https://github.com/TimofteRazvan/flcd-language/tree/main/LAB4

Class Scanner

private List<string> operators – contains all operator tokens as strings
private List<string> separators – contains all separator tokens as strings
private List<string> keywords – contains all reserved keyword tokens as strings
private SymbolTable table – hashtable containing symbols
private List<Pair<string, int>> pif – program information file repr. as token, pos
private string file – current file
private string line – current line String
private int crtLine – current line index
private int crtChar – current character index

//constructor file

//param: file = string representing the filename we'll work with

public Scanner(string file)

//scanning function – reads all lines of file, detects what each string separated by separators is and treats them individually, all while increasing the indexes for line upon hitting a newline and for character upon being done with a token / reaching a space – also calls write to pif

//throws: CustomException (if error met)

public void Scan()

//writes the pif and the symbol table to a file using streamwriter

//throws: CustomException

private void WriteToFile()

//gateway to calling the other checker functions to figure out what the character (string) is and if it is correct

//throws: CustomException (if error met)

private void Detect ()

//checks if the character / string is a token

private bool CheckToken()

//checks if the character / string is an integer (and checks if it is valid)

private bool CheckInt()

```
//checks if the character / string is a string (and checks if it is valid)
private bool CheckString()

//checks if the character / string is an identifier
private bool CheckIdentifier()
```

---

class Hash Table – array of linked list of pairs of K and V generics

public HashTable () - default constructor

public HashTable (int size) - a constructor where you can set the number of linked lists

public void Put (K key, V value) – puts for key a value

public V Get (K key) – returns value at key

public bool Contains (Object value) — checks if the value exists in the table

public List<K> GetKeys() - returns all the keys from the table

public String ToString() - it returns the hash table as a string

---

class Symbol Table – contains 1 hash table of both ids and constants

public SymbolTable() - constructor

public void Put (Object value) – checks if value exists, adds

public String toString() - it returns the symbol table as a string

---

// custom exception class for throwing errors

```
public class CustomException : Exception

// constructor for the exception class
public CustomException(string message) : base(message)
```

public class Pair – custom Pair class to use since C# only has immutable Tuple and non-generic Pair public class Pair(K, V) – constructor

public class Key() – getter for Key

public class Value() – getter for Value

public class App – main program class

public static void main() – main method

// finite automaton class

```
public class FiniteAutomaton

  private List<string> states; - states of the FA
  private List<string> alphabet; - alphabet of the FA
  private Dictionary<Pair<string, string>, List<string>> transitions; - transitions defined by pairs like ((A, 1), B)
representing state A going to state B via 1
  private string initialState; - the starting state
  private List<string> finalStates; - potential final states
```

// empty constructor

```
public FiniteAutomaton()
```

// parametrized constructor

```
public FiniteAutomaton(List<string> states, List<string> alphabet, Dictionary<Pair<string, string>, List<string>>
transitions, string initialState, List<string> finalStates)
```

// function that reads from the FA.in file and puts strings into states/alphabet etc.

```
public void ReadFromFile(string file)
```

// function that checks if determinism is found (if state A does not always lead to state B via 1)

```
public bool CheckDeterminism()
```

// For a DFA, verifies if a sequence is accepted by the FA.

```
public bool CheckCorrectness(string sequence)
```

FA.in BNF

```
<fa> ::= States: <states> Alphabet: <alphabets> Initial State: <state> End States: <endStates> Transitions: <transitions>

<states> :== <state> | <state> <states>

<state> :== A | B | C | ... | Z

<alphabets> :== <alphabet> | <alphabet> <alphabets>

<alphabet> :== 0 | 1 | ... | 9 | a | ... | z

<endStates> :== <state> | <endStates>

<transitions> :== <transition> | <transition> <transitions>

<transition> :== <state> <alphabet> <state>
```