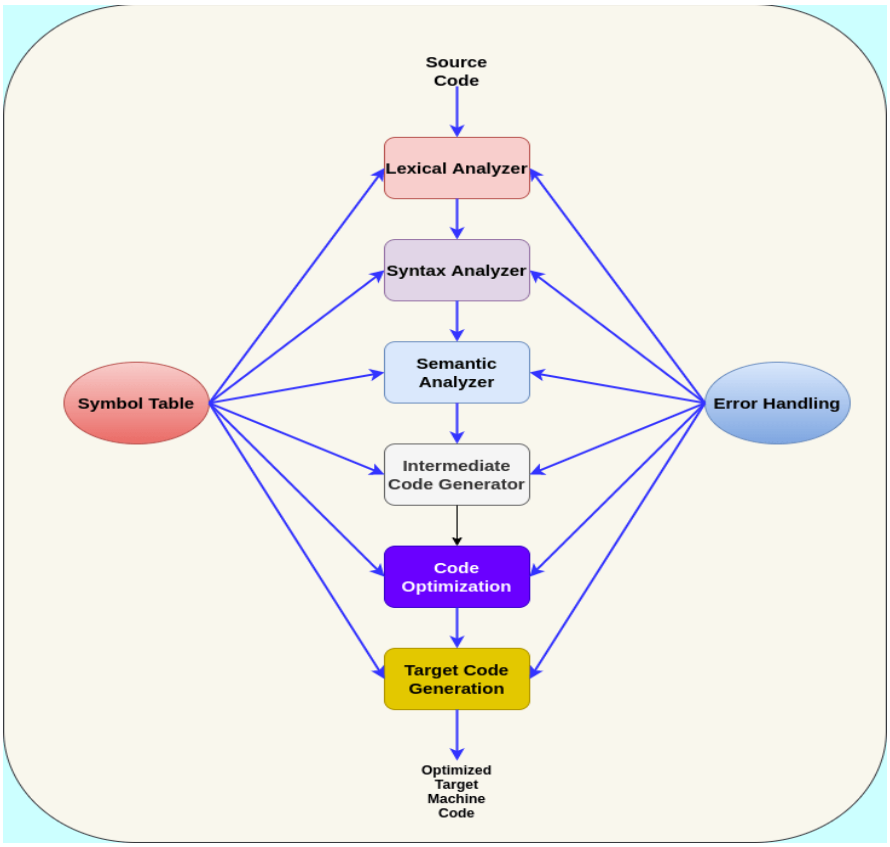
SYMBOL TABLE:

Symbol Table is an important data structure created and maintained by the compiler in order to keep track of semantics of variables i.e. it stores information about the scope and binding information about names, information about instances of various entities such as variable and function names, classes, objects, etc.

- It is built-in lexical and syntax analysis phases.
- The information is collected by the analysis phases of the compiler and is used by the synthesis phases of the compiler to generate code.
- It is used by the compiler to achieve compile-time efficiency.

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

- Lexical Analysis: Creates new table entries in the table, for example like entries about tokens.
- 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 the 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 the symbol table is associated with attributes that support the compiler in different phases.



Items stored in Symbol table:

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

Information used by the compiler from Symbol table:

- Data type and name
- Declaring procedures
- Offset in storage
- If structure or record then, a pointer to structure table.
- For parameters, whether parameter passing by value or by reference
- Number and type of arguments passed to function
- Base Address

Operations of Symbol table – The basic operations defined on a symbol table include:

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 structures for implementing 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 the beginning of the list till available pointer and if not found we get an error “use of the undeclared name”
- While inserting a new name we must ensure that it is not already present otherwise an error occurs i.e. “Multiple defined names”
- Insertion is fast O(1), but lookup is slow for large tables – O(n) on average
- The advantage is that it takes a minimum amount of space.

Linked List

- This implementation is using a linked list. A link field is added to each record.
- Searching of names is done in order pointed by the link of the link field.
- A pointer “First” is maintained to point to the first record of the symbol table.

- Insertion is fast O(1), but lookup is slow for large tables – O(n) on average

Hash Table

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

Binary Search Tree

- Another approach to implementing a symbol table is to use a binary search tree i.e. we add two link fields i.e. left and right child.
- All names are created as child of the root node that always follows the property of the binary search tree.
- Insertion and lookup are O(log2 n) on average.

Example 1:

```
void main()

{

int cat;

}
```

S.No.	Name	Type	Size	Address	Properties..
1.	cat	int	4	011011010	.....

Example 2:

```
extern double var(double a);     // Declare an external function
double fun(int count)            // Define a public function
{
   double sum = 0.0;
   for (int i = 1; i <= count; i++)
      sum += var((double) i);
   return sum;
}
```

The symbol table for the above code:

Symbol name	Type	Scope
var	function, double	extern
a	double	function parameter
fun	function, double	global
count	int	function parameter
sum	double	block local
i	int	for-loop statement