Menoufia University

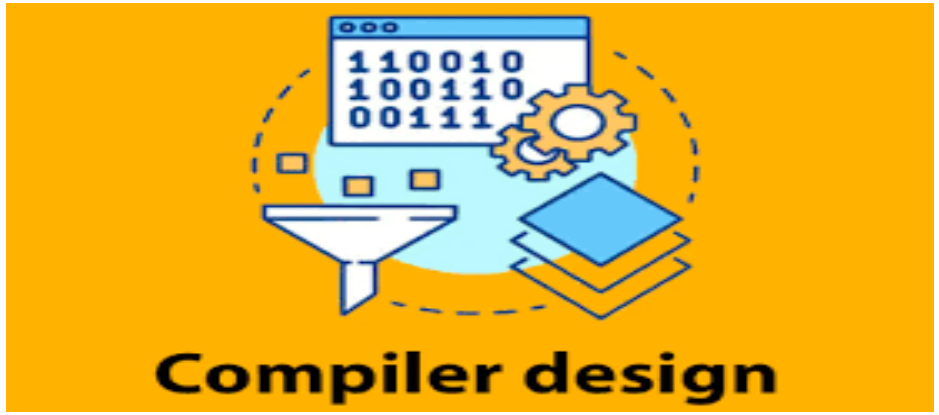
Faculty of computers & Information

Computer Science Department.

Compiler Design

4 Year – first Semester

Lecture 8



DR. Eman Meslhy Mohamed

Lecturer at Computer Science department

2023-2024

Agenda

- Symbol Table
- Hash Table
- DS for symbols tables
- Scope Information

Symbol Table

- Symbol table is an important data structure used in a compiler.
- Symbol table is used to store the information about the occurrence of various entities such as objects, classes, variable name, interface, function name etc. it is used by both the analysis and synthesis phases.
- The symbol table used for following purposes:
  - It is used to store the name of all entities in a structured form at one place.
  - It is used to verify if a variable has been declared.
  - It is used to determine the scope of a name.
  - It is used to implement type checking by verifying assignments and expressions in the source code are semantically correct.

Symbol Table

- A symbol table can either be linear or a hash table. Using the following format, it maintains the entry for each name.

<symbol name, type, attribute>

- For example, suppose a variable store the information about the following variable declaration:

static int salary

- then, it stores an entry in the following format:

<salary, **int**, **static**>

- The clause attribute contains the entries related to the name.

Implementation

- The symbol table can be implemented in the unordered list if the compiler is used to handle the small amount of data.
- A symbol table can be implemented in one of the following techniques:
  - Linear (sorted or unsorted) list
  - **Hash table**
  - Binary search tree
- Symbol table are mostly implemented as hash table.

Symbol Table operations

- The symbol table provides the following operations:
  - Insert ()
- Insert () operation is more frequently used in the analysis phase when the tokens are identified and names are stored in the table.
- The insert() operation is used to insert the information in the symbol table like the unique name occurring in the source code.
- The insert () function takes the symbol and its value in the form of argument.
- For example:

  int x;
- Should be processed by the compiler as: insert (x, **int**)

Symbol Table operations

- lookup()
  - In the symbol table, lookup() operation is used to search a name. It is used to determine:
  - The existence of symbol in the table.
  - The declaration of the symbol before it is used.
  - Check whether the name is used in the scope.
  - Initialization of the symbol.
  - Checking whether the name is declared multiple times.
- The basic format of lookup() function is as follows:

lookup (symbol)

- This format is varies according to the programming language.

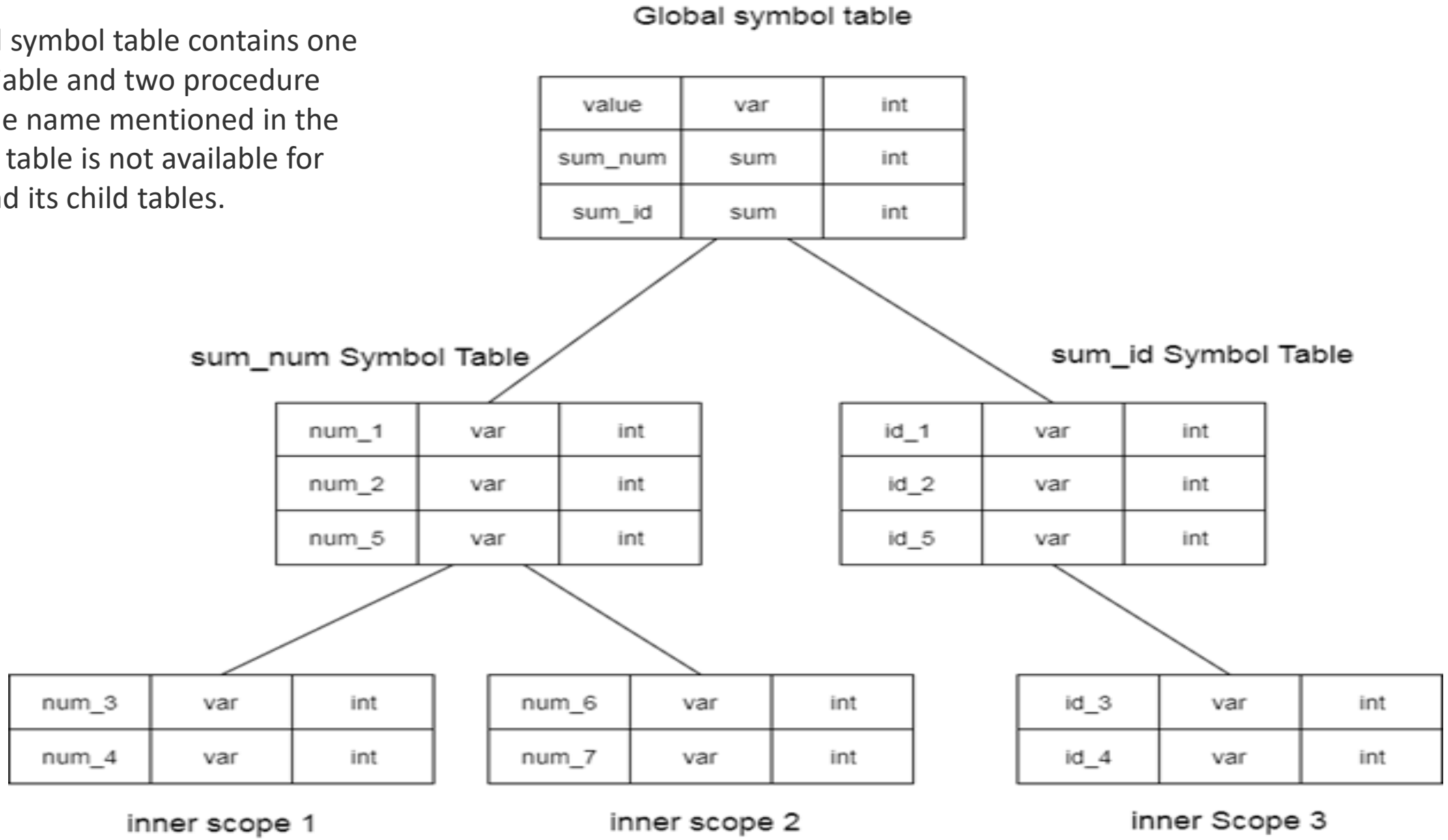
Data structure for symbol table

- A compiler contains two type of symbol table: global symbol table and scope symbol table.
- Global symbol table can be accessed by all the procedures and scope symbol table.
- A compiler maintains multiple block levels of symbol tables:
  - Level 0: A null hash table at level 0
  - Level 1: Keyword in the hash table at level 1
  - Level 2: Global symbol table which can be accessed by all the procedures
  - Level 3: Scope symbol tables that are created for each scope in the program

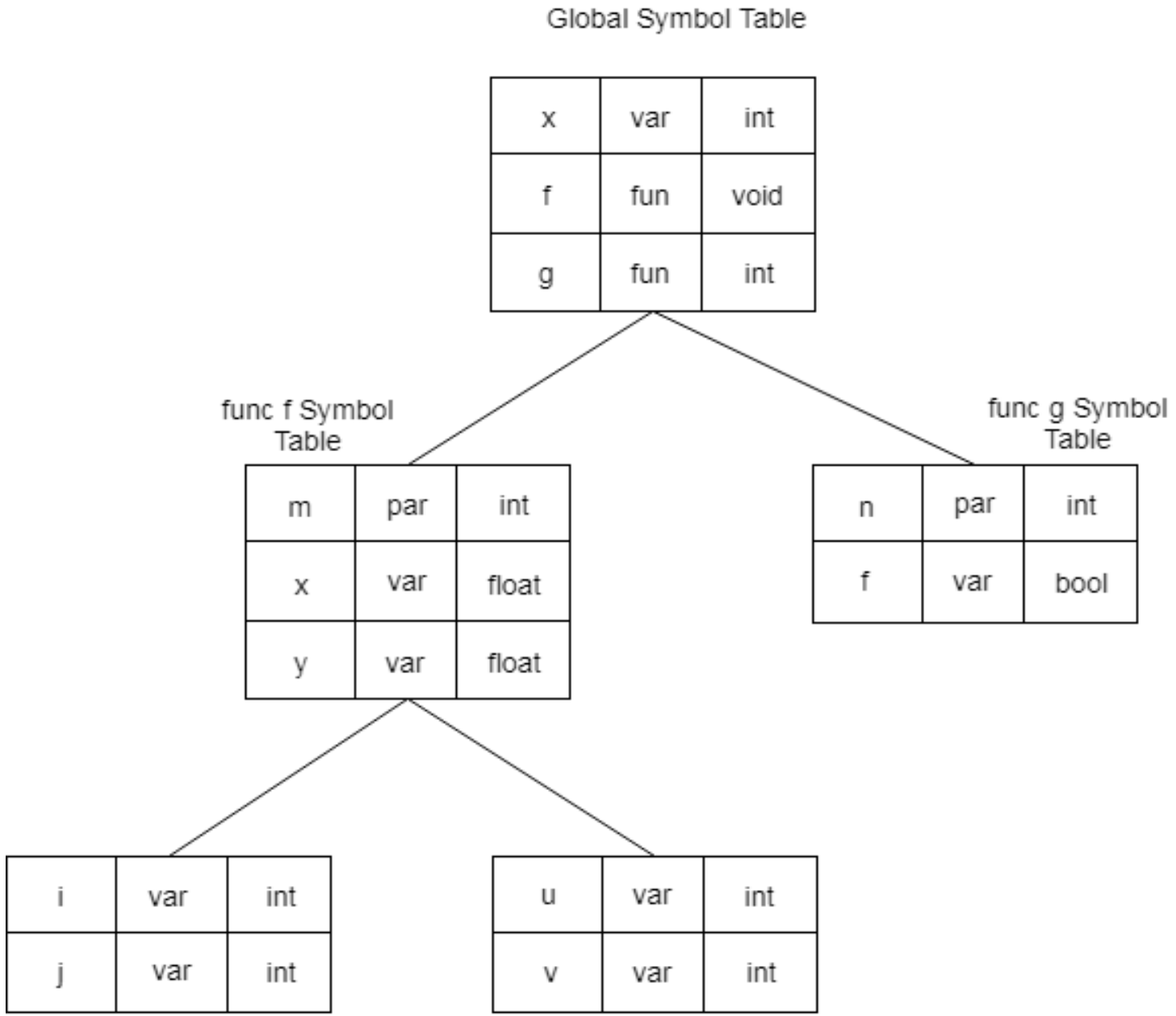
Example

```
1.int value=10;
2.
3.int sum_num()
4.    {
5.      int num_1;
6.      int num_2;
7.
8.          {
9.           int num_3;
10.            int num_4;
11.           }
12. int num_5;
13.          {
14.           int_num 6;
15.           int_num 7;
16.          }
17.    }
18. int  sum_id
19.    {
20.      int id_1;
21.      int id_2;
22.          {
23.           int id_3;
24.           int id_4;
25.          }
26.      int id_5;
27.    }
```

The global symbol table contains one global variable and two procedure names. The name mentioned in the sum_num table is not available for sum_id and its child tables.



```
1.nt x;
2.void f(int m) {
3.     float x, y;
4.{
5.          int i, j;
6.          int u, v;
7.}
8.}
9.int g (int n)
10.{
11.      bool t;
12.}
```



Search

- Data structure hierarchy of symbol table is stored in the semantic analyzer. If you want to search the name in the symbol table then you can search it using the following algorithm:
  - First a symbol is searched in the current symbol table.
  - If the name is found then search is completed else the name will be searched in the symbol table of parent until,
  - The name is found or global symbol is searched.

Hash Table

- **Searching Techniques-**
- In data structures, There are several searching techniques like linear search, binary search, search trees etc.

- In these techniques, time taken to search any particular element depends on the total number of elements.
- **Example-**

- **Linear Search** takes O(n) time to perform the search in unsorted arrays consisting of n elements.
- **Binary Search** takes O(logn) time to perform the search in sorted arrays consisting of n elements.
- It takes O(logn) time to perform the search in **Binary Search Tree** consisting of n elements.
- **Drawback-** The main drawback of these techniques is-
  - As the number of elements increases, time taken to perform the search also increases.
  - This becomes problematic when total number of elements become too large.

Hashing in DS

- In data structures, Hashing is a well-known technique to search any particular element among several elements.

- It minimizes the number of comparisons while performing the search.
- Hashing is the process of mapping large amount of data item to smaller table with the help of hashing function.
- Hashing is also known as **Hashing Algorithm** or **Message Digest Function**.
- It is a technique to convert a range of key values into a range of indexes of an array.
- It is used to facilitate the next level searching method when compared with the linear or binary search.
- Hashing allows to update and retrieve any data entry in a constant time O(1).

Hashing in DS

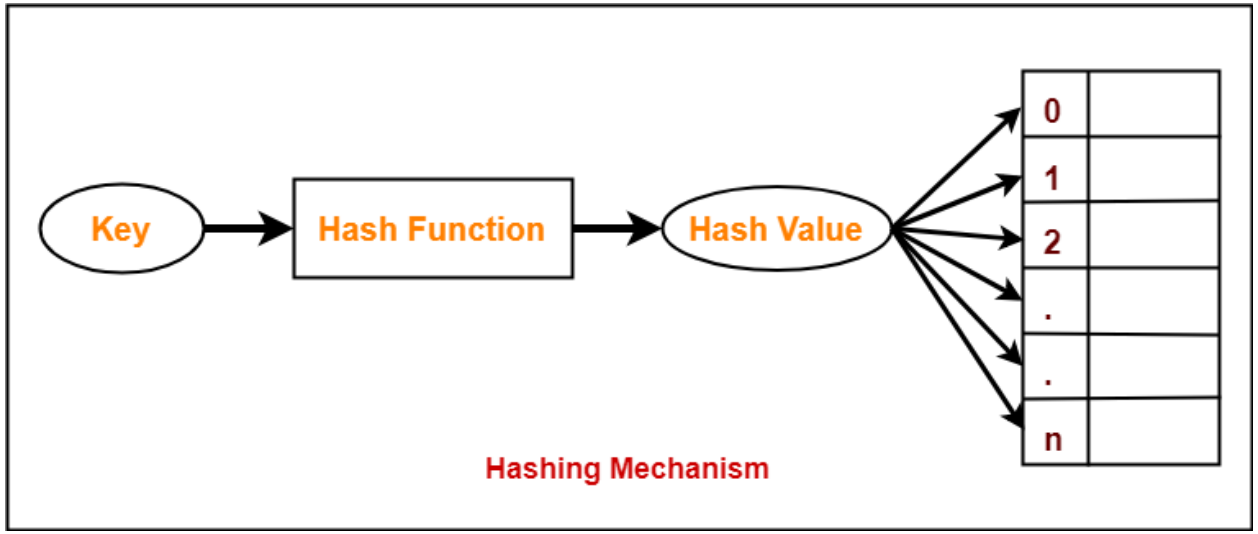
- Constant time O(1) means the operation does not depend on the size of the data.
- Hashing is used with a database to enable items to be retrieved more quickly.
  - **Advantage-**
  - Unlike other searching techniques,
- Hashing is extremely efficient.
- The time taken by it to perform the search does not depend upon the total number of elements.
- It completes the search with constant time complexity O(1).

Hashing Mechanism

- In hashing, An array data structure called as **Hash table** is used to store the data items.
- Based on the hash key value, data items are inserted into the hash table.
  - **Hash Key Value-** Hash key value is a special value that serves as an index for a data item.
- It indicates where the data item should be be stored in the hash table.
- Hash key value is generated using a hash function.
  - **Hash Function-**
  - Hash function is a function that maps any big number or string to a small integer value.
- Hash function takes the data item as an input and returns a small integer value as an output.
- The small integer value is called as a hash value.
- Hash value of the data item is then used as an index for storing it into the hash table.
- A fixed process converts a key to a hash key is known as a **Hash Function.**

Hashing Mechanism

- This function takes a key and maps it to a value of a certain length which is called a **Hash value** or **Hash.**
- Hash value represents the original string of characters, but it is normally smaller than the original.
- It transfers the digital signature and then both hash value and signature are sent to the receiver. Receiver uses the same hash function to generate the hash value and then compares it to that received with the message.
- If the hash values are same, the message is transmitted without errors



- **Types of Hash Functions-**
- There are various types of hash functions available such as-

1. Mid Square Hash Function
2. Division Hash Function
3. Folding Hash Function etc

- It depends on the user which hash function he wants to use.
- **Properties of Hash Function-**
- The properties of a good hash function are-

- It is efficiently computable.
- It minimizes the number of collisions.
- It distributes the keys uniformly over the table.

➢ Hashing is a well-known searching technique.

➢ It minimizes the number of comparisons while performing the search.

➢ It completes the search with constant time complexity O(1).

Hash Table

- Hash table or hash map is a data structure used to store key-value pairs.
- It is a collection of items stored to make it easy to find them later.
- It uses a hash function to compute an index into an array of buckets or slots from which the desired value can be found.
- It is an array of list where each list is known as bucket.
- It contains value based on the key.
- Hash table is used to implement the map interface and extends Dictionary class.
- Hash table is synchronized and contains only unique elements.

Fig. Hash Table

- The above figure shows the hash table with the size of n = 10. Each position of the hash table is called as **Slot**. In the above hash table, there are n slots in the table, names = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. Slot 0, slot 1, slot 2 and so on. Hash table contains no items, so every slot is empty.

Division Hash Function

- As we know the mapping between an item and the slot where item belongs in the hash table is called the hash function. The hash function takes any item in the collection and returns an integer in the range of slot names between 0 to n-1.
- Suppose we have integer items {26, 70, 18, 31, 54, 93}. One common method of determining a hash key is the division method of hashing and the formula is :
  - **Hash Key = Key Value % Number of Slots in the Table**

- Division method or reminder method
- takes an item and divides it
- by the table size and returns
- the remainder as its hash value.
- Constant amount of time O(1) is

required to compute the hash value and index of the hash table at that location.

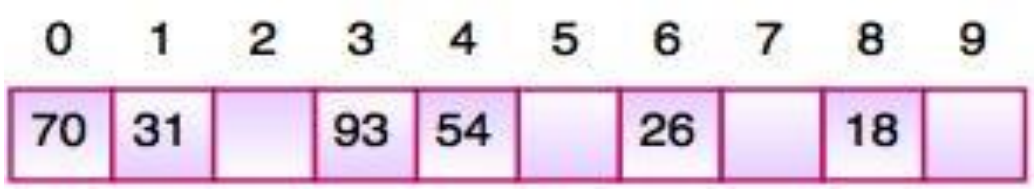
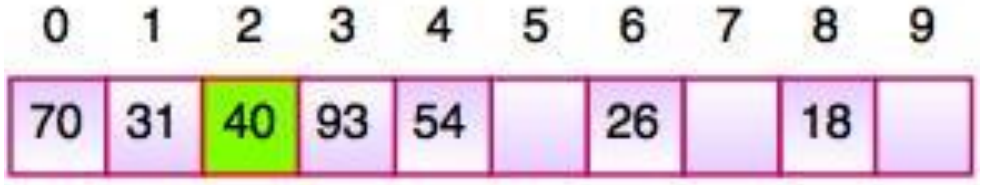


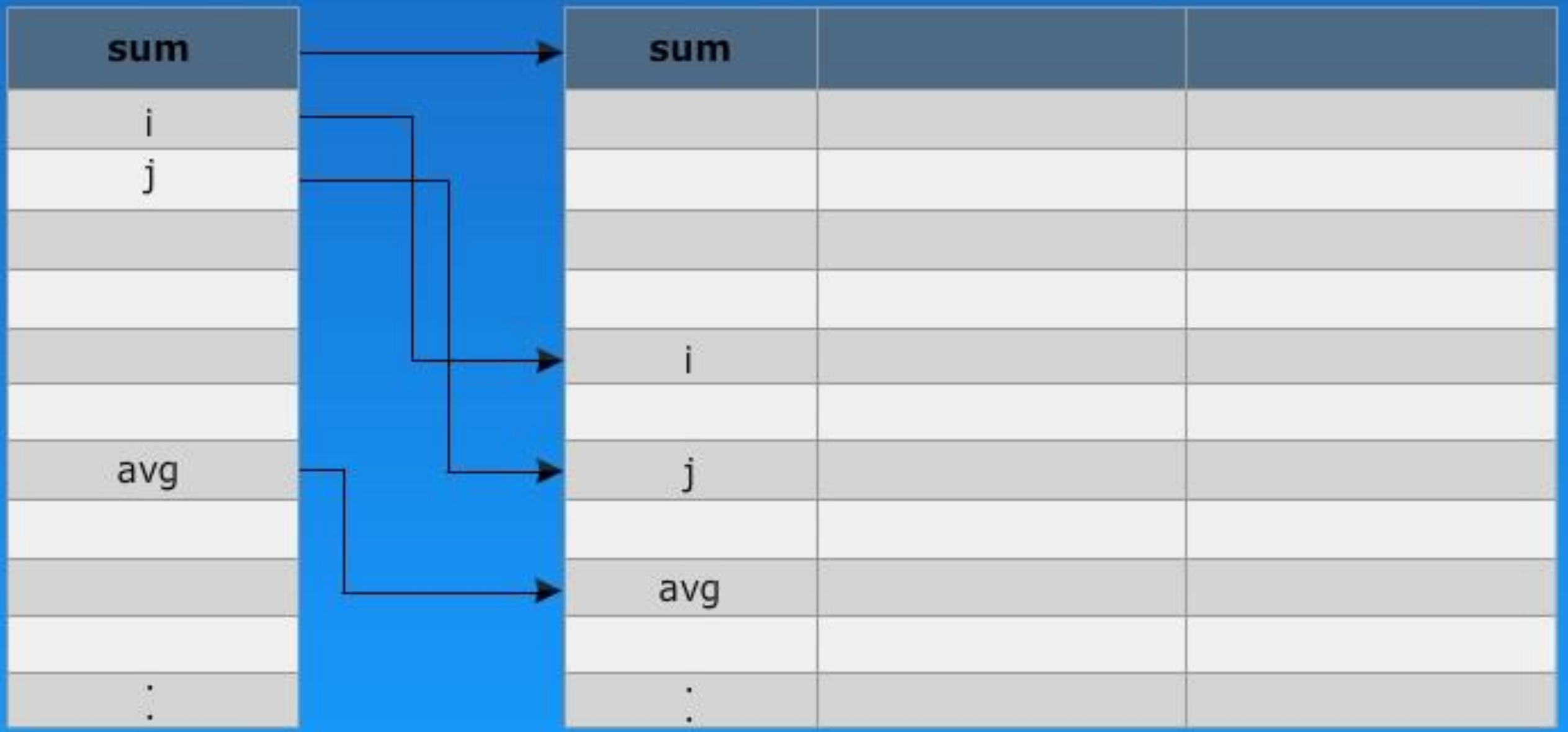
Fig. Hash Table

Data Item	Value % No. of Openings	Hash Value
26	26 % 10 = 6	6
70	70 % 10 = 0	0
18	18 % 10 = 8	8
31	31 % 10 = 1	1
54	54 % 10 = 4	4
93	93 % 10 = 3	3

- Take the above example, if we insert next item 40 in our collection, it would have a hash value of 0 (40 % 10 = 0). But 70 also had a hash value of 0, it becomes a problem. This problem is called as **Collision** or **Clash**. Collision creates a problem for hashing technique.
- **Linear probing is used for resolving the collisions in hash table**, data structures for maintaining a collection of key-value pairs.
- The simplest method is called Linear Probing. Formula to compute linear probing is:
  - **P = (1 + P) % (MOD) Table_size**
- **For example,**
- If we insert next item 40 in our collection, it would have a hash value of 0 (40 % 10 = 0). But 70 also had a hash value of 0, it becomes a problem.
- P = H(40)=44 % 10 = **0**
  Position 0 is occupied by 70.
- so we look elsewhere for a position to store 40.

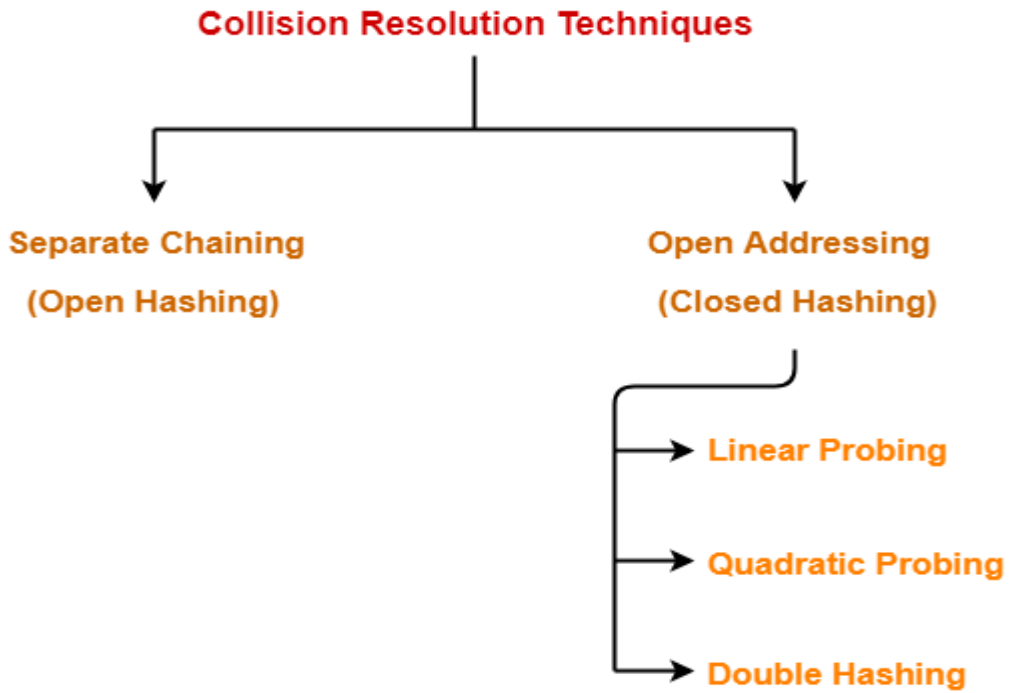


- Using Linear Probing:
  P= (P + 1) % table-size
  0 + 1 % 10 = **1**
  But, position 1 is occupied by 31, so we look elsewhere for a position to store 40.


Name
Information
Hash Link
sum
i
j
avg
sum
i
j
avg
Hash Table
Symbol Table

HASH VALUE	NAME	ATTRIBUTE
0	sort	
1		
2	size	
.	j	
.	a	
TableSize - 1		

- Hash function is used to compute the hash value for a key.
- Hash value is then used as an index to store the key in the hash table.
- Hash function may return the same hash value for two or more keys.
- When the hash value of a key maps to an already occupied bucket of the hash table, it is called as a Collision.
- Collision Resolution Techniques are the techniques used for resolving or handling the collision.



Separate Chaining-

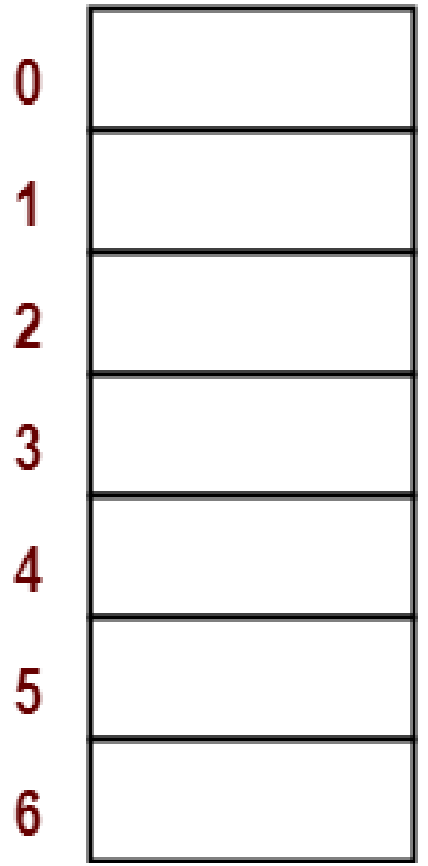
- To handle the collision,

- This technique creates a linked list to the slot for which collision occurs.
- The new key is then inserted in the linked list.
- These linked lists to the slots appear like chains.
- That is why, this technique is called as **separate chaining**.
- **Time Complexity-**
- **For Searching-**
- In worst case, all the keys might map to the same bucket of the hash table.
- In such a case, all the keys will be present in a single linked list.
- Sequential search will have to be performed on the linked list to perform the search.
- So, time taken for searching in worst case is O(n).
- **For Deletion-**
- In worst case, the key might have to be searched first and then deleted.
- In worst case, time taken for searching is O(n).
- So, time taken for deletion in worst case is O(n).

Example

- Using the hash function ‘key mod 7’, insert the following sequence of keys in the hash table-
  - 50, 700, 76, 85, 92, 73 and 101
- Use separate chaining technique for collision resolution.
- The given sequence of keys will be inserted in the hash table as-
- **Step-01:**

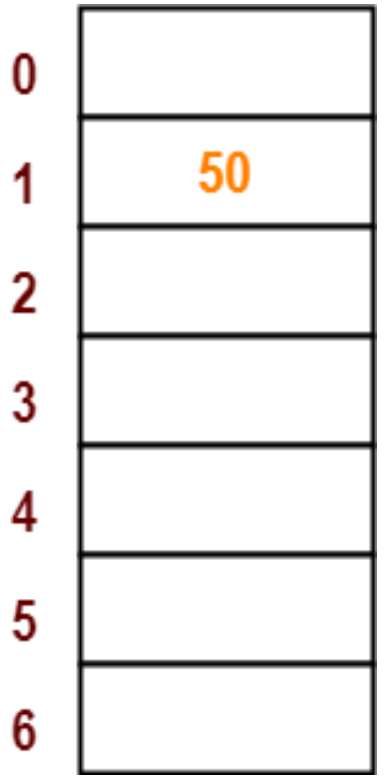
- Draw an empty hash table.
- For the given hash function, the possible range of hash values is [0, 6].
- So, draw an empty hash table consisting of 7 buckets as-



Example

- Using the hash function ‘key mod 7’, insert the following sequence of keys in the hash table-
  - 50, 700, 76, 85, 92, 73 and 101
- Use separate chaining technique for collision resolution.
- The given sequence of keys will be inserted in the hash table as-
- **Step-02:**

- Insert the given keys in the hash table one by one.
- The first key to be inserted in the hash table = 50.
- Bucket of the hash table to which key 50 maps = 50 mod 7 = 1.
- So, key 50 will be inserted in bucket-1 of the hash table as-



Example

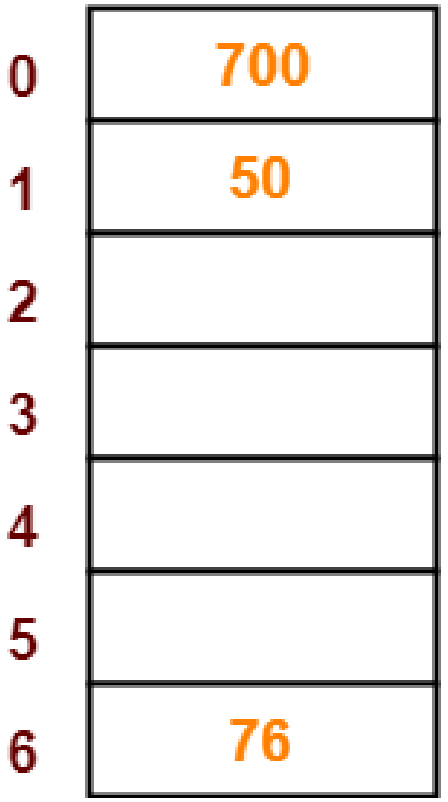
- Using the hash function ‘key mod 7’, insert the following sequence of keys in the hash table-
  - 50, 700, 76, 85, 92, 73 and 101
- Use separate chaining technique for collision resolution.
- The given sequence of keys will be inserted in the hash table as-
- **Step-03:**

- The next key to be inserted in the hash table = 700.
- Bucket of the hash table to which key 700 maps = 700 mod 7 = 0.
- So, key 700 will be inserted in bucket-0 of the hash table as-

Example

- Using the hash function ‘key mod 7’, insert the following sequence of keys in the hash table-
  - 50, 700, 76, 85, 92, 73 and 101
- Use separate chaining technique for collision resolution.
- The given sequence of keys will be inserted in the hash table as-
- **Step-04:**

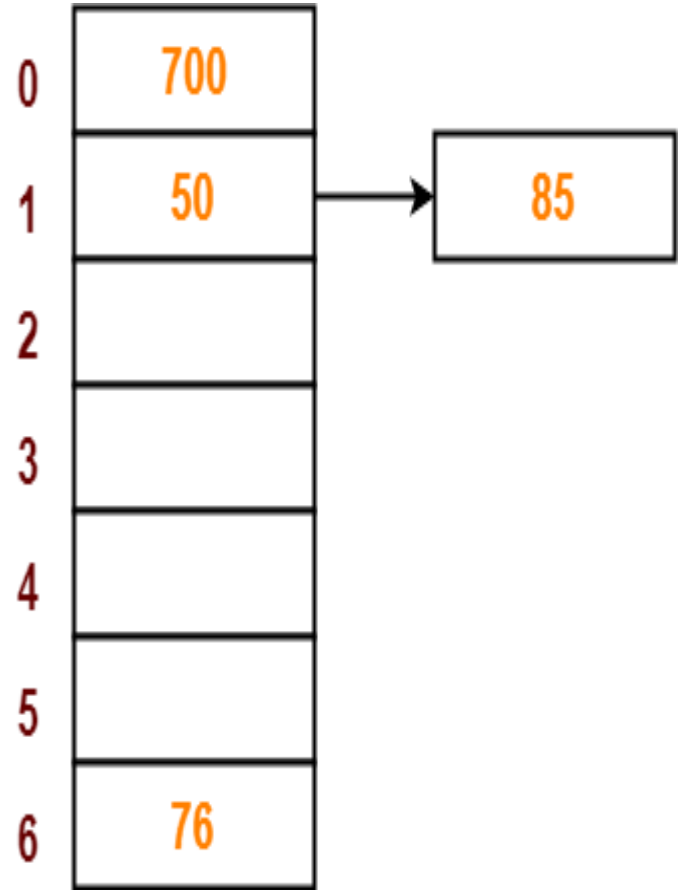
- The next key to be inserted in the hash table = 76.
- Bucket of the hash table to which key 76 maps = 76 mod 7 = 6.
- So, key 76 will be inserted in bucket-6 of the hash table as-



Example

- Using the hash function ‘key mod 7’, insert the following sequence of keys in the hash table-
  - 50, 700, 76, 85, 92, 73 and 101
- Use separate chaining technique for collision resolution.
- The given sequence of keys will be inserted in the hash table as-
- **Step-05:**

- The next key to be inserted in the hash table = 85.
- Bucket of the hash table to which key 85 maps = 85 mod 7 = 1.
- Since bucket-1 is already occupied, so collision occurs.
- Separate chaining handles the collision by creating a linked list to bucket-1.
- So, key 85 will be inserted in bucket-1 of the



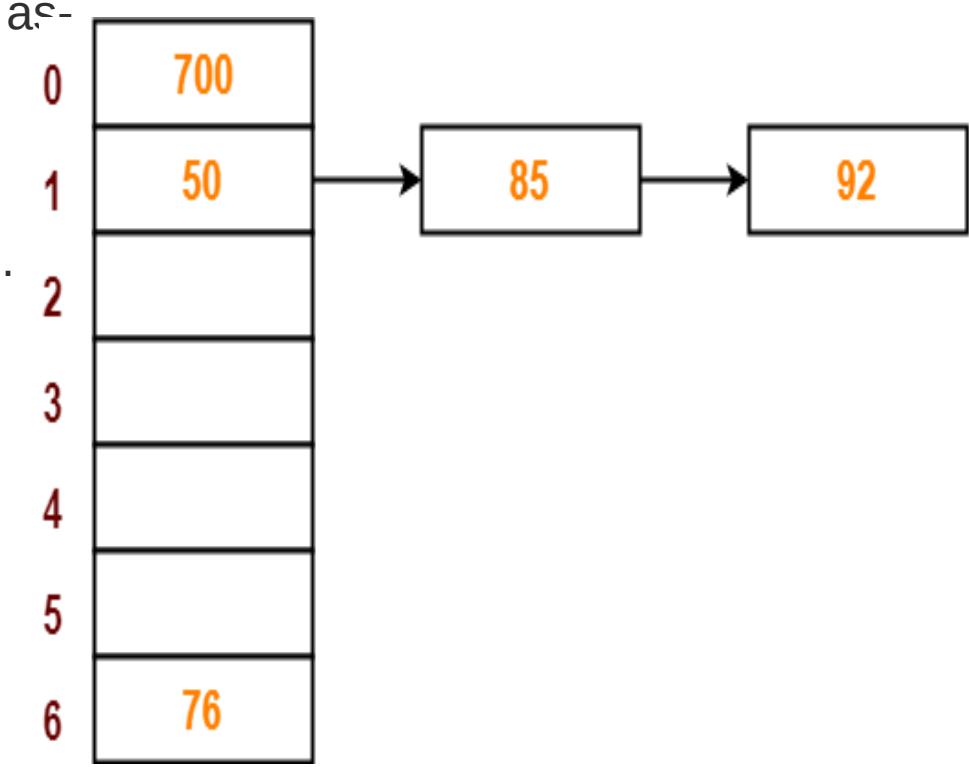
Example

- Using the hash function ‘key mod 7’, insert the following sequence of keys in the hash table-
  - 50, 700, 76, 85, 92, 73 and 101
- Use separate chaining technique for collision resolution.
- The given sequence of keys will be inserted in the hash table as-
- **Step-06:**

- The next key to be inserted in the hash table = 92.
- Bucket of the hash table to which key 92 maps = 92 mod 7 = 1.
- Since bucket-1 is already occupied, so collision occurs.
- Separate chaining handles the collision by creating

a linked list to bucket-1.

- So, key 92 will be inserted in bucket-1 of the hash table as-



Example

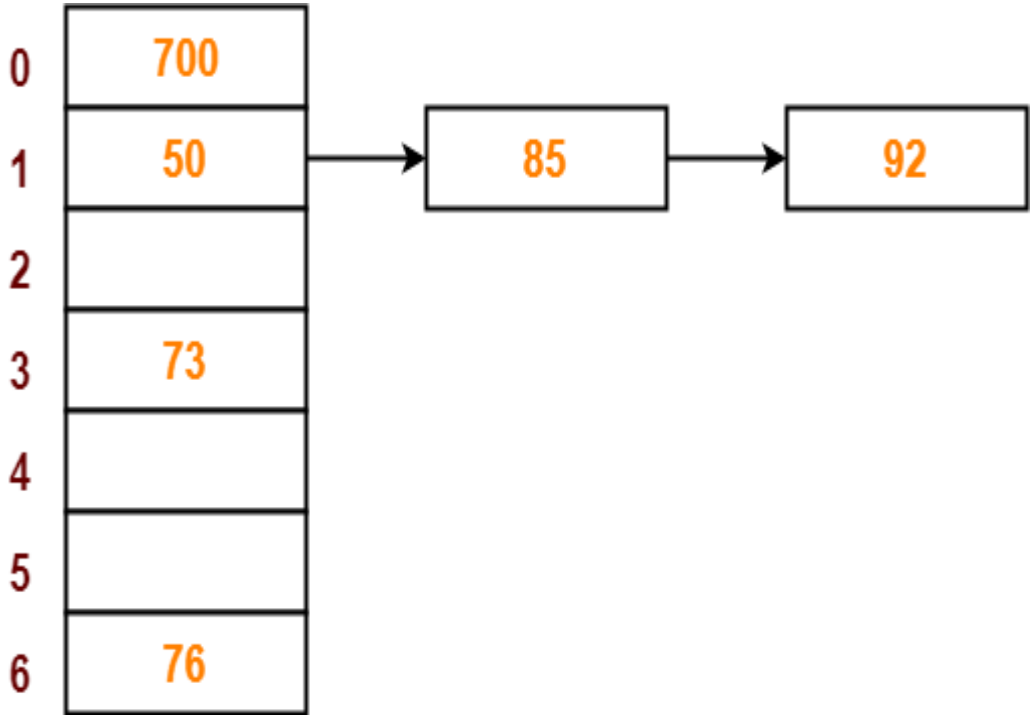
- Using the hash function ‘key mod 7’, insert the following sequence of keys in the hash table-
  - 50, 700, 76, 85, 92, 73 and 101
- Use separate chaining technique for collision resolution.
- The given sequence of keys will be inserted in the hash table as-
- **Step-07:**
-

- The next key to be inserted in the hash table = 73.
- Bucket of the hash table to which

key 73 maps = 73 mod 7 = 3.

- So, key 73 will be inserted in bucket-3

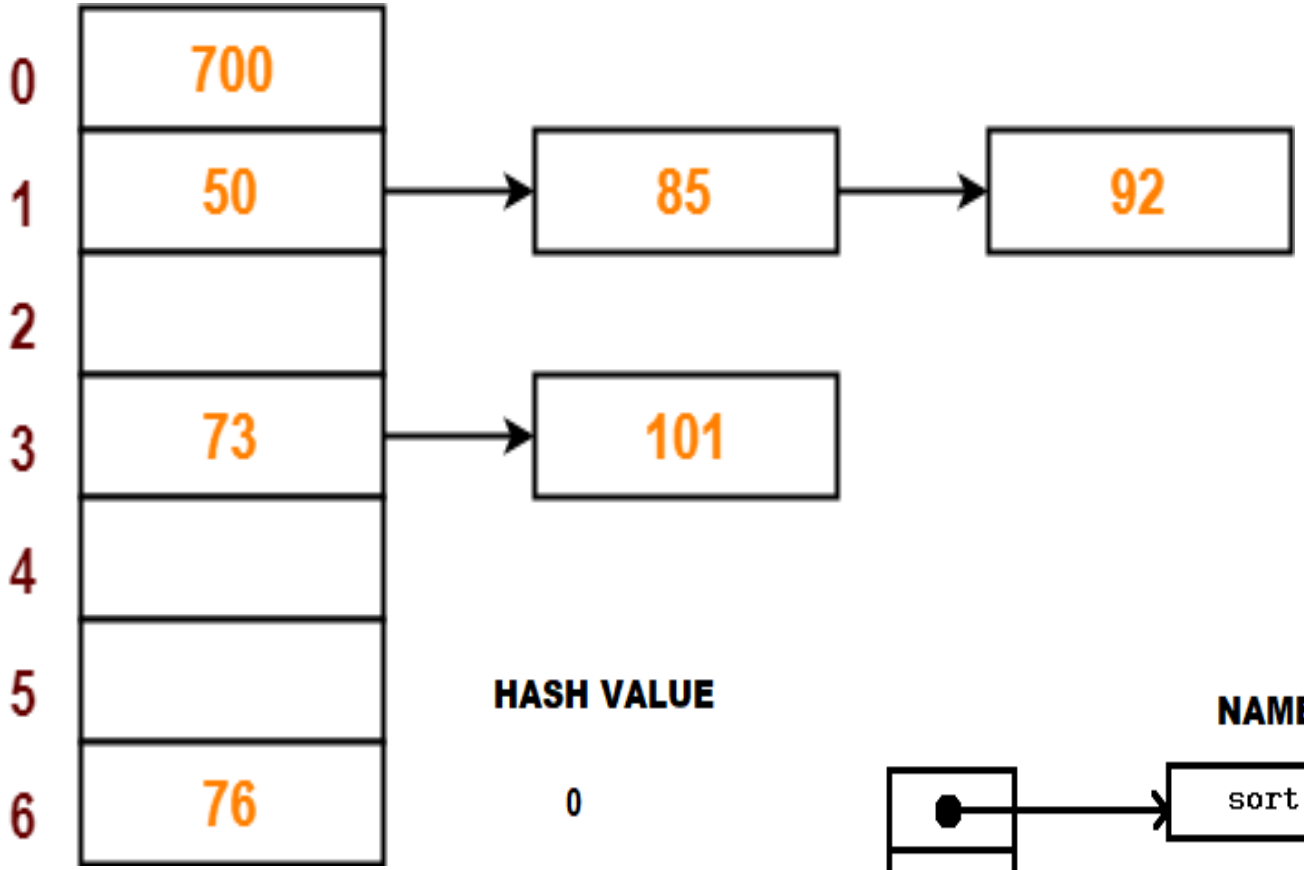
of the hash table as-

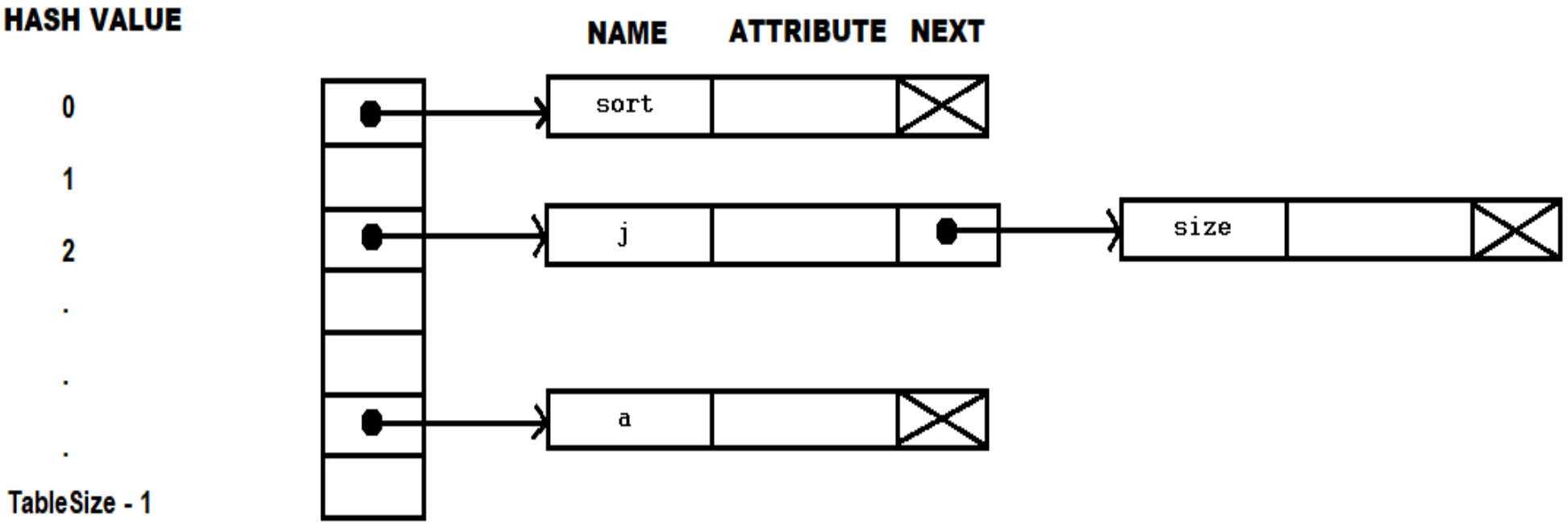


Example

- Using the hash function ‘key mod 7’, insert the following sequence of keys in the hash table-
  - 50, 700, 76, 85, 92, 73 and 101
- Use separate chaining technique for collision resolution.
- The given sequence of keys will be inserted in the hash table as-
- **Step-08:**

- The next key to be inserted in the hash table = 101.
- Bucket of the hash table to which key 101 maps = 101 mod 7 = 3.
- Since bucket-3 is already occupied, so collision occurs.
- Separate chaining handles the collision by creating a linked list to bucket-3.
- So, key 101 will be inserted in bucket-3 of the hash table as-


0
700
1
50
85
92
2
3
73
101
4
5
6
76


HASH VALUE
0
1
2
.
.
.
TableSize - 1
NAME
ATTRIBUTE
NEXT
sort
j
size
a

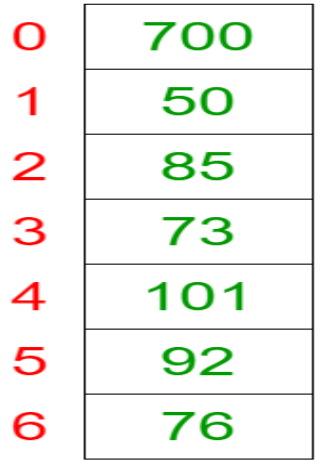
Quiz

- Using the hash function ‘key mod 7’, insert the following sequence of keys in the hash table-
  - 50, 700, 76, 85, 92, 73 and 101
- Use linear probing technique for collision resolution.

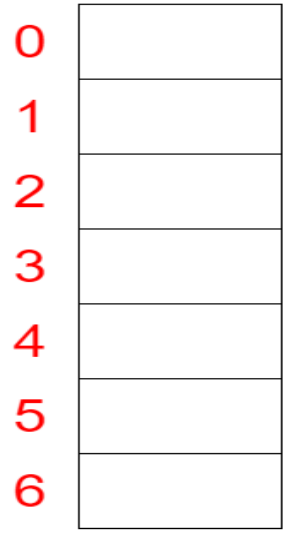
0	700
1	50
2	85
3	92
4	73
5	101
6	76

- In quadratic probing,

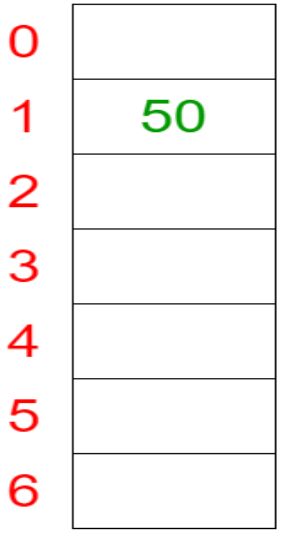
- When collision occurs,
- we probe for i^2'th bucket in i^{th} iteration.
- We keep probing until an empty bucket is found.



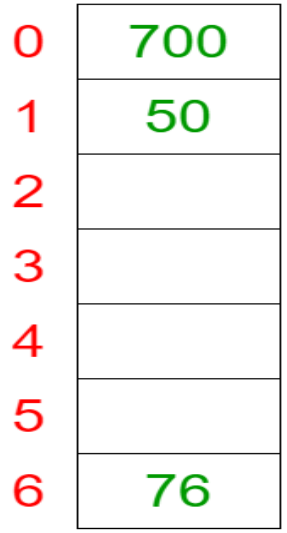
Insert 73 and 101



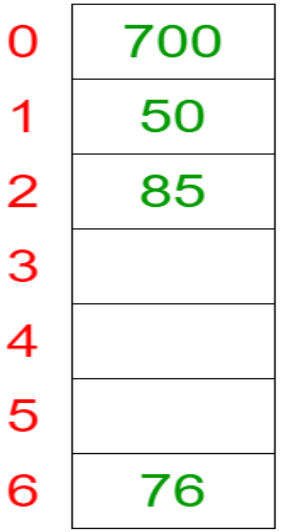
Initial Empty Table



Insert 50



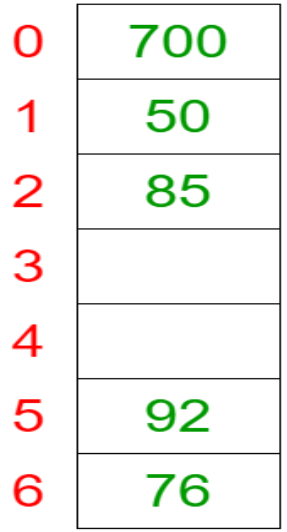
Insert 700 and 76



Insert 85:

Collision occurs.

Insert at 1 + 1*1 position



Insert 92:

Collision occurs at 1.

Collision occurs at 1 + 1*1 p

a) **[6 points]** Draw the contents of the two hash tables below after inserting the values shown. Show your work for partial credit. *If an insertion fails, please indicate which values fail and attempt to insert any remaining values*. The hash function used is H(k) = k mod tablesize.

Table 1: **Separate chaining**, (where each bucket points to an unsorted linked list)

Insert: 4, 21, 6, 28, 14, 12

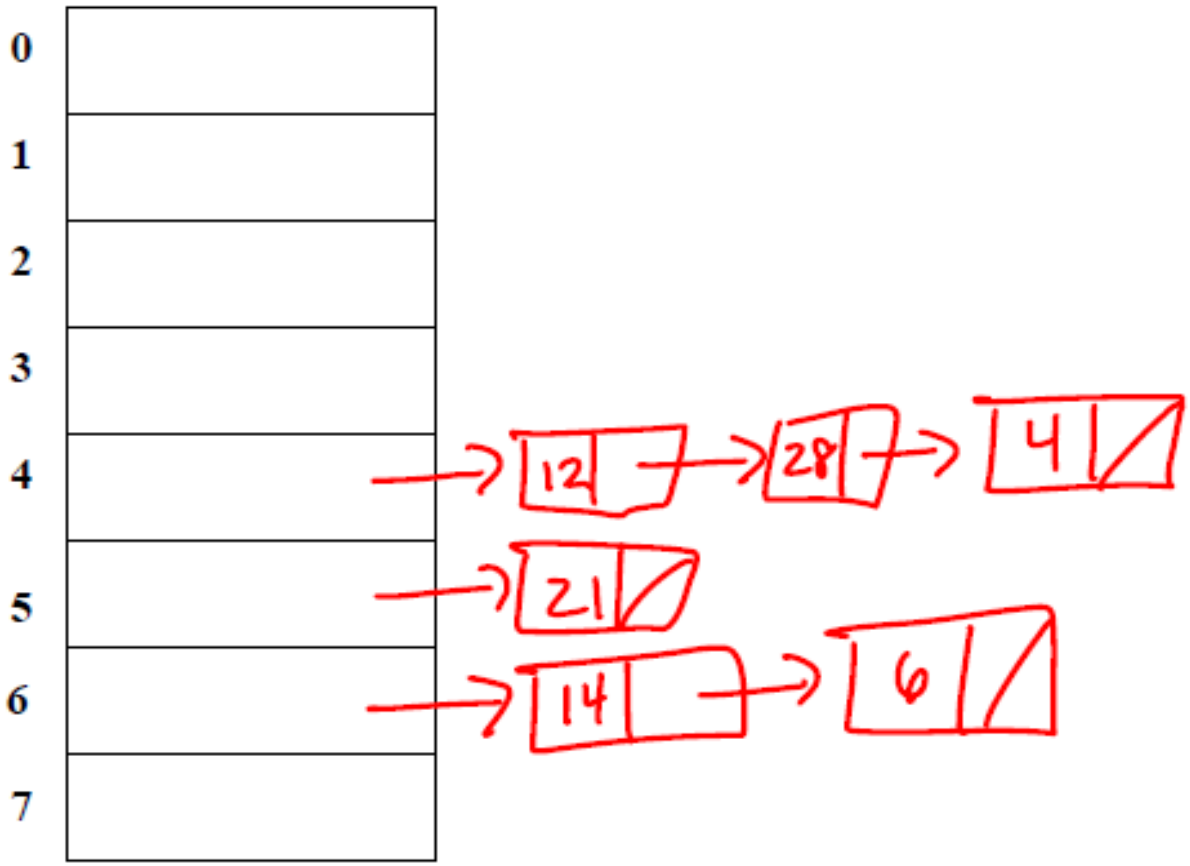
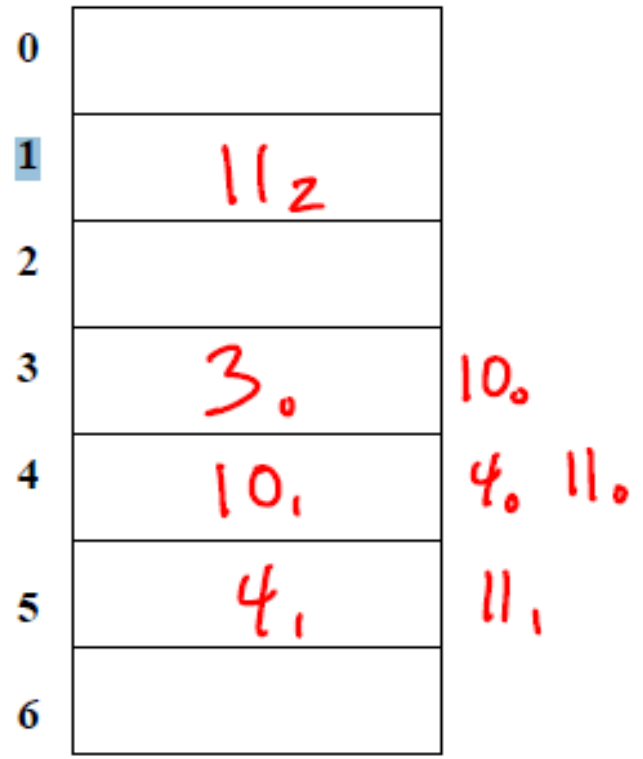


Table 2: **Quadratic Probing**

Insert: 3, 10, 4, 11

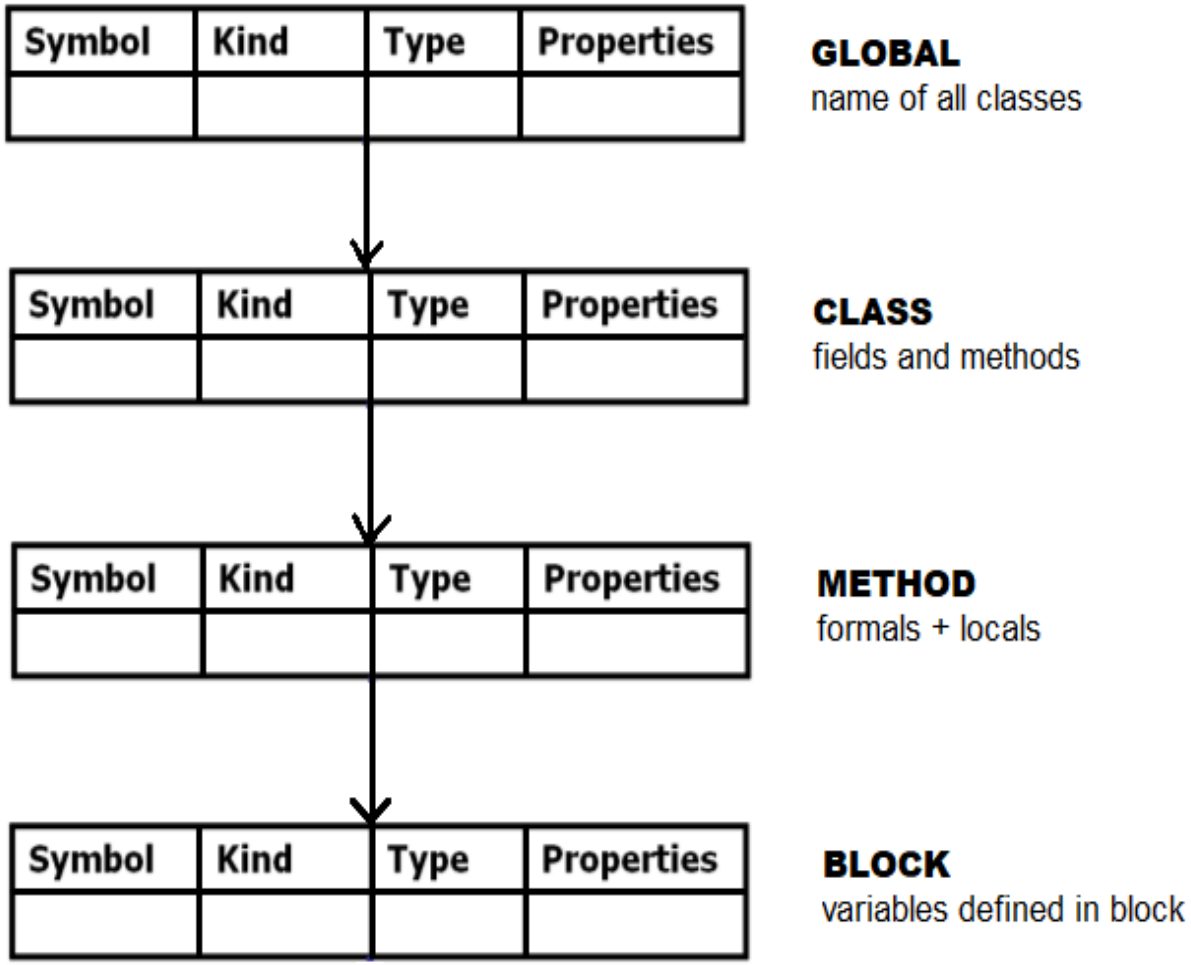


Implementing Symbol Tables

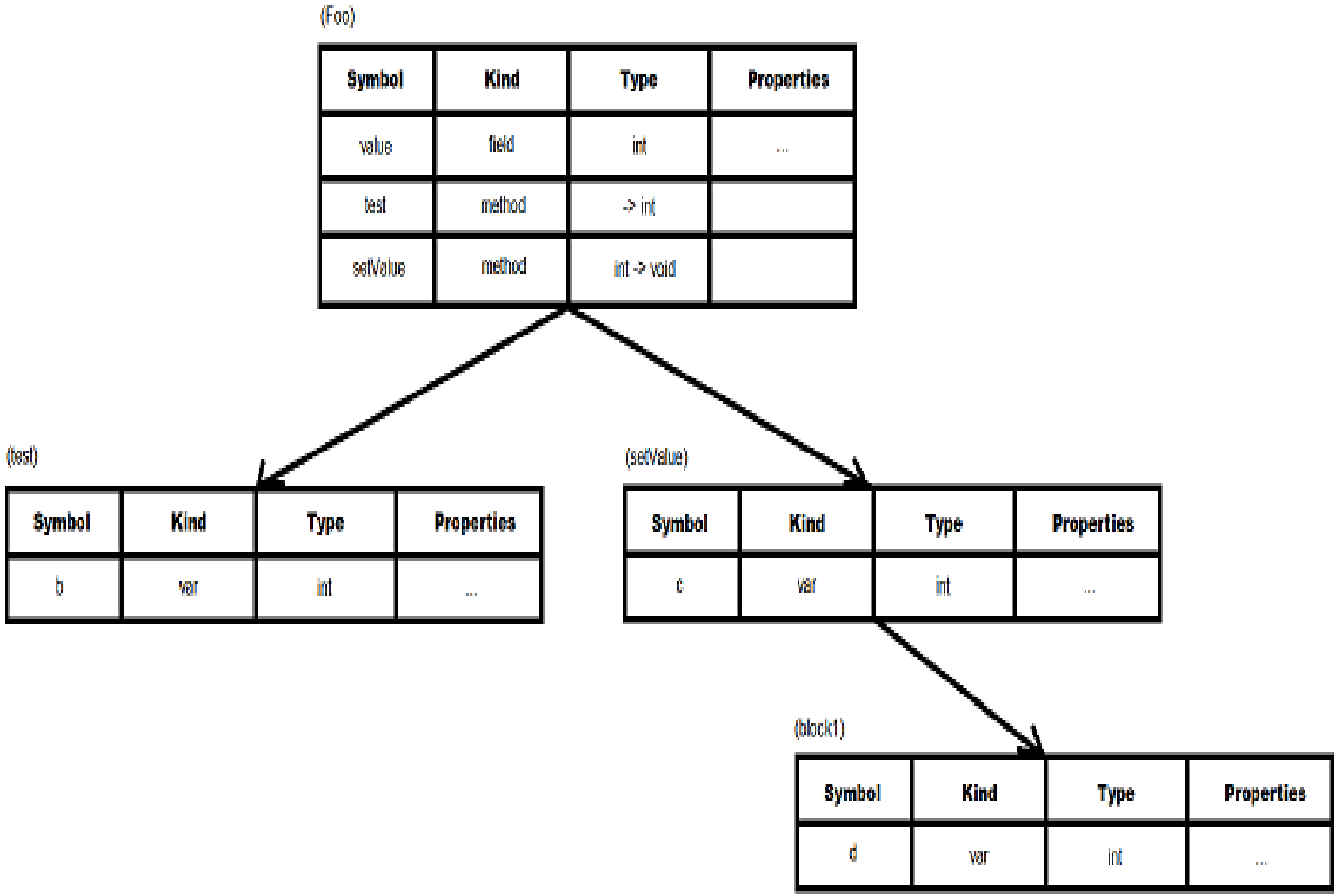
- A symbol table is a data structure that gathers scope information and keeps information about identifiers. Each item in the symbol table has the following properties: Name, Kind, Type, and others. The following is the format of the symbol table:

Symbol	Kind	Type	Properties

- The following is the scope nesting mirrored hierarchy of symbol tables:



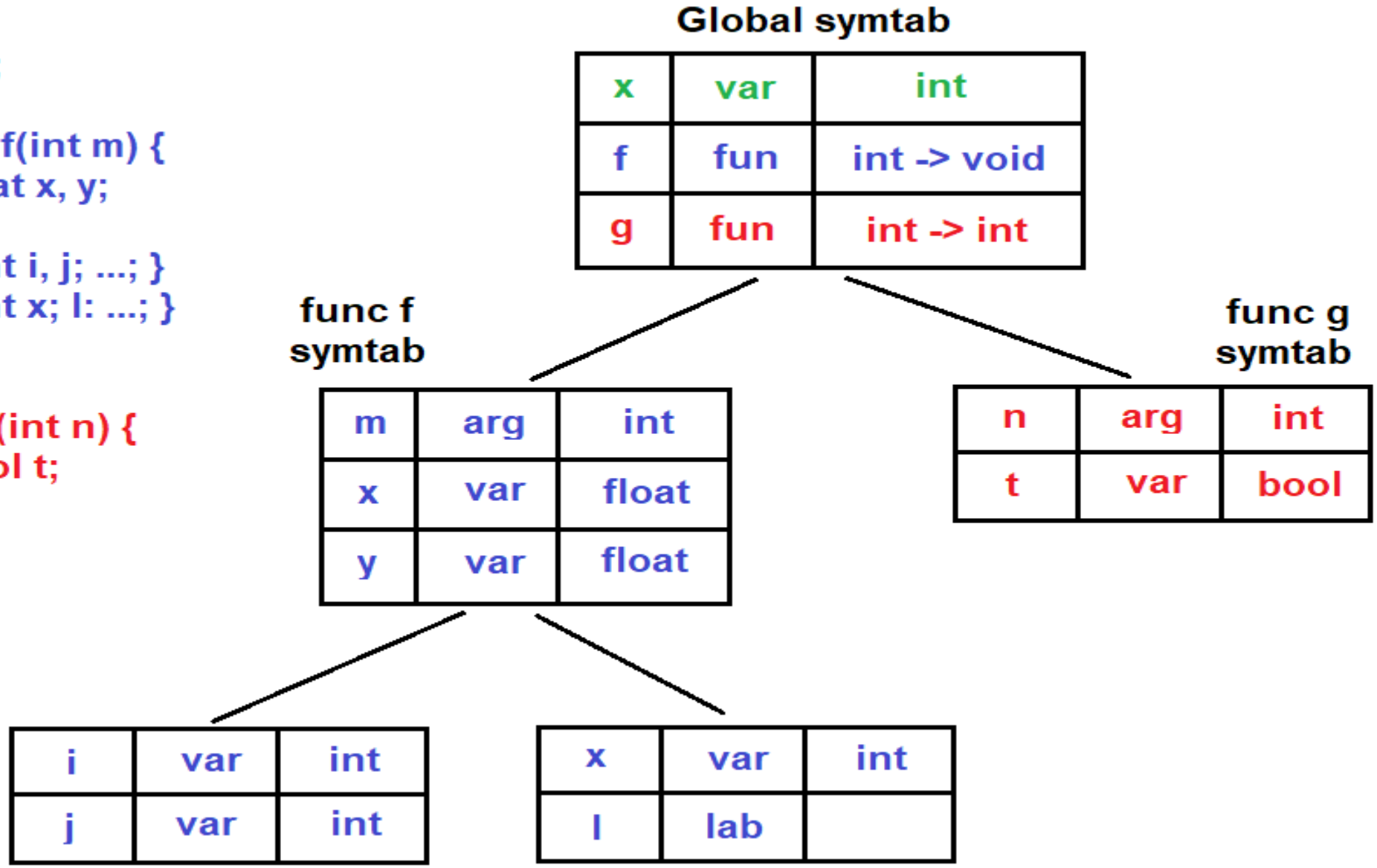
```
class Foo {
    int value;
    int test() {
        int b = 3;
        return value + b;
    }
    void setValue(int c) {
        value = c;
        {   int d = c;
            c = c + d;
            value = c;
        }
    }
}
class Bar {
    int value;
    void setValue(int c) {
        value = c;
    }
}
```



```
int x;

void f(int m) {
  float x, y;
  ...
  { int i, j; ...; }
  { int x; l: ...; }
}

int g(int n) {
  bool t;
  ...;
}
```



Thanks