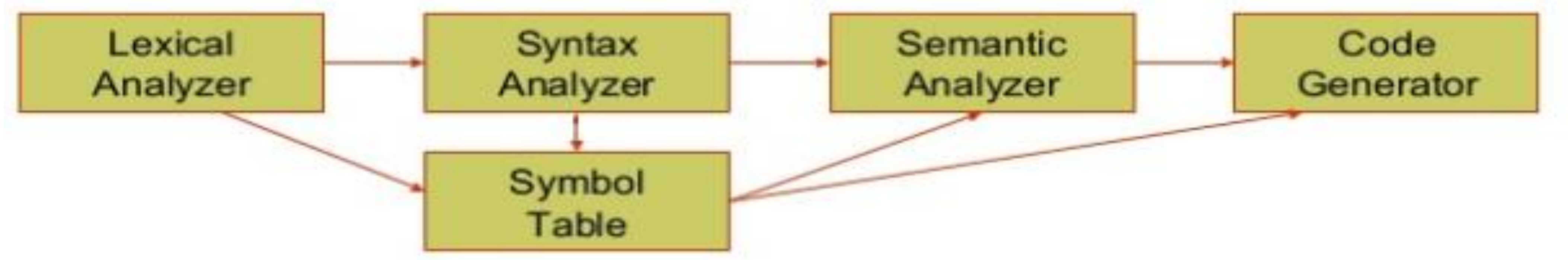
Symbol Table

What is Symbol Table

- Symbol table is a data structure that facilitate effective and efficient way of managing information about various names appearing in the source program.
- The information is collected by the analysis phases of compiler and is used by synthesis phases of compiler to generate code.
- It is used by compiler to achieve compile time efficiency.



Symbol table interaction with other phases

It is used by various phases of compiler as follows :-

- **Lexical Analysis:** Creates new table entries in the table, example like entries about token.
- **Syntax Analysis:** Adds information regarding attribute type, scope, dimension, line of reference, use, etc. in the table.
- **Semantic Analysis:** Uses available information in the table to check for semantics i.e. to verify that expressions and assignments are semantically correct(type checking) and update it accordingly.
- **Intermediate Code generation:** Refers symbol table for knowing how much and what type of run-time is allocated and table helps in adding temporary variable information.
- **Code Optimization:** Uses information present in symbol table for machine dependent optimization.
- **Target Code generation:** Generates code by using address information of identifier present in the table.

Symbol Table Entries

Each entry in symbol table is associated with attributes that support compiler in different phases. Items stored in Symbol table are:

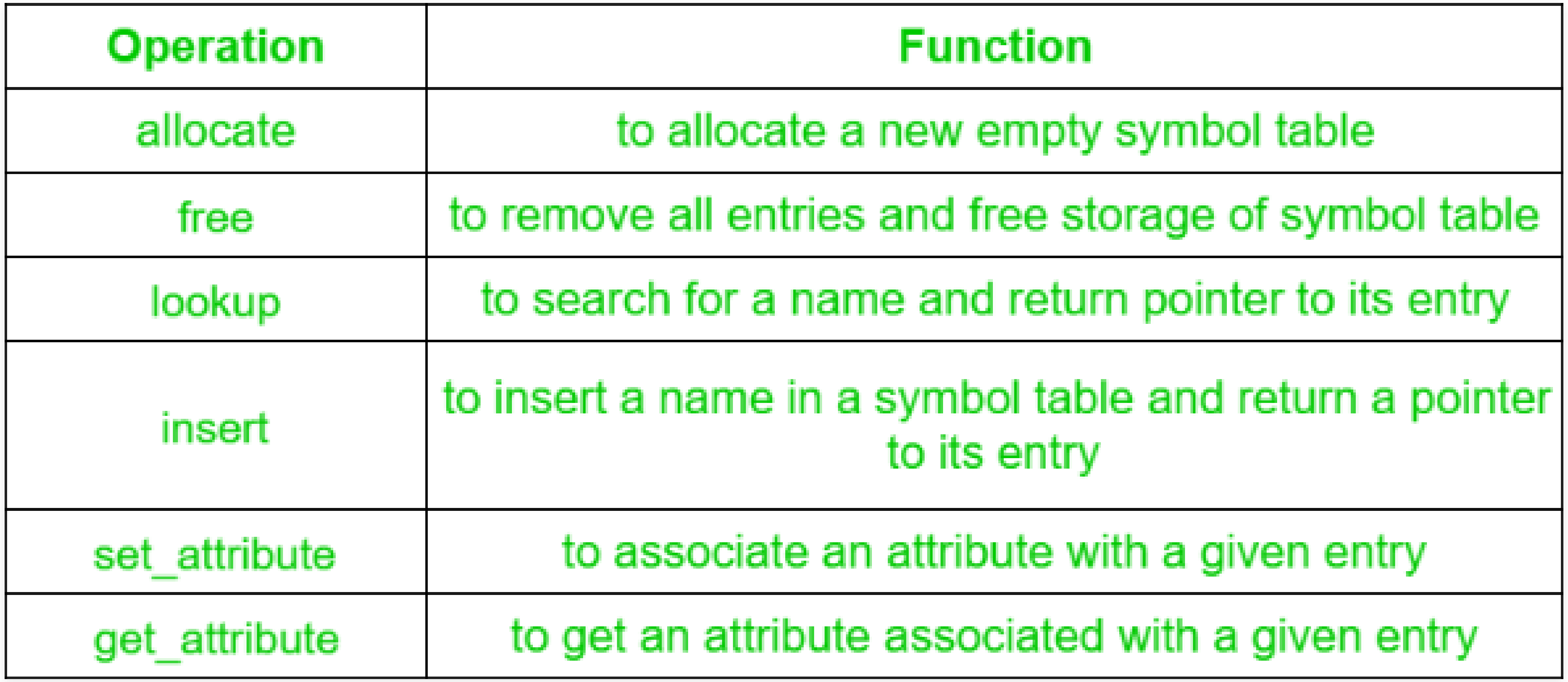
- Variable names and values
- Procedure and function names
- Literal constants and strings
- Compiler generated temporaries
- Labels in source languages

Functionalities provided by Symbol Table

A symbol table may serve the following purposes:

- To store the names of all entities in a structured form at one place.
- To verify if a variable has been declared.
- To implement type checking, by verifying assignments and expressions in the source code are semantically correct.
- To determine the scope of a name (scope resolution).

Operations on Symbol Table



Operation	Function
allocate	to allocate a new empty symbol table
free	to remove all entries and free storage of symbol table
lookup	to search for a name and return pointer to its entry
insert	to insert a name in a symbol table and return a pointer to its entry
set_attribute	to associate an attribute with a given entry
get_attribute	to get an attribute associated with a given entry

Implementation of Symbol Table

Following are commonly used data structure for implementing symbol table :-

- **List –**
- **Linked List –**
- **Hash Table –**
- **Binary Search Tree –**

Implementation of Symbol Table

List –

- In this method, an array is used to store names and associated information.
- A pointer **“available”** is maintained at end of all stored records and new names are added in the order as they arrive
- To search for a name we start from beginning of list till available pointer and if not found we get an error **“use of undeclared name”**
- While inserting a new name we must ensure that it is not already present otherwise error occurs i.e. **“Multiple defined name”**
- Insertion is fast O(1), but lookup is slow for large tables – O(n) on average
- Advantage is that it takes minimum amount of space.

Implementation of Symbol Table

Linked List –

- This implementation is using linked list. A link field is added to each record.
- Searching of names is done in order pointed by link of link field.
- A pointer **“First”** is maintained to point to first record of symbol table.
- Insertion is fast O(1), but lookup is slow for large tables – O(n) on average

Implementation of Symbol Table

Hash Table –

- In hashing scheme two tables are maintained – a hash table and symbol table and is the most commonly used method to implement symbol tables..
- A hash table is an array with index range: 0 to table_size – 1.These entries are pointer pointing to names of symbol table.
- To search for a name we use hash function that will result in any integer between 0 to table_size – 1.
- Insertion and lookup can be made very fast – O(1).
- Advantage is quick search is possible and disadvantage is that hashing is complicated to implement.

Implementation of Symbol Table

Binary Search Tree –

- Another approach to implement symbol table is to use binary search tree i.e. we add two link fields i.e. left and right child.
- All names are created as child of root node that always follow the property of binary search tree.
- Insertion and lookup are O($\log_2$ n) on average.

Scope Management in Symbol Table

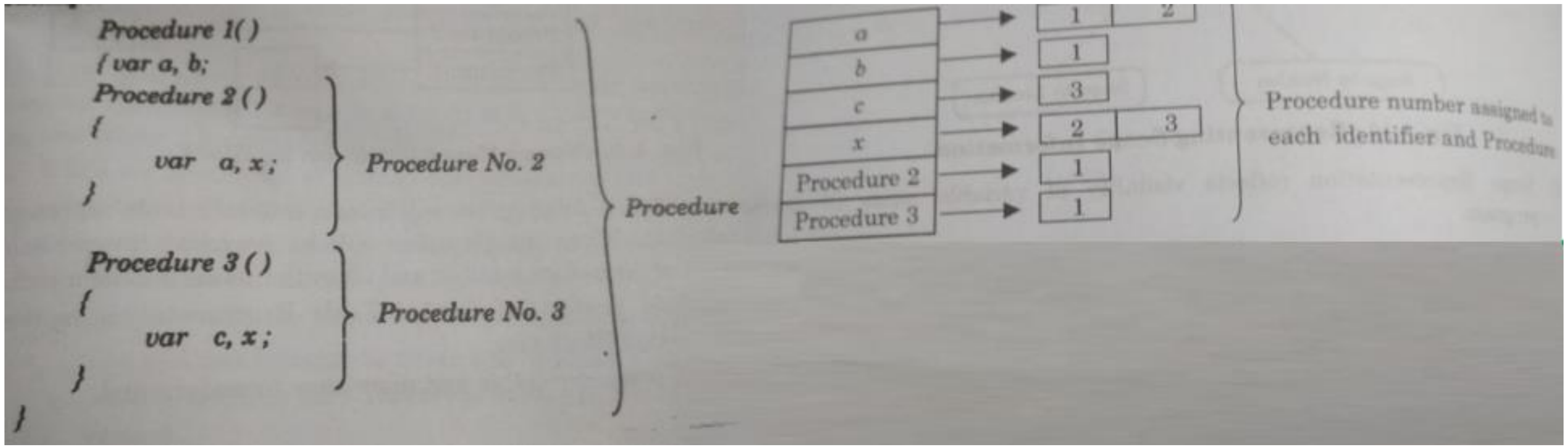
Scope –

- The scope of an identifier represents the portion of a program in which that identifier is accessible.
- It matches identifiers’ declaration with its uses.
- It specifies the lifetime of variable in a particular block.
- The same identifier may refer to different things in different parts of the program: Different scopes for same name don’t overlap.

Scope Management in Symbol Table

Way to Represent Scope –

- Scope by Number: Store all values in a single symbol table. All blocks are given a unique number. While storing identifier entry in symbol table, we also store its block number. In this manner identifiers with same name but from different blocks will not overwrite in symbol table.



```
Procedure 1()
{ var a, b;
Procedure 2 ()
{
      var  a, x ;
}

Procedure 3 ()
{
      var  c, x ;
}
}
```

Scope Management in Symbol Table

Way to Represent Scope –

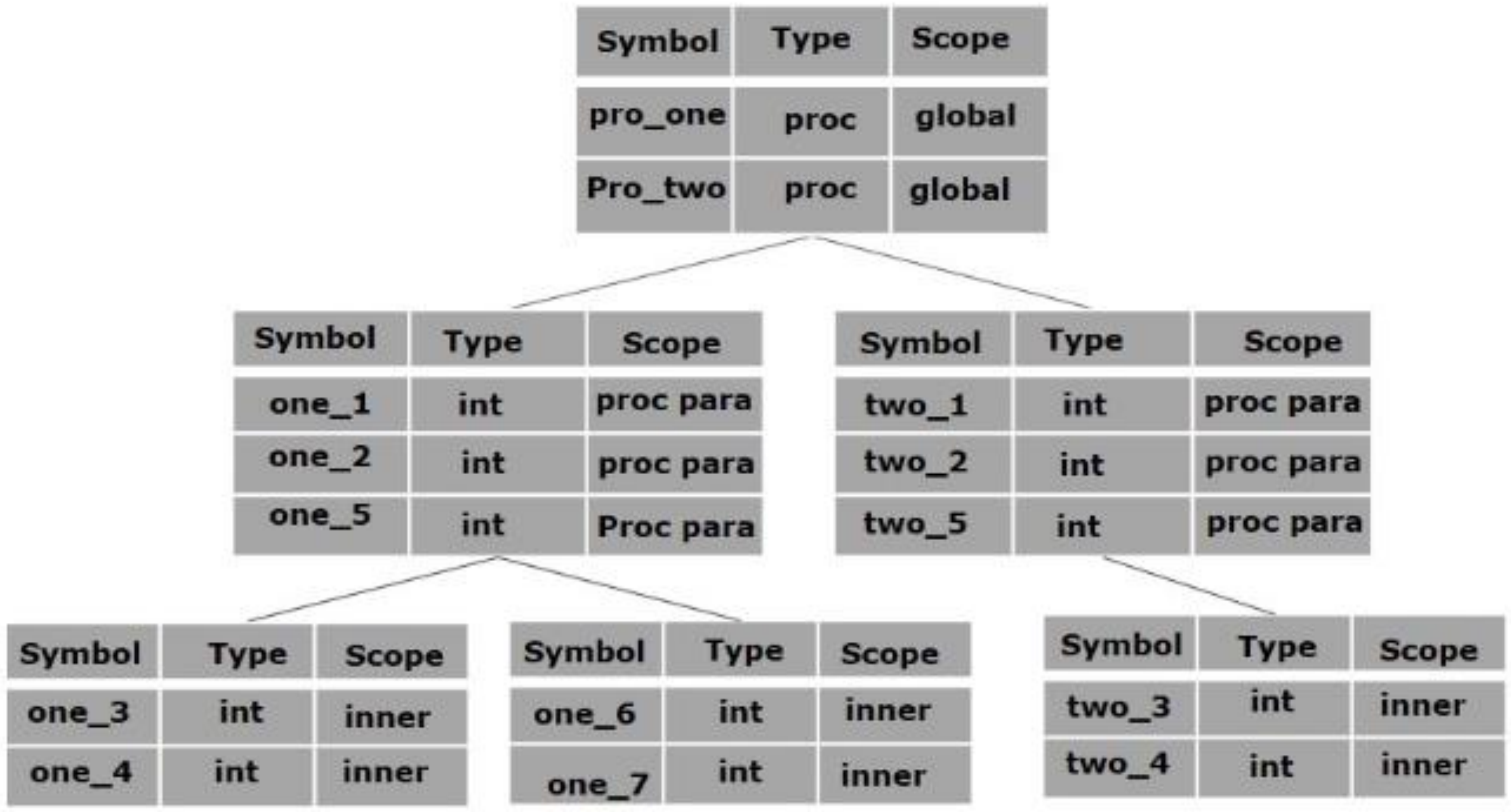
- Scope by Location: Separate symbol table is created for each procedure or block.
- A compiler maintains two types of symbol tables:
  - **Global symbol table** which can be accessed by all the procedures
  - **Scope symbol tables** that are created for each scope in the program.
- To determine the scope of a name, symbol tables are arranged in hierarchical structure.

Representing Scope Information in Symbol Table

```
int value=10;
void pro_one()
{
   int one_1;
   int one_2;
                 {                      \
                      int one_3;     |_  inner scope 1
                      int one_4;     |
                  }                      /
   int one_5;
                 {                      \
                      int one_6;     |_  inner scope 2
                      int one_7;     |
                  }                      /
  }
```

```
void pro_two()
{
   int two_1;
   int two_2;
         {                       \
                int two_3;     |_  inner scope 3
                int two_4;     |
         }                       /
   int two_5;
   }
. . .
```

Representing Scope Information in Symbol Table



Symbol	Type	Scope
pro_one	proc	global
Pro_two	proc	global

Symbol	Type	Scope
one_1	int	proc para
one_2	int	proc para
one_5	int	Proc para

Symbol	Type	Scope
two_1	int	proc para
two_2	int	proc para
two_5	int	proc para

Symbol	Type	Scope
one_3	int	inner
one_4	int	inner

Symbol	Type	Scope
one_6	int	inner
one_7	int	inner

Symbol	Type	Scope
two_3	int	inner
two_4	int	inner