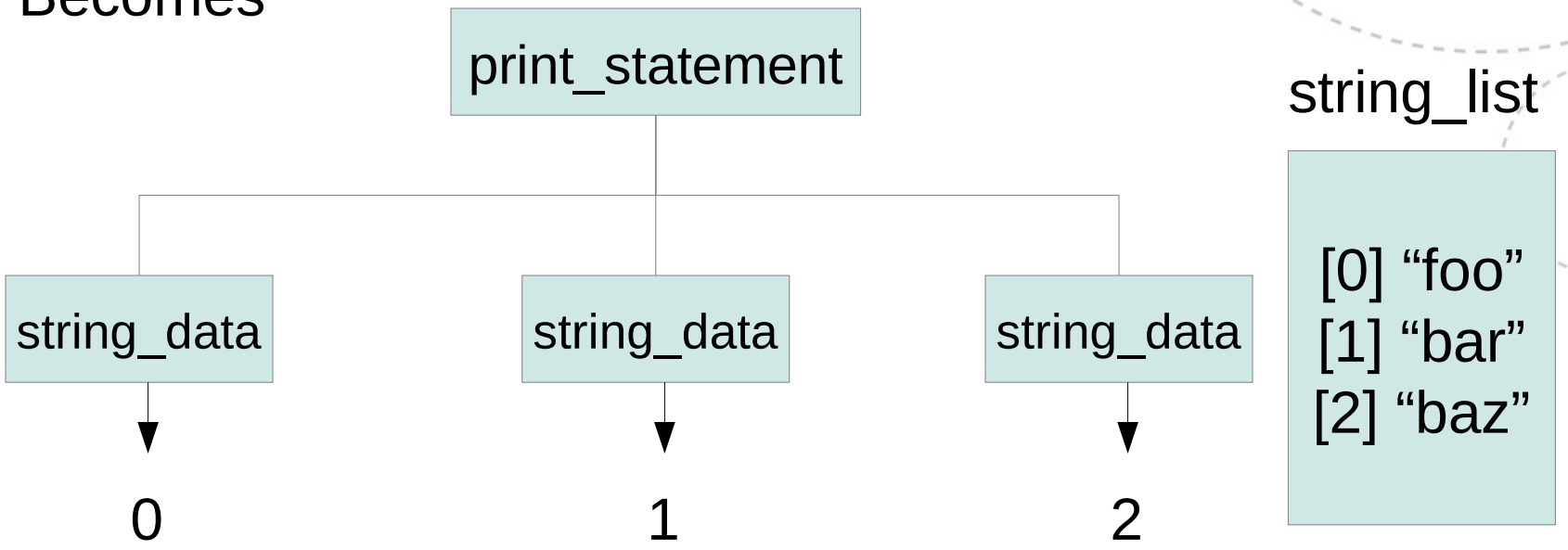
Global index of string literals

- This



Global index of string literals

- Becomes



Local name tables

- In the end, we want them in a single table:
  local #0: x
  local #1: y
  local #2: z
  local #3: x
  local #4: y

Local name tables for blocks

- Only temporarily
  - While traversing the inner block, looking up “x” should result in the symtab entry for local #3
  - When exiting the block, we go back to expecting local #0 for “x”
- We can use a stack for temporary hash tables
  - Push a new one when entering a block
  - Insert locally declared names, make them point onwards to the real symtab entry
  - Look up names in a bottom-up fashion
  - Pop temp table when exiting block
- Naming does not matter after linking is done, but number local variables so that we can tell inner and outer symbols apart

Minor tips

- -h flag to show options. Newly added are
  - -u : If you prefer the old tree_print over the student contribution demonstrated on Piazza
  - -s : Invokes print_symbol_table
- You are welcome to declare more internal helper functions to keep your code from become one giant and messy singleton function

Scores and such

- Full marks for successful run and correct sequencing
- Partial marks for an attempt, depending on how much has been done and how much is working
- Zero (but approved) for handing in blank
  - Encourage everyone to **try** everything, and worst case comment some notes about what you would have done
- How much do they *really* count? 1/100 on PS4 is 1/1000 in the course.
  - You can't numerically exclude yourself from getting an A by losing points on this assignment alone
- Start early, lots of work to be done