GitHub: https://github.com/neon1024/Formal-Languages-and-Compiler-Design/tree/main/Labs/Lab%202

There are two main classes: HashTable and SymbolTable. The HashTable class implements a hash table with open addressing and double hashing for collision resolution. The SymbolTable class provides a higher-level abstraction for managing symbols using the HashTable.

HashTable:
The HashTable class is a custom implementation of a hash table that supports dynamic resizing and uses double hashing for collision resolution.

Attributes:
capacity: The total number of slots in the hash table.
size: The number of elements currently in the hash table.
table: A list that holds the key-value pairs, initialized to None for each slot.
load_factor_threshold: The threshold that triggers a resize when the table exceeds this load factor. Default is set to 0.7.

Methods:
__init__(self, capacity=11)
Initializes a hash table with a specified capacity (default is 11) and an empty table.

h2(self, key)
Hash function that computes the index based on the key. If the key is a string, it computes the ASCII sum of the characters in the string and then returns the modulus of the sum with the current table capacity.

h1(self, key, index)
Second hash function used for double hashing. It computes the next probing index when a collision occurs. The new index is (h2(key) + index) % capacity.

_resize(self)
Increases the capacity of the hash table when the load factor exceeds the threshold. It doubles the capacity and then finds the next prime number to ensure a good distribution of keys.

__setitem__(self, key, value)
Inserts or updates a key-value pair in the hash table. Uses open addressing and double hashing to resolve collisions. If the load factor exceeds the threshold, the table is resized.

__getitem__(self, key)
Retrieves the value associated with the given key. Probes the table in case of collisions using double hashing.

__delitem__(self, key)
Removes a key-value pair from the hash table. It rehashes the table after removing the element to maintain the probing sequence.

_rehash_after_removal(self, remove_index)
Rehashes items that were in the probing sequence following the removed item. Ensures that all key-value pairs remain accessible even after an item is deleted.

__len__(self)
Returns the number of elements in the hash table.

keys(self)
Returns a list of all keys currently stored in the hash table.

values(self)
Returns a list of all values currently stored in the hash table.

clear(self)
Clears the hash table by setting all slots to None and resetting the size to 0.

__repr__(self)
Returns the string representation of the hash table.

__str__(self)
Returns a human-readable string of the hash table with "Empty" for unused slots.

SymbolTable:
The SymbolTable class is a higher-level abstraction built on top of the HashTable class. It provides a way to store and manage symbols, in the context of a programming language’s symbol table.

Attributes:
__hash_table: An instance of the HashTable class used to store symbols.

__current_free_position: An internal counter to keep track of the next available position in the hash table for inserting symbols.

Methods:
__init__(self)
Initializes an empty symbol table with a new HashTable and a counter set to 0.

add(self, symbol)
Adds a new symbol to the hash table at the current free position. The position is used as the key.

symbols(self)
Returns a list of all symbols stored in the symbol table.

clear(self)
Clears the symbol table and resets the current free position to 0.

__len__(self)
Returns the number of symbols stored in the symbol table.

__repr__(self)
Returns a string representation of the symbol table.

__str__(self)
Returns a string representation of the underlying hash table in the symbol table.