

# Recitation 9: Symbol Tables

---

COP3402 FALL 2015 – ARYA POURTABATABAIE  
FROM EURIPIDES MONTAGNE, FALL 2014

# Why Symbol Tables?

---

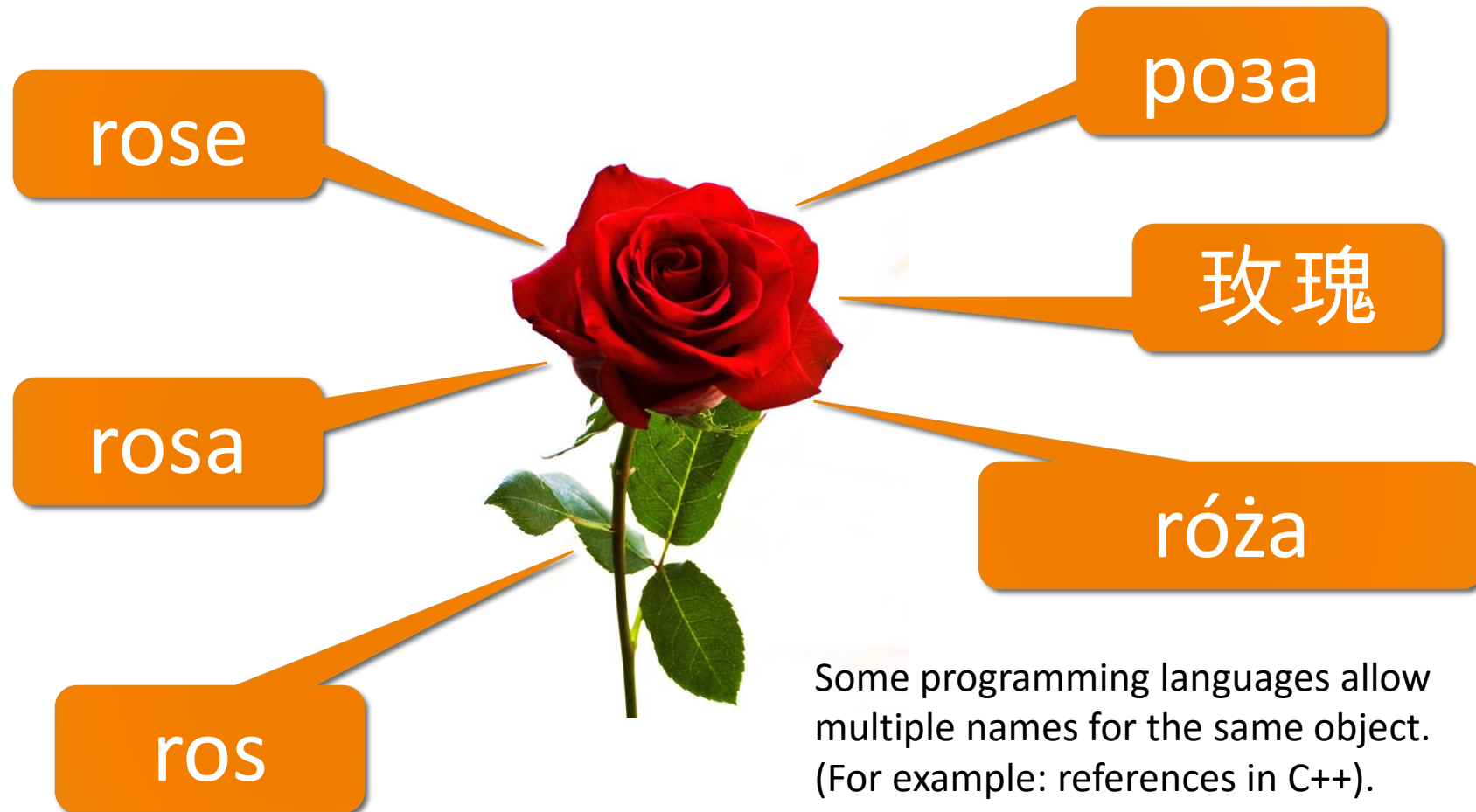
*What's in a name?  
That which we call a  
rose, by any other  
name would smell  
as sweet...*



*Romeo and Juliet* by Henri-Pierre Picou

# Why Symbol Tables?

---



Some programming languages allow multiple names for the same object. (For example: references in C++).

# Why Symbol Tables?

---



rose

rose

rose

rose?



The same word could mean different objects in different contexts.

(For example: a local variable and a global variable).

# Symbol Tables

---

It records information about *symbol names* in a program.

Don't confuse *symbol* and *identifier*:

- A **symbol** (or **name**) is the object itself (variable, function, procedure, program, etc).
- An **identifier** is a way to reference some symbol.



# When is the Symbol Table used?

---

- Lexical Analysis time
  - Lexical Analyzer scans program
  - Finds Symbols
  - Adds Symbols to symbol table
- Syntactic Analysis Time
  - Information about each symbol is filled in
    - (Your assignments just do all of it here, because the lexical analyzer is able to tell essentially nothing about the symbol – even its scope)
- Used for type checking during semantic analysis

# Info provided by Symbol Table

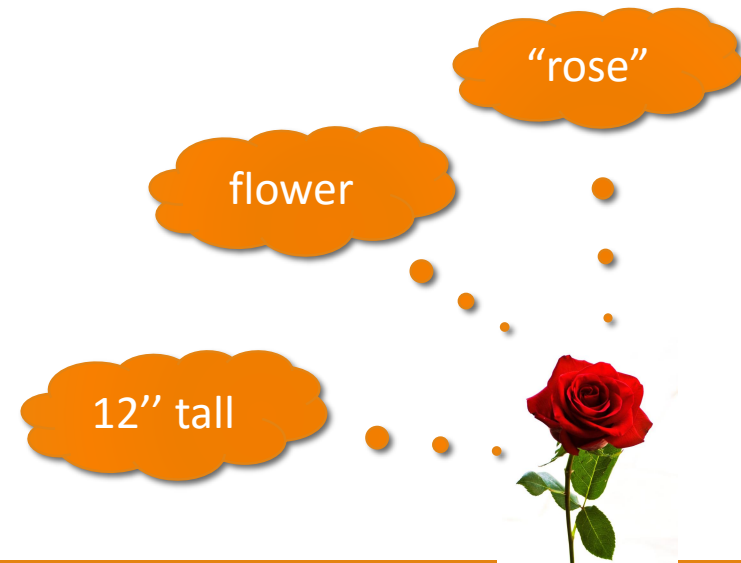
---

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

# Symbol Attributes

---

- Each piece of info associated with a name is called an **attribute**.
- Attributes are language dependent:
  - Actual characters of the name (“rose”).
  - Type (variable, function, program, etc).
  - Storage allocation info (number of bytes).
  - Line number where declared.
  - Lines where referenced.
  - Scope.





# Symbol Classes

---

- Different Classes of Symbols have different Attributes.
- Variable, Type, Constant, parameter, record field.
  - Type is one of attributes (*int, float, char*).
- Procedure or function.
  - Number of parameters, parameters themselves, result type.
- Array
  - # of Dimensions, Array bounds.
- File
  - Record size, record type.

# Other Attributes

---

- A scope of a variable can be represented by
  - A number (scope is just one of the attributes).
  - Different symbol tables for different scopes.
- Object Oriented languages have classes like
  - Method names, class names, object names.
  - Scoping is VERY important. (Inheritance).
- Functional Languages (like LISP)
  - Binding Issues.

# Symbol Table Operations

---

Two operations required:

- **Insert:** adds a symbol to the table.
- **Lookup:** finds a symbol in the table (and get its attributes).

Insertion is only done once per symbol.

Lookup is done many times per symbol.

We need fast lookups.

# Example program

---

```
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}
11      Call P(a)
12  END {Main}
```

# Symbol Table: External Structure

---

The way we handle the symbols algorithmically. Four basic methods:

- Unordered List
- Ordered List
- Binary Tree
- Hash Table

# Symbol Table: Unordered List

---

Fast inserts:  $O(1)$

Slow lookups:  $O(n)$

Only useful if there is a small number of symbols (less than a couple dozen).

Identifier	Class	Scope
Main	Program	0
a	Variable	0
b	Variable	0
P	Procedure	0
x	Parameter	1
a	Variable	1

# Symbol Table: Ordered List

---

Ordered by identifier.

Ordered array:

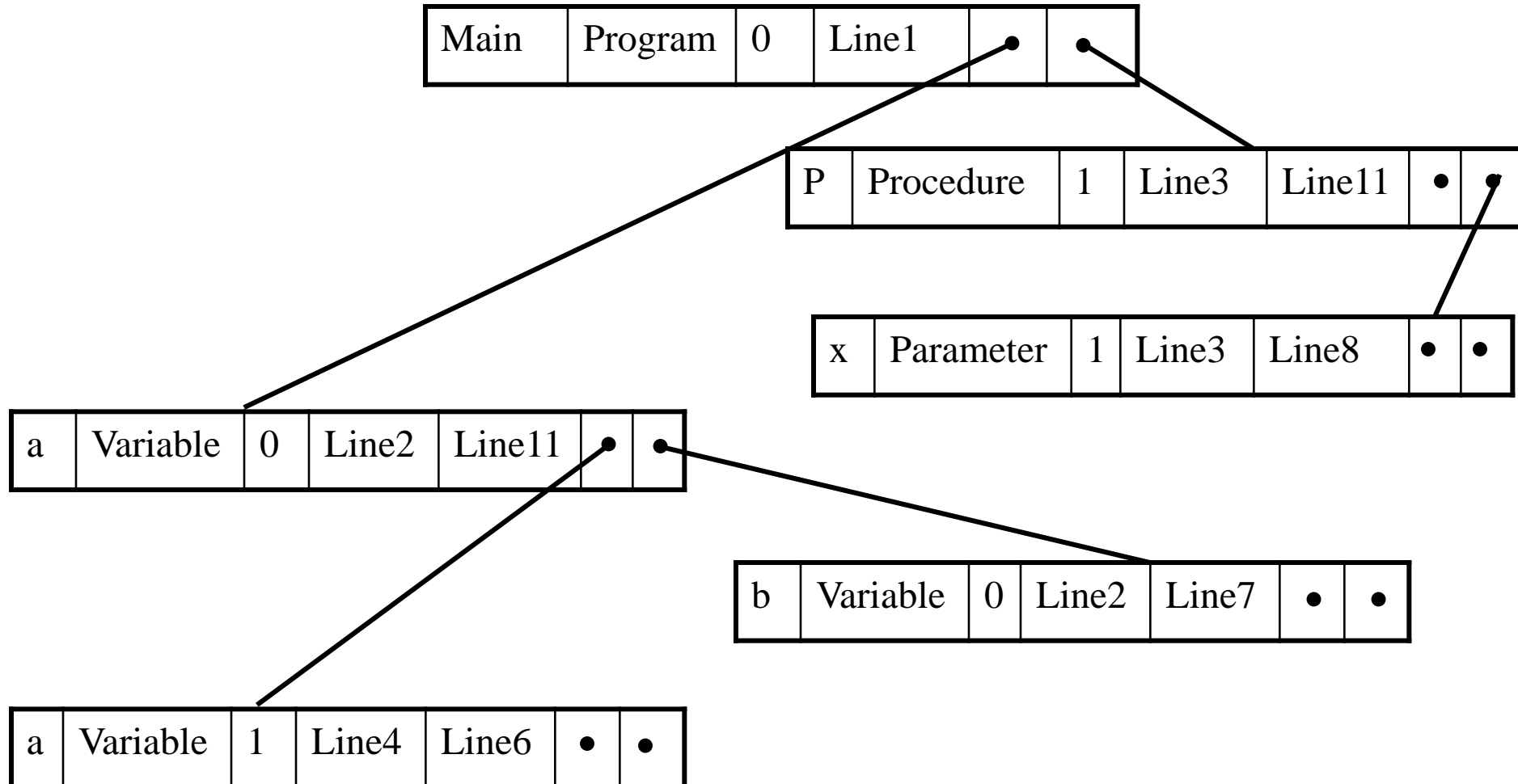
- Slow inserts:  $O(n)$
- Fast lookups:  $O(\log n)$

Linked list:

- Slow inserts:  $O(n)$
- Slow lookups:  $O(n)$

Identifier	Class	Scope
a	Variable	0
a	Variable	1
b	Variable	0
Main	Program	0
P	Procedure	0
x	Parameter	1

# Symbol Table: Binary Tree





# Symbol Table: Binary Tree

---

- Fast inserts:  $O(\log n)$
- Fast lookups:  $O(\log n)$
- Space efficient.
- Easy to print alphabetized list of names.
- Scoping is difficult, unless a different tree is used for each scope.

# Symbol Table: Hash Table

---

Most efficient. Used by production compilers.

Fast insertion:  $O(1)$ .

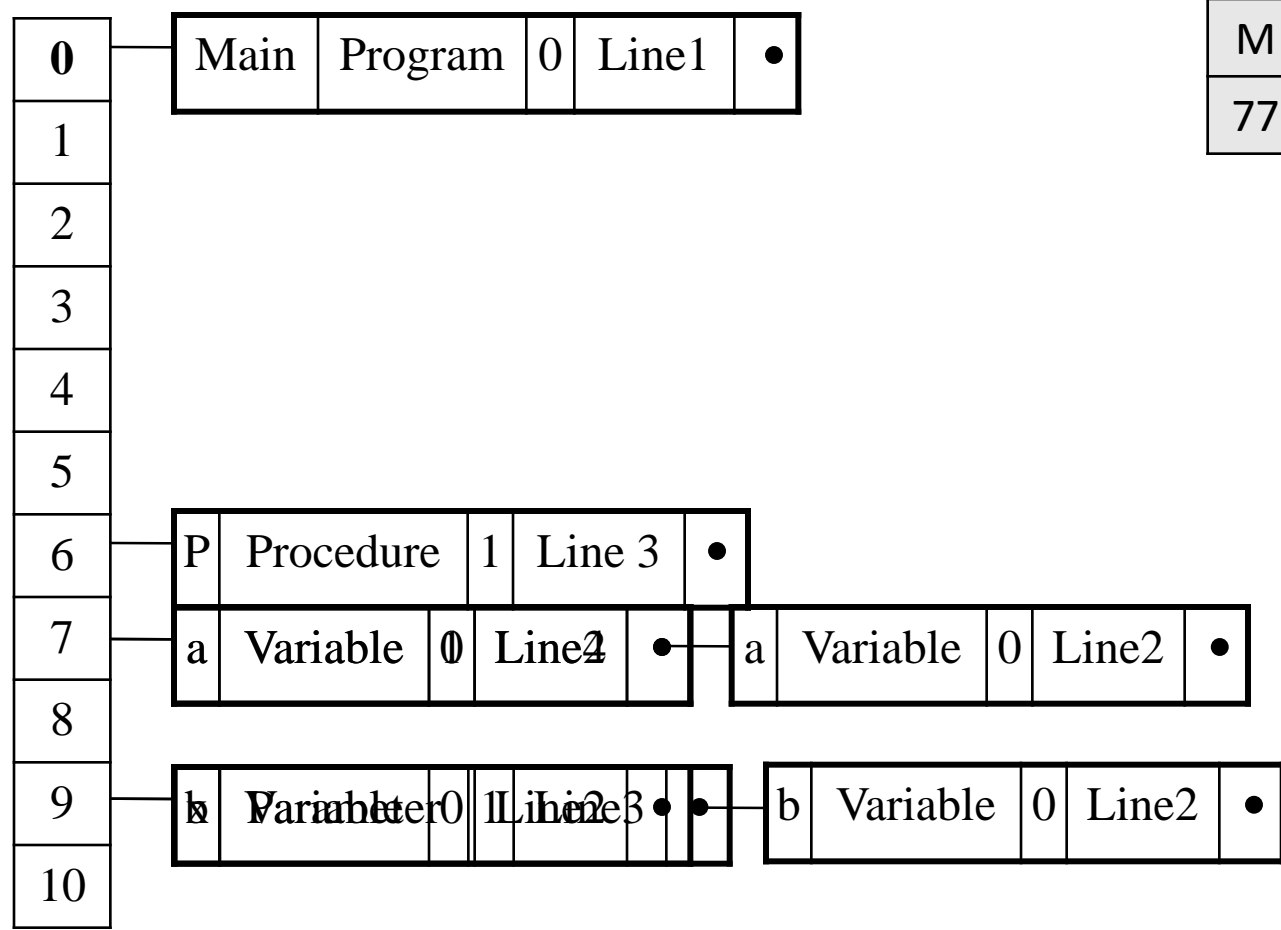
Fast lookup:  $O(1)$  best case,  $O(n)$  worst case (very rare).

A good hashing function is needed.

As an example, let's use the following hashing function:

- $H(id) = (\text{First letter} + \text{last letter}) \bmod 11$
- *This is not a good hashing function – it's an example.*

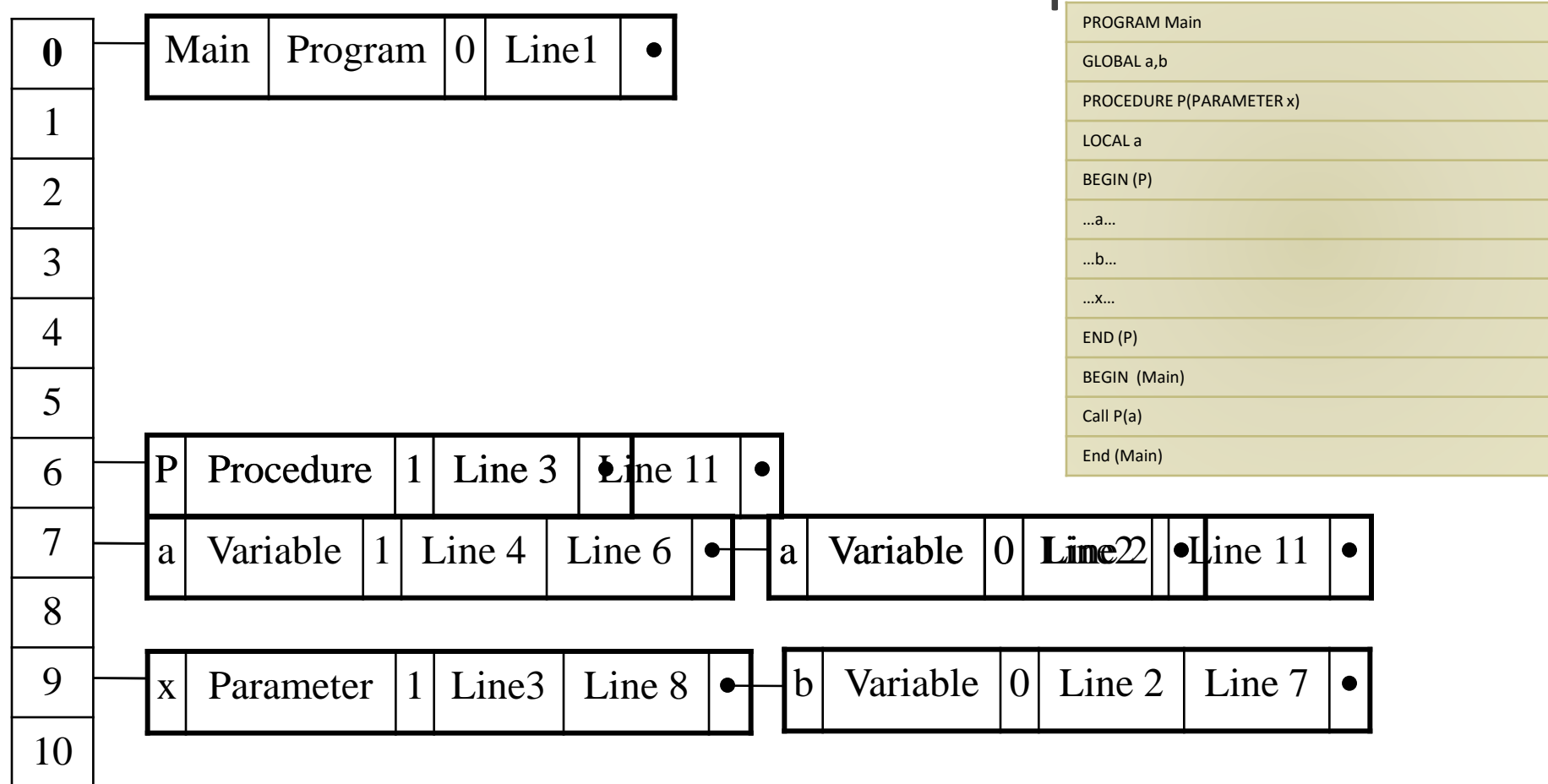
# Symbol Table: Hash Table insert



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

```
PROGRAM Main
GLOBAL a,b
PROCEDURE P(PARAMETER x)
LOCAL a
BEGIN (P)
...a...
...b...
...x...
END (P)
BEGIN (Main)
Call P(a)
End (Main)
```

# Symbol Table: Hash Table update



# Symbol Table: Hash Table

---

Scooping is easy to implement. No need to use extra tables.

Drawbacks?

- It is not as space efficient as a binary tree.

# Symbol Table: Internal Structure

---

The internal structure is how we organize each symbol and its attributes.

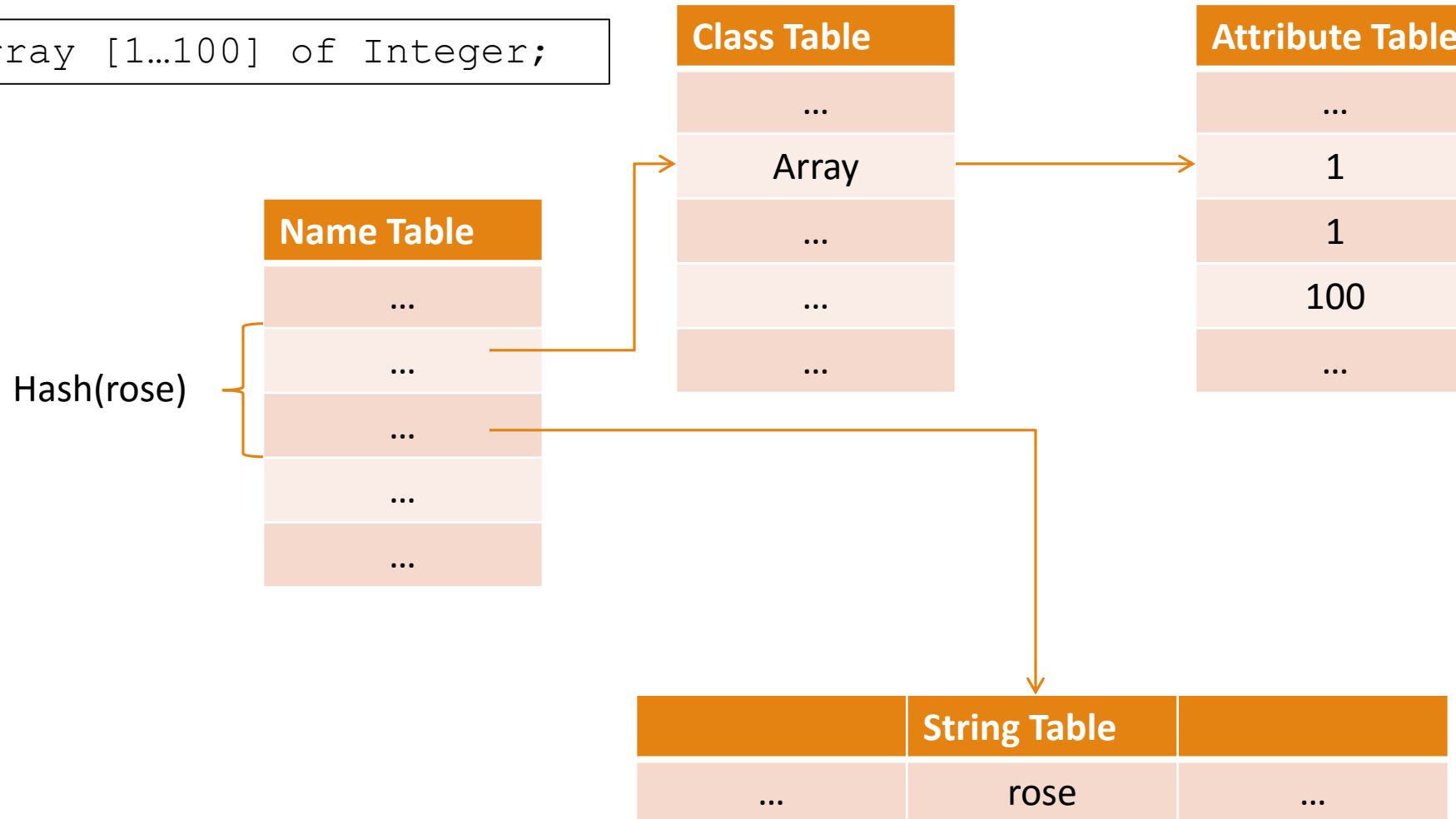
Logical view: a symbol table is a list of names, and each name has a list of attributes.

Implementation: a symbol table might have multiple tables:

- String table.
- Class table.
- Name table.

# Example of Internal Structure

```
rose: Array [1...100] of Integer;
```



# Questions?

---