

Symbol Tables

what is a symbol table?

- One of the most important things a compiler keeps track of is the **symbols** in the program
 - What names are used for **variables, functions, structs, classes**, etc.
- One symbol table per **scope**
 - A scope is a region of a program where certain symbols are accessible (e.g., global, local to a function)
 - Scopes can be nested
 - Within a function, can access both local variables and global variables

```
int x, y, z;  
float z[20];  
int ** p;  
struct S {  
    int x;  
    float y;  
};  
struct S q;
```

what do we keep track of?

- What a symbol table keeps track of for each symbol depends on the symbol and the scope
 - **Variables:** name, type, size of variable (needed for allocating space)
 - If variable is global, may keep track of address, if local to a function, keep track of where in the **activation record** it is (where in a function's stack frame)
 - **Arrays:** name, type, size, number of elements
 - **Functions:** name, return type, number and type of arguments
 - **Structs:** name, types and sizes of variables
- Note that symbols may refer to each other: e.g., variable type/size might be determined by a struct definition

```
int x, y, z;  
float z[20];  
int ** p;  
struct S {  
    int x;  
    float y;  
};  
struct S q;
```

next: abstract syntax trees