Lecture Notes 6

Names

- Name (or identifier) – Denotes language construct or entity
- Case Sensitivity/Insensitivity – Does the case of the letters in the name matter or not
- Name Length – Some languages limit length of name, others have no limit
- Reserved Word – A word that cannot be used as a name in programming language
  - Keyword vs Reserved Word – Book distinguishes the two stating that a keyword can be redefined, reserved word cannot

Variables

- Location – Where is structure located (more abstract than address)
- Address – Specific memory location of a target architecture
- Value – Storable quantity
- Type – Specifies the range of values in which a variable can be assigned
- Lifetime – Time in which the variable is bound to a memory location
- Scope – Range of statements in which the variable is visible
- L-Value – Left hand side of an assignment, it is a location (or address) of a variable
- Aliases – Variable names that can all be used to access same memory location
- R-Value – Right hand side of an assignment, value that can be stored in a variable

Binding

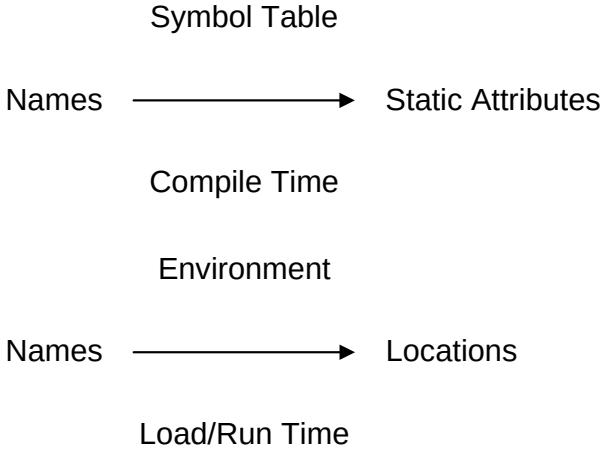
- Attributes – Properties associated with a Name or Identifier

```
const int n = 5;
int x;

double f(int n){
...
}
```

- Binding – Associating attribute to an entity (or name)
- Binding time – When is the attribute bound to the name
  - Static vs. Dynamic binding – Before or during execution
  - Language definition time – When the language is being specified
  - Language implementation time – When the compiler (or interpreter) is being written
  - Translation time or Compile time – When the code is compiled (or interpreted)

    - Link time – When the modules (or object files) are being linked together
    - Load time – When the program is being loaded
    - Execution or Run time – When the program is actually running
- Static Binding – Binding occurs before running and does not change during execution
- Dynamic Binding – Binding occurs during execution and can change during program execution
- Explicit Declaration – A program statement that explicitly states variable names and types
- Implicit Declaration – A means of associating variables with types without explicit declaration statements
- Type Inference – Implicit type determination using context
- Allocation – Process of binding a variable to a memory cell
- Deallocation – Process of unbinding a variable and returning memory cell to available memory
- Static Variable – Variable bound to a memory cell prior to program execution
- Stack-Dynamic (or Automatic Allocated) Variables – Variables whose storage bindings are created when declaration statements are executed
- Explicit Heap-Dynamic (or Dynamically Allocated) Variables – Memory cells are explicitly allocated during runtime
- Implicit Heap-Dynamic Variables – Variables bound to heap storage when assigned values
- Symbol Table – Structure that translates names to attributes in compiler or interpreter
    - Names → Attributes
    - Compiler



Memory

Locations ⟶ Values

Run Time

- Interpreter

Environment

Names ⟶ Attributes (Locations & Values)

Interpretation

Scope

- Visible – A variable can be referenced or assigned in a statement
- Local – A variable that is declared in a program unit or block
- Static Scoping – Binding names to nonlocal variables
- Static Parent – The subprogram (or block) in which the subprogram (or block) is declared
- Static Ancestor – The subprogram (or block) that encloses the subprogram (or block) in question
- Block – A section of code
- Block-Structured Language – A language that is structured using blocks (e.g. C style)
- Dynamic Scoping – Scope is based upon the calling sequence, not the subprogram spacing

Scope Analysis

- Scope Analysis – Process of adding/removing bindings as declarations are made/blocks are exited
- Static Scoping (Lexical Scoping) – Bindings are static, handled by compiler

```
1   int x;
2   char y;
3
4   void p(void){
5       double x;
6       ...
7       {
8           int y[10];
9           ...
10      }
11      ...
12  }
```

```
13
14   void q(void){
15       int y;
16       ...
17   }
18
19   int main(){
20       char x;
21       ...
22   }
```

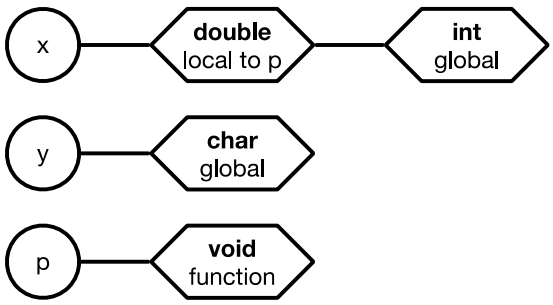


Figure 1. Symbol Table at Line 5

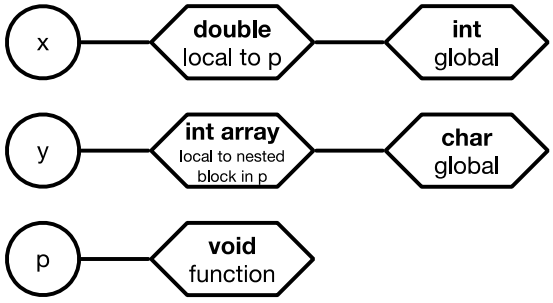


Figure 2. Symbol Table at Line 8

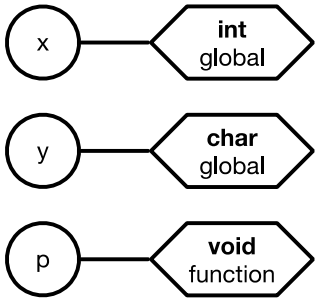


Figure 3. Symbol Table at Line 13

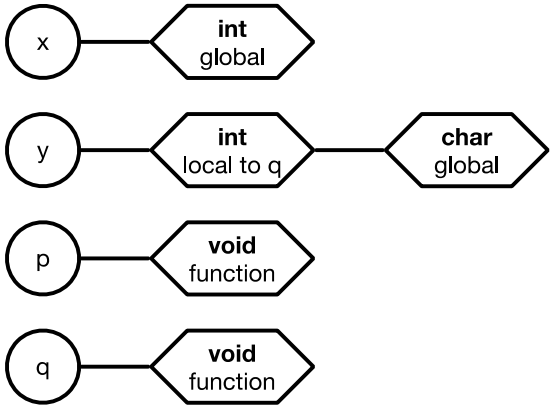


Figure 4. Symbol Table at Line 15

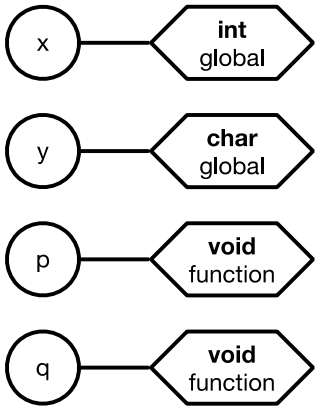


Figure 5. Symbol Table at Line 18

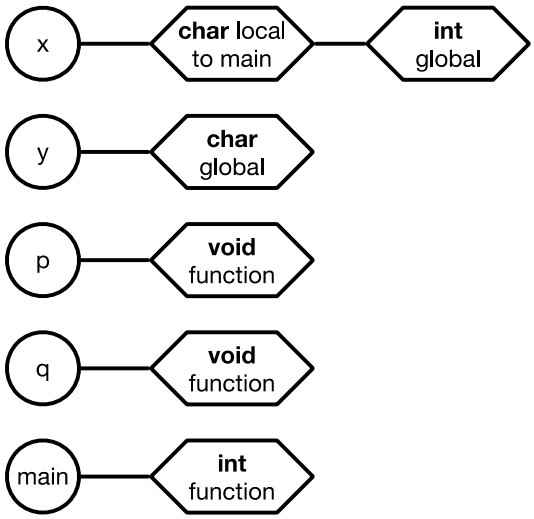


Figure 6. Symbol Table at Line 20

- Dynamic Scoping – Bindings are dynamic, handled at runtime

```
 1  #include <stdio.h>
 2
 3  int x = 1;
 4  char y = 'a';
 5
 6  void p(void){
 7      double x = 2.5;
 8      printf("%c\n", y);
 9      {
10          int y[10];
11      }
12  }
13
14  void q(void){
15      int y = 42;
16      printf("%d\n", x);
17      p();
18  }
19
20  int main(){
21      char x = 'b';
22      q();
23      return 0;
24  }
```

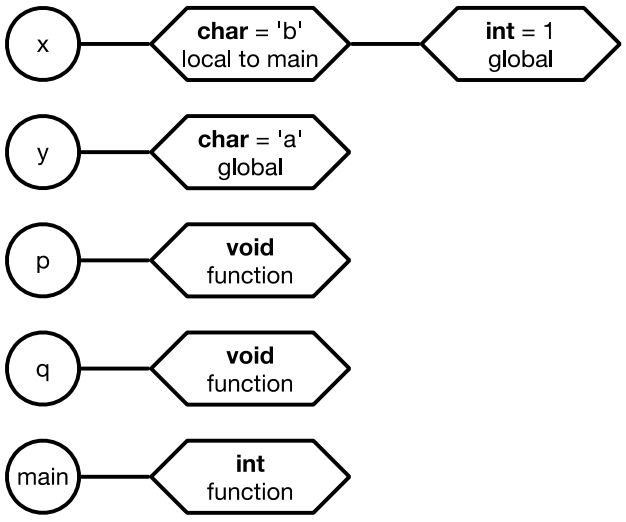


Figure 7. Symbol Table at Line 22

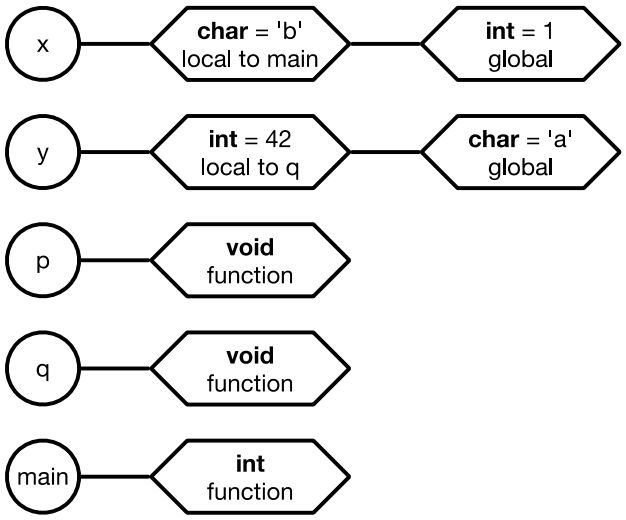


Figure 8. Symbol Table at Line 16

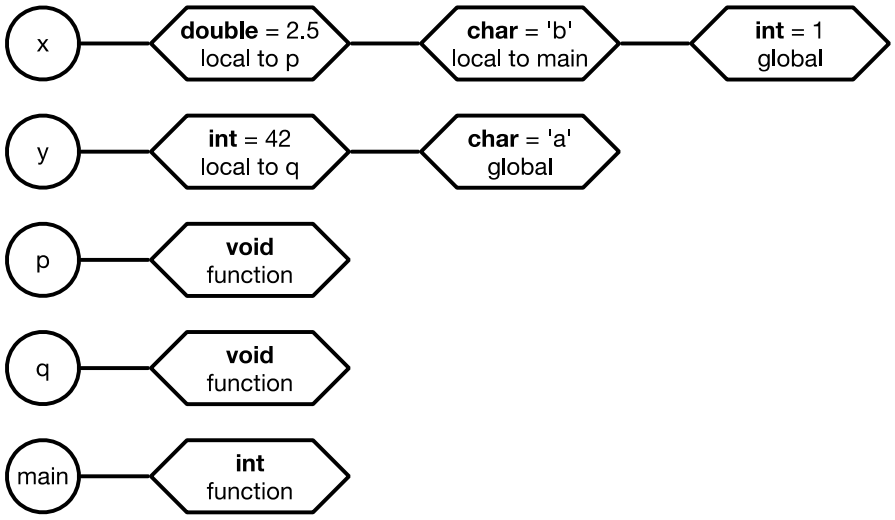


Figure 9. Symbol Table at Line 8

Miscellaneous

- Referencing Environment – A collection of all variables that are visible in the statement
- Active Subprogram – A subprogram that has begun but not completed
- Named Constant – A variable that is bound to a value only once