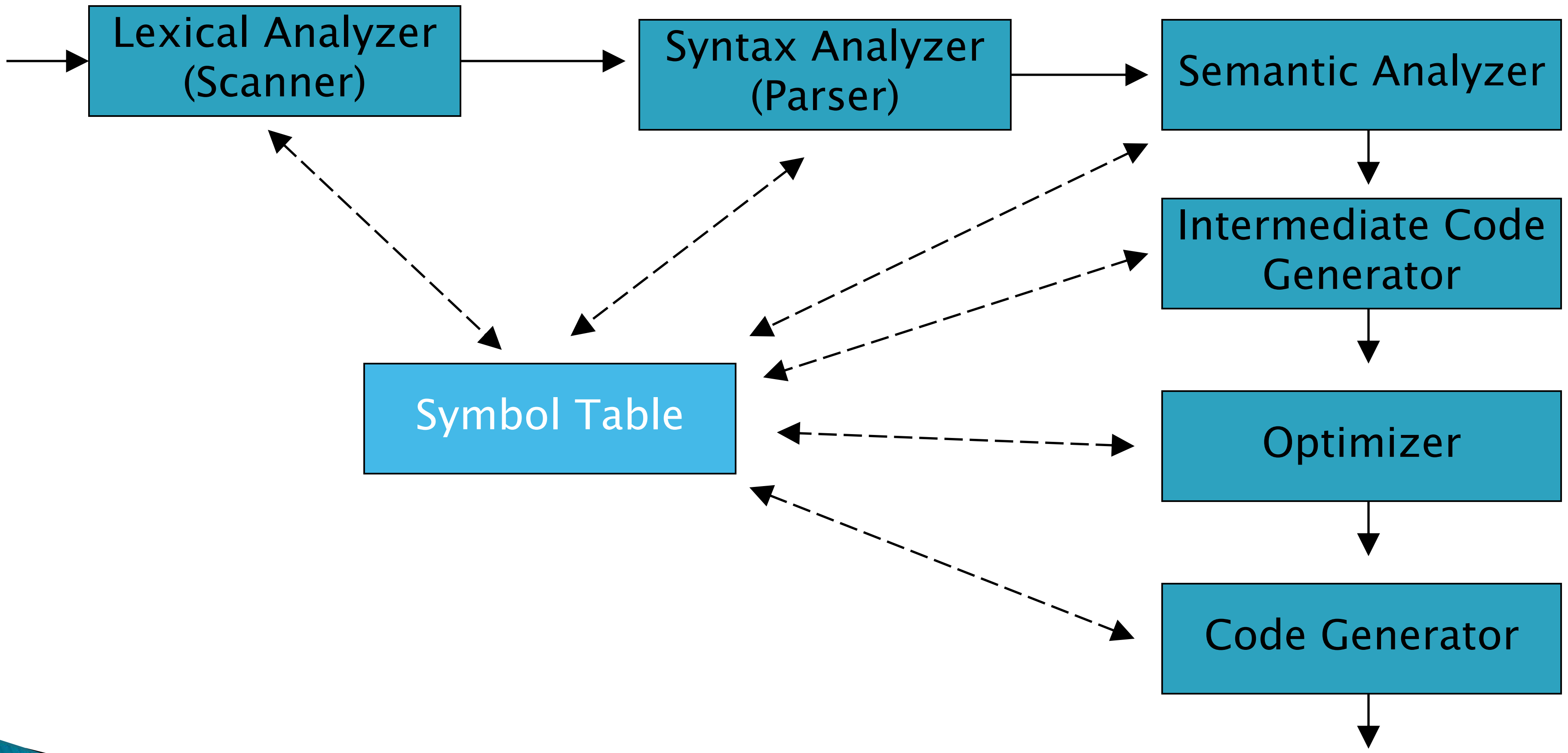
SYMBOL TABLE

Symbol Table



- we have seen how source code is analyzed as a series of tokens (lexical analysis), and how these tokens are analyzed as a structured program (syntactic analysis).
- Syntactic analysis checks identifiers are used appropriately WITHIN each statement (locally).
- Semantic analysis checks that identifiers are used appropriately within the program as a whole (globally).

- For semantic checking, we need to check whether:
  - Variable not declared multiple times
  - Variable declared before assigned
  - Variable assigned before referenced
  - Assignment compatible with declared type
  - Operations on variables compatible with type

Symbol Table

- Symbol Table is compile time data structure that is used by compiler to hold information about source program constructs. (Keep the track of the identifiers)
- A Symbol table is necessary component because
    - Declaration of identifiers appears once in a program.
    - Use of identifiers may appear in many places of the program

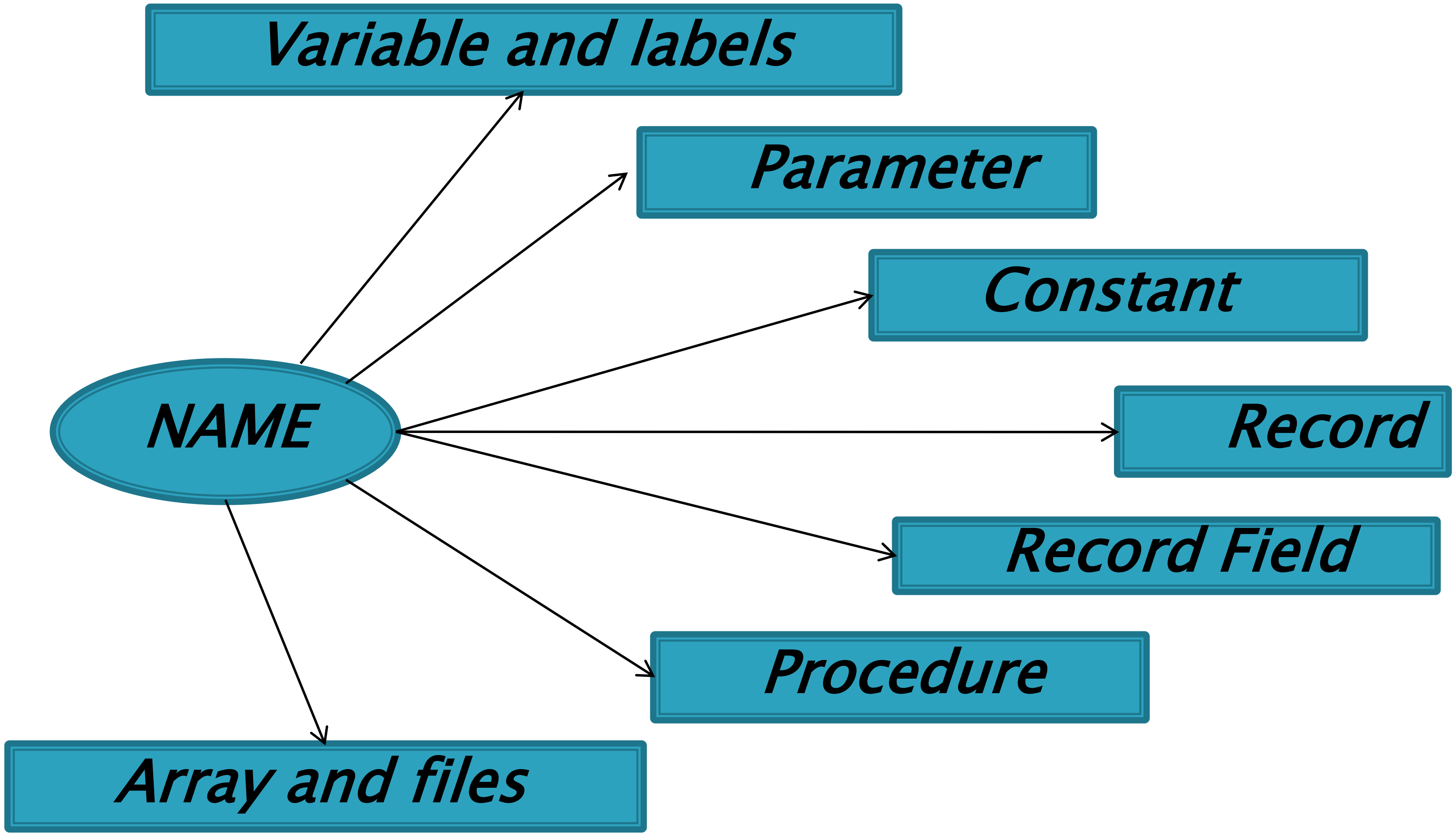
Symbol Table

- A compiler uses symbol table to keep track of scope and binding information about names.
- **Symbol table allows us to add new entries and find existing entries efficiently.**
- Symbol table is searched every time a name is encountered in the source program.
- Changes to the table occur if a new name or new information about an existing name is discovered.

Information Provided by Symbol Table

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

Symbol Table – Information



Symbol Table – Properties

- Each piece of information associated with a name is called Properties/ Attributes.
- Properties/ Attributes are language dependent.

Variable, Constants	Procedure or function	Array
• Type , Line number where declared , Lines where referenced , Scope	• Number of parameters, parameters themselves, Scope of parameters, return type.	• # of Dimensions, Array bounds, Type of Array

Who creates Symbol Table??

- Identifiers and attributes are entered by the analysis phases when processing a 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. ~~lexical analysis time~~ at time of lexical analysis
- In block-structured languages with scopes and explicit declarations:
  - The parser and/or semantic analyzer enter identifiers and corresponding attributes.

Use of Symbol Table

imp

- Symbol table information used by analysis and synthesis phase of compiler.
- To verify that used identifiers have been declared.
- To verify that expression and assignments are semantically correct. (Type Checking)
- To generate intermediate or target code. (ICG)

Constructing Symbol Table

- There are three main operations to be carried out on the symbol table :
    - Searching whether a string has already been stored.
    - Inserting an entry for a new string.
    - Deleting a string when it goes out of scope.
- Corresponding three Functions :
    - lookup(s) : if string s available in ST then returns its index else returns 0.
    - insert(s, t) : add new entry for string s with token t in ST and returns its index.
    - delete(s) : delete string s from ST.

Continue…

- Other operations :
  - allocate : to allocate a new empty symbol table
  - free : to remove all entries and free the storage of symbol table.
  - set_attribute : to associate an attribute with give entry.
  - get_attribute : to get an attribute associated with a given entry.

Example 1

```
defproc myproc (int A, float B) .
   int D, E;
   D = 0;
   E = A / round(B);
   if (E > 5) {
         print D
   }
}
```

Example 1

```
defproc myproc (int A, float B) {
  int D, E;
  D = 0;
  E = A / round(B);
  if (E > 5) {
      print D
  }
}
```

Symbol Table

Symb	Token	Dtype	Init?

id means IDENTIFIER

Example 1

```
defproc myproc (int A, float B) {
  int D, E;
  D = 0;
  E = A / round(B);
  if (E > 5) {
      print D
  }
}
```

Symbol Table

Symb	Token	Dtype	Init?
myproc	id	procname	-

Event: identifier = procedure name

Action: Add name to symbol name

Example 1

A and B both are intialized because we are passing argument value inside main function at time of function call

```
defproc myproc (int A, float B) {
  int D, E;
  D = 0;
  E = A / round(B);
  if (E > 5) {
      print D
  }
}
```

Symbol Table

Symb	Token	Dtype	Init?
myproc	id	procname	-
A	id	int	yes

Event: identifier = variable declaration, function arg

Action: Add name to symbol name, as initialised

Example 1

```
defproc myproc (int A, float B) {
  int D, E;
  D = 0;
  E = A / round(B);
  if (E > 5) {
      print D
  }
}
```

Symbol Table

Symb	Token	Dtype	Init?
myproc	id	procname	-
A	id	int	yes
B	id	float	yes

Event: identifier = variable declaration, function arg

Action: Add name to symbol name, as initialised

Example 1

```
defproc myproc (int A, float B) {
  int D, E;
  D = 0;
  E = A / round(B);
  if (E > 5) {
      print D
  }
}
```

Symbol Table

Symb	Token	Dtype	Init?
myproc	id	procname	-
A	id	int	yes
B	id	float	yes

Event: identifier = variable declaration

Check: Already in symbol table? if so, fail

Else: Add name to symbol name, not initialised

Example 1

```
defproc myproc (int A, float B) {
  int D, E;
  D = 0;
  E = A / round(B);
  if (E > 5) {
      print D
  }
}
```

Symbol Table

Symb	Token	Dtype	Init?
myproc	id	procname	-
A	id	int	yes
B	id	float	yes
D	id	int	no

Event: identifier = variable declaration

Check: Already in symbol table? if so, fail

Else: Add name to symbol name, not initialised

Example 1

```
defproc myproc (int A, float B) {
  int D, E;
  D = 0;
  E = A / round(B);
  if (E > 5) {
      print D
  }
}
```

Symbol Table

Symb	Token	Dtype	Init?
myproc	id	procname	-
A	id	int	yes
B	id	float	yes
D	id	int	no

Event: identifier = variable declaration

Check: Already in symbol table? if so, fail

Else: Add name to symbol name, not initialised

Example 1

```
defproc myproc (int A, float B) {
  int D, E;
  D = 0;
  E = A / round(B);
  if (E > 5) {
      print D
  }
}
```

Symbol Table

Symb	Token	Dtype	Init?
myproc	id	procname	-
A	id	int	yes
B	id	float	yes
D	id	int	no
E	id	int	no

Event: identifier = variable declaration

Check: Already in symbol table? if so, fail

Else: Add name to symbol name, not initialised

Example 1

```
defproc myproc (int A, float B) {
  int D, E;
  D = 0;
  E = A / round(B);
  if (E > 5) {
      print D
  }
}
```

Symbol Table

Symb	Token	Dtype	Init?
myproc	id	procname	-
A	id	int	yes
B	id	float	yes
D	id	int	no
E	id	int	no

Event: identifier = variable assignment

Check: ERROR if not in symbol table

Action: find entry in ST and set initialised

Example 1

```
defproc myproc (int A, float B) {
  int D, E;
  D = 0;
  E = A / round(B);
  if (E > 5) {
      print D
  }
}
```

Symbol Table

Symb	Token	Dtype	Init?
myproc	id	procname	-
A	id	int	yes
B	id	float	yes
D	id	int	YES
E	id	int	YES

Event: identifier = variable assignment

Action: find entry in ST and set initialised

Example 1

```
defproc myproc (int A, float B) {
  int D, E;
  D = 0;
  E = A / round(B);
  if (E > 5) {
      print D
  }
}
```

Symbol Table

Symb	Token	Dtype	Init?
myproc	id	procname	-
A	id	int	yes
B	id	float	yes
D	id	int	YES
E	id	int	YES

Event: identifier = variable reference

Check: report ERROR if not in symbol table

Check: report ERROR if not initialised

Action: none

Symbol Table Data Structure

- Insertion is done only once.
- Lookup is done many times. So efficient look up operation is required.
- The data structure should be designed to allow the compiler to find the record for each name quickly and to store or retrieve data from that record quickly.

Symbol Table Implementation Techniques

- Unordered list
- Ordered list
- Binary Search Tree
- Hash Table

Unordered list

- Simplest to implement.
- Implemented as array or link list.
- Array : static
- Link list : dynamic
- Insertion becomes fast O(1), but lookup is slow for large table O(n) on average.

Ordered list

- If array/link list is sorted, it can be searched using binary search $O(\log_2 n)$.
- Insertion into a sorted array/link list is expensive :: O(n) on average.
- Lookup operation is efficient.

Binary Search Tree

- Grow dynamically
- Insertion and lookup both are $O(\log_2 n)$ on average.

Hash Table

- A hash table is an array with index range:

  0 to TableSize – 1
- Most commonly used data structure to implement symbol tables
- Insertion and lookup can be made very fast – O(1)

Example 2

```
01  PROGRAM Main
02      GLOBAL a, b
03      PROCEDURE P (PARAMETER x)
04              LOCAL a
05      BEGIN {P}
06              …a…
07              …b…
08              …x…
09      END  {P}
10  BEGIN{Main}          program execution starts form here
11      Call P(a)
12 END {Main}
```

Unordered list

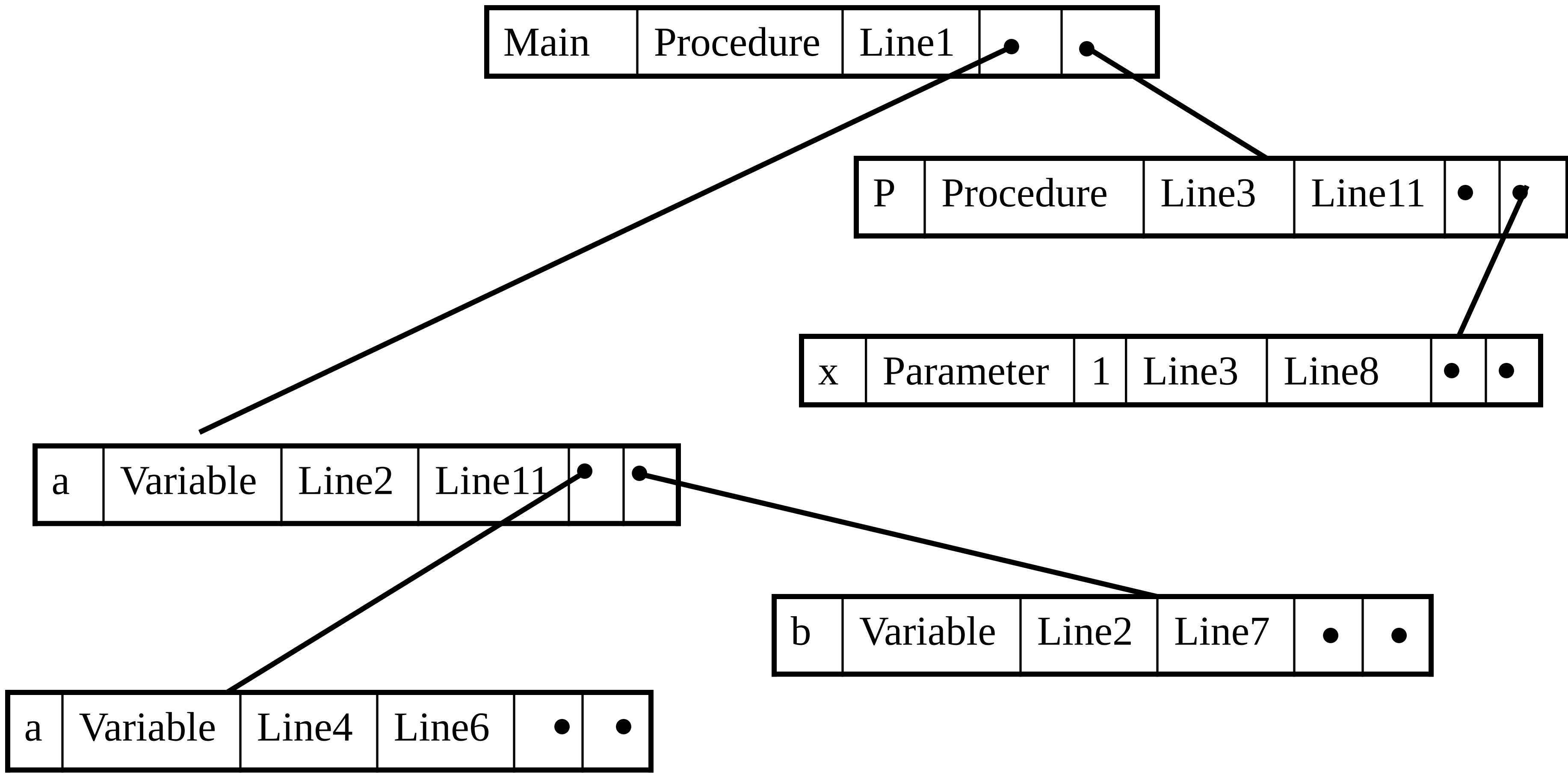
Key	Symbol	Token	Data Type	Initialize	Declared Line #	Referenced Line #	Parameters
1	Main	Id: Procedure			1		No
2	a	Id : Variable		No	2	11	
3	b	Id : Variable		No	2	7	
4	P	Id : Procedure			3	11	Yes 1 , x
5	x	Id : Variable		No	3	8	
6	a	Id : Variable		No	4	6	

Ordered list (sorted)

Key	Symbol	Token	Data Type	Initialize	Declared Line #	Referenced Line #	Parameters
1	a	Id : Variable		No	2	11	
2	a	Id : Variable		No	4	6	
3	b	Id : Variable		No	2	7	
4	Main	Id : Procedure			1		No
5	P	Id : Procedure			3	11	Yes 1 , x
6	x	Id : Variable		No	3	8	

Binary Search Tree

how to insert data in binary search tree see in geeksforgeeks

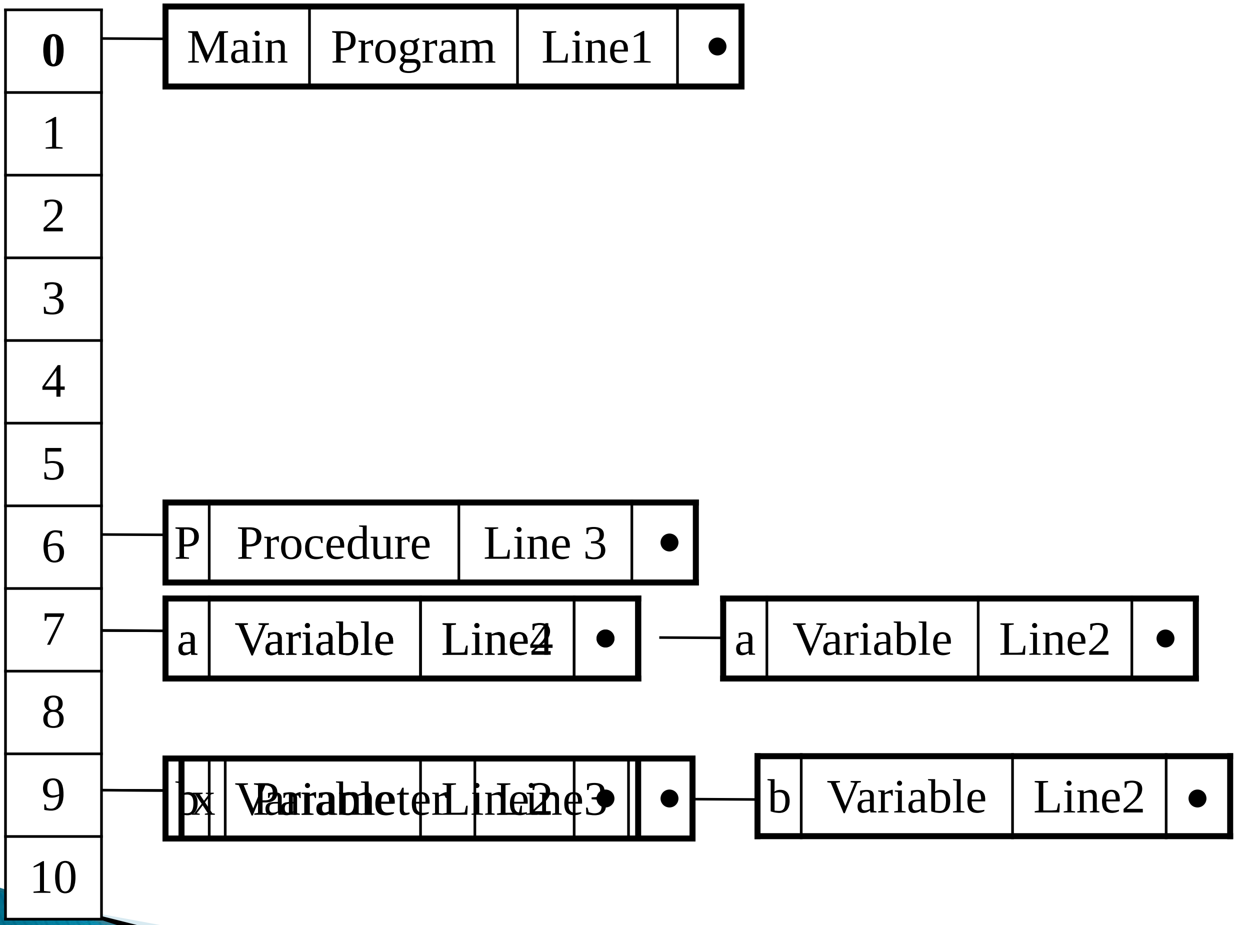


Main | Procedure | Line1 | • | •
P | Procedure | Line3 | Line11 | • | •
x | Parameter | 1 | Line3 | Line8 | • | •
a | Variable | Line2 | Line11 | • | •
b | Variable | Line2 | Line7 | • | •
a | Variable | Line4 | Line6 | • | •

Hash Table

M	n	a	b	P	x
77	110	97	98	80	120

H(Id) = (# of first letter + # of last letter) mod 11



How ST differs from other DS?

- ST differs from other data structure in method of accessibility.
- Other data structures are index/pointer accessible, where symbol table is context/content accessible.

Questions:

- What is symbol table? Why it is used? Which phase of compiler set the attributes of token in a symbol table?
- Explain the significance of symbol table, how it is generated and where it generated and how it is used.
- Discuss the importance of symbol table in compiler design. How is the symbol table manipulated at various phases of compilation? What should be the typical entries in symbol table for C language?
- What information does symbol table? Write down about the data structures used to represent the symbol table.
- [redacted] When the symbol table entries are used?
- How symbol table differs from other data structure?

significance means mahatva