

SYMBOL TABLE DESIGN



THE STRUCTURE OF A COMPILER

- Up to this point we have treated a compiler as a single box that maps a source program into a semantically equivalent target program.



WHAT'S INSIDE THIS BOX?

- If we open up this box a little, we see that there are two parts to this mapping:

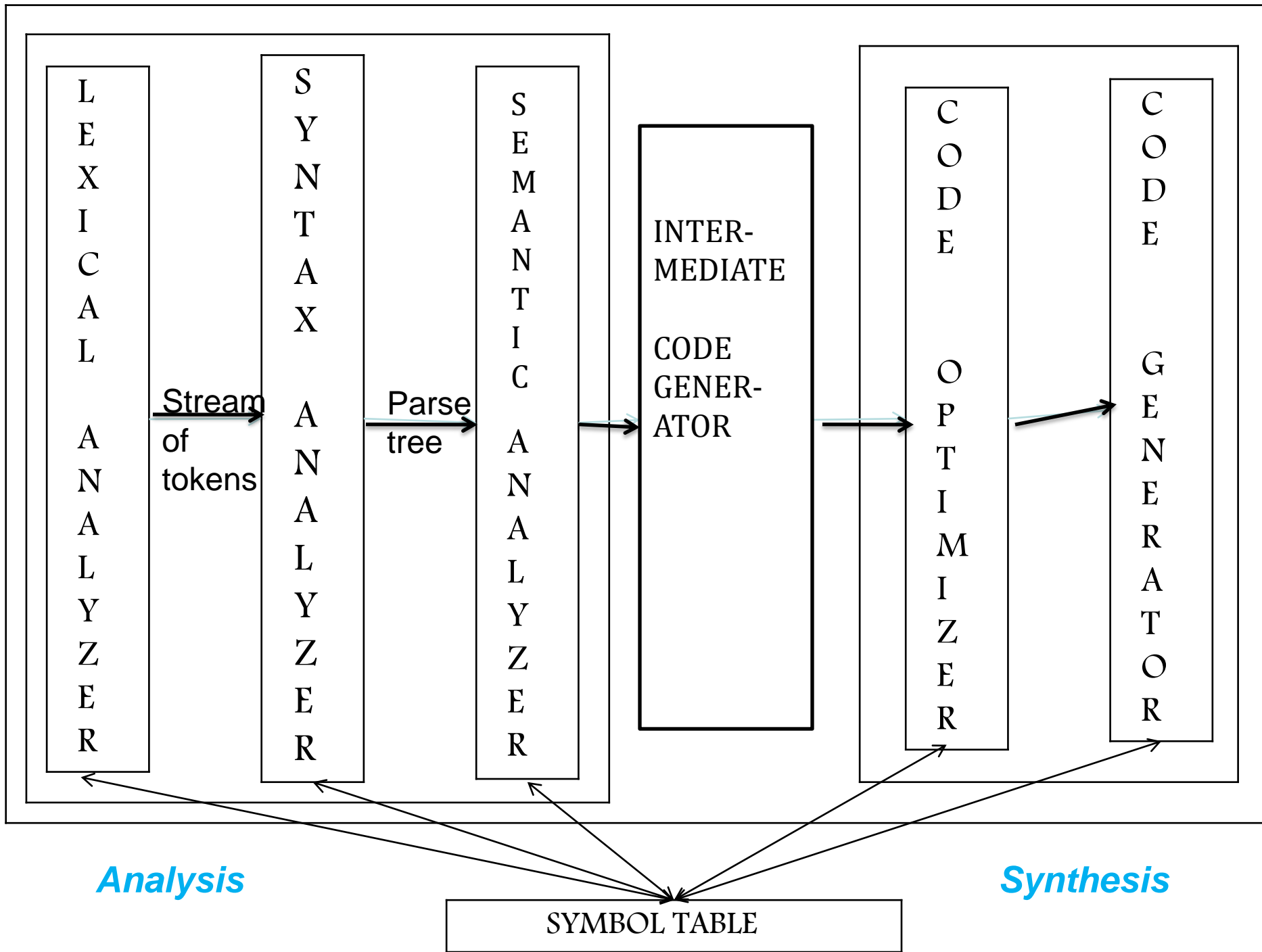


ANALYSIS



SYNTHESIS





ANALYSIS

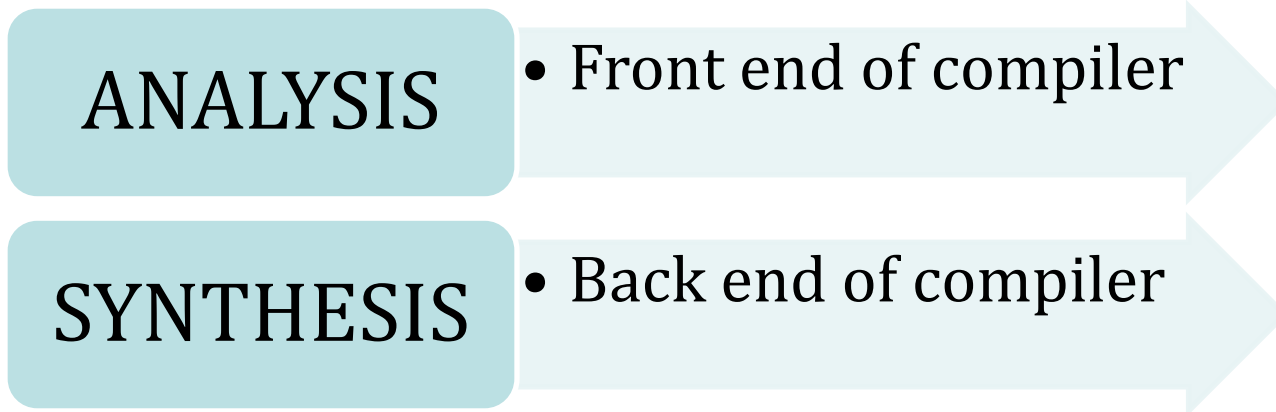
Breaks up the source program into constituent pieces and imposes a grammatical structure on them. It then uses this structure to create an intermediate representation of the source program.

If the analysis part detects that the source program is either syntactically ill formed or semantically unsound, then it must provide informative messages, so the user can take corrective action.

*The analysis part also collects information about the source program and stores it in a data structure called a **symbol table**, which is passed along with the intermediate representation to the synthesis part.*

SYNTHESIS

- The synthesis part constructs the desired target program from the intermediate representation and the information in the *symbol table*



COMPILERS ROLE??

- An essential function of a compiler –

Record the variable names used in the source program and collect information about various attributes of each name.

- These attributes may provide information about the storage allocated for a name , its type and its scope , procedure names ,number and types of its arguments, the method of passing each argument and the type returned


SO , WHAT EXACTLY IS SYMBOL TABLE??

*Symbol tables are **data structures** that are used by compilers to hold information about source-program constructs.*

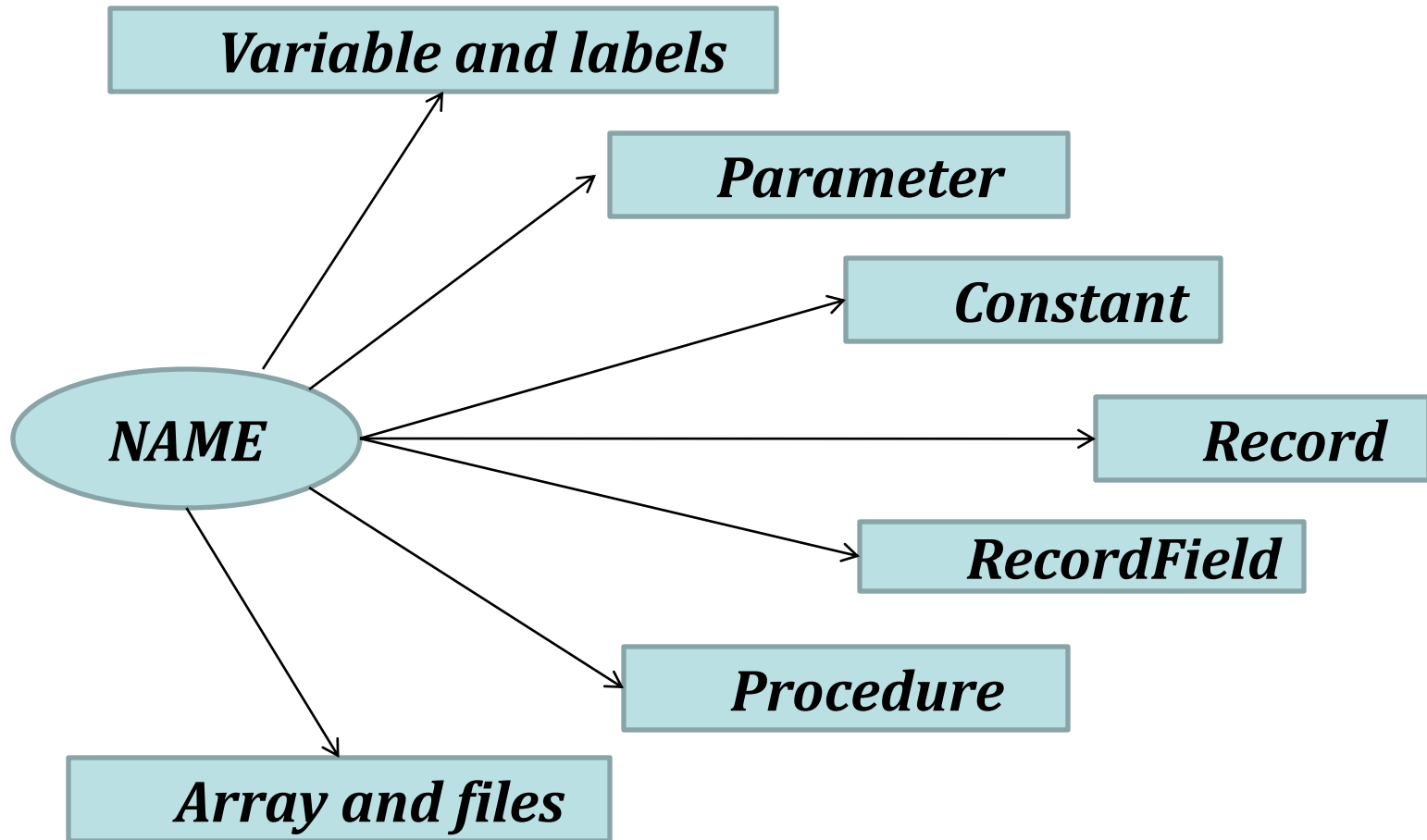
A symbol table is a necessary component because

- Declaration of identifiers appears once in a program
- Use of identifiers may appear in many places of the program text

INFORMATION PROVIDED BY SYMBOL TABLE

- *Given an Identifier which name is it?*
 - *What information is to be associated with a name?*
 - *How do we access this information?*
- 

SYMBOL TABLE - NAMES




SYMBOL TABLE-ATTRIBUTES


- Each piece of information associated with a name is called an *attribute*.
- Attributes are language dependent.
- Different classes of Symbols have different Attributes

Variable, Constants	Procedure or function	Array
<ul style="list-style-type: none">• Type , Line number where declared , Lines where referenced , Scope	<ul style="list-style-type: none">• Number of parameters, parameters themselves, result type.	<ul style="list-style-type: none">• # of Dimensions, Array bounds.


WHO CREATES SYMBOL TABLE??

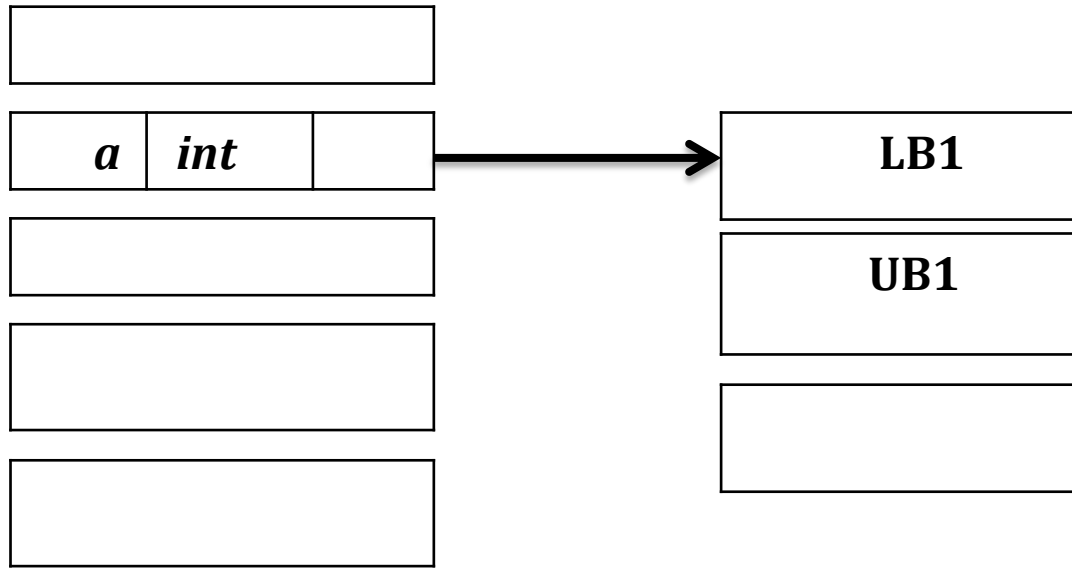
- Identifiers and attributes are entered by the analysis phases when processing a definition (declaration) of an identifier
 - In simple languages with only global variables and implicit declarations:
 - ✓ The scanner can enter an identifier into a symbol table if it is not already there
 - In block-structured languages with scopes and explicit declarations:
 - ✓ The parser and/or semantic analyzer enter identifiers and corresponding attributes
- 

USE OF SYMBOL TABLE

- Symbol table information is used by the analysis and synthesis phases
 - To verify that used identifiers have been defined (declared)
 - To verify that expressions and assignments are semantically correct – type checking
 - To generate intermediate or target code
- 
- A decorative graphic at the bottom of the slide consisting of two wavy, overlapping lines. The top line is a dark blue color, and the bottom line is a light gray color. Both lines curve across the width of the slide, creating a modern, abstract footer design.

IMPLEMENTATION OF SYMBOL TABLE


- Each entry in the symbol table can be implemented as a ***record*** consisting of several field.
 - These fields are ***dependent*** on the information to be saved about the name
 - But since the information about a name depends on the usage of the name the ***entries in the symbol table records will not be uniform.***
 - Hence to keep the symbol tables records uniform some information are kept outside the symbol table and a pointer to this information is stored in the symbol table record.
- 

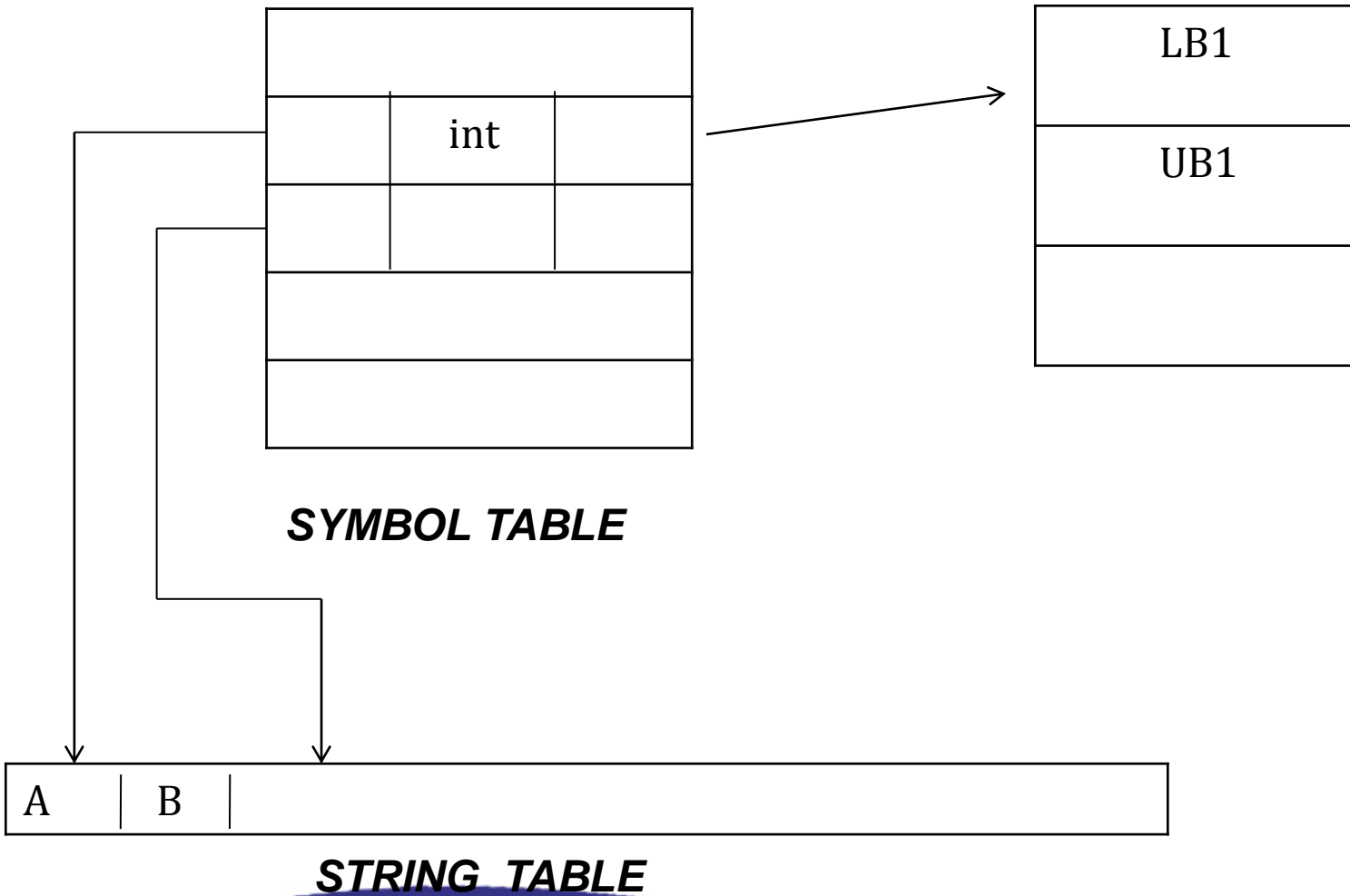


SYMBOL TABLE

A pointer steers the symbol table to remotely stored information for array a.

WHERE SHOULD NAMES BE HELD??

- If there is modest upper bound on the length of the name , then the name can be stored in the symbol table record itself.
 - But If there is no such limit or the limit is already reached then an indirect scheme of storing name is used.
 - A separate array of characters called a '***string table***' is used to store the name and a pointer to the name is kept in the symbol table record
- 




SYMBOL TABLE AND SCOPE

- Symbol tables typically need to support multiple declarations of the same identifier within a program.

The scope of a declaration is the portion of a program to which the declaration applies.

- We shall implement scopes by setting up *a separate symbol table for each scope.*
- 

- The rules governing the scope of names in a block-structured language are as follows
 1. A name declared within a block B is valid only within B.
 2. If block B1 is nested within B2, then any name that any name that is valid for B2 is also valid for B1, unless the identifier for that name is re-declared in B1.
 - The scope rules required a more complicated symbol table organization than simply a list association between names and attributes.
 - Each table is list names and there associated attributes and the tables are organized into a stack.
- 

SYMBOL TABLE ORGANIZATION

