

GitHub: <https://github.com/edwardinio18/LFTC/tree/main/Labs/Lab4>

For my symbol table, I chose to implement a hash table which can be used for both identifiers and constants, storing them in a key-value pair with their respective indices to be able to find them easily in the tables and match their identifiers with their correct values.

## **HashTable class**

The hash table class serves as the underlying data structure for the symbol table. It's responsible for storing and managing key-value pairs.

### **Constructor**

The constructor initializes the hash table with a specified capacity, which determines the size of the hash table. It uses an array to create the table.

### **Methods**

getCapacity(): Returns the capacity of the hash table.

hash(int \$key): This method figures out where to store an integer key's value in the hash table. It calculates a position based on the key's value and the table's capacity.

hashString(string \$key): This method determines where to store a string key's value in the hash table. It computes a position based on the string's characters and the table's capacity.

contains(int|string \$key): Checks if the hash table contains a specific key.

getHashValue(int|string \$key): This method calculates a numerical value (hash) for a given key (either an integer or a string). It's used internally to determine where to store or find key-value pairs in the hash table.

`add(int|string $key, int $value)`: This method is used to add a new keyvalue pair to the hash table.

`getPosition(int|string $key)`: This method allows you to get the position of an identifier/constant. If the key doesn't exist, it returns -1 to indicate that the key wasn't found.

`getHashTable()`: This method returns the entire hash table as an array, which includes all the key-value pairs currently stored in the hash table.

## **SymbolTable class**

The symbol table class manages symbols and their corresponding codes, catering to both integers and strings.

### **Constructor**

The constructor initializes the symbol table with a specified capacity. The symbol table uses a hash table internally.

### **Methods**

`add(int|string $symbol)`: This method adds a new symbol, which can be either an integer or a string, to the symbol table. The code associated with each symbol is incremented automatically upon addition.

`getPosition(int|string $key)`: This method is used to retrieve the position of an identifier/constant associated with a specific key. It returns the position in the symbol table for the given key. If the symbol is not found, it returns -1.

`__toString()`: This method overrides the default `__toString` method to provide a string representation of the symbol table instance. It displays the content of the symbol table in a readable format, showing the mapping of symbols to their associated codes.

## **Scanner class**

The scanner class breaks down input programs into tokens and keeps track of identifiers and constants.

## **Constructor**

The constructor initializes the scanner with symbol tables and empty lists, and loads tokens from a file.

## **Methods**

setProgram(string \$program): Set the program to be scanned.

readTokens(): Reads tokens from a file and populates the reservedWords and tokens arrays.

skipSpaces(): Skip spaces and increment the current line number when encountering newline characters.

skipComments(): Skip comments in the input program.

treatStringConstant(): Extract and process string constants from the input program.

treatIntConstant(): Extract and process integer constants from the input program.

getPosition(string \$match, string \$type = 'string'): Get the position of a constant in the symbol table and add it if it doesn't exist.

checkIfValid(string \$possibleIdentifier, string \$programSubstring): Check if an identifier is valid and not a reserved word.

treatIdentifier(): Extract and process identifiers from the input program.

treatFromTokenList(): Extract and process tokens from the reservedWords and tokens arrays.

nextToken(): Advance to the next token in the input program and add it to the Program Internal Form (PIF).

scan(string \$programFileName): Scan the input program and generate the Program Internal Form (PIF).

## **FA Class**

The FA class represents a finite automaton. It processes a configuration file to establish the states, alphabet, transitions, initial state, and output states of the automaton.

## **Constructor**

The constructor reads the finite automaton configuration from a file. It initializes the class properties and sets up the automaton's states, alphabet, transitions, initial and output states.

## **Methods**

init(): Parses the file content and initializes the finite automaton's states, alphabet, transitions, initial state, and output states.

parseList(string \$string): Parses a string representation of a list and converts it into an array.

parseTransitions(string \$string): Parses a string representation of transitions and converts it into an array of Transition objects.

printStates(): Outputs the states of the finite automaton to the console.

printAlphabet(): Outputs the alphabet of the finite automaton to the console.

printOutputStates(): Outputs the output states of the finite automaton to the console.

printInitialState(): Outputs the initial state of the finite automaton to the console.

printTransitions(): Outputs the transitions of the finite automaton to the console.

checkAccepted(string \$word): Determines if a word is accepted by the finite automaton.

getNextAccepted(string \$word): Finds the next word that is accepted by the automaton based on the current word.

## **Transition Class**

The Transition class encapsulates the concept of a transition in a finite automaton. It holds information about the source state, the destination state, and the label (input) that triggers the transition.

## **Constructor**

The constructor initializes a Transition object with specified source state, destination state, and label.

## **Methods**

getFrom(): Retrieves the source state of the transition.

getTo(): Retrieves the destination state of the transition.

getLabel(): Retrieves the label (input) associated with the transition.

setFrom(string \$from): Sets the source state of the transition.

setTo(string \$to): Sets the destination state of the transition.

setLabel(string \$label): Sets the label (input) associated with the transition.