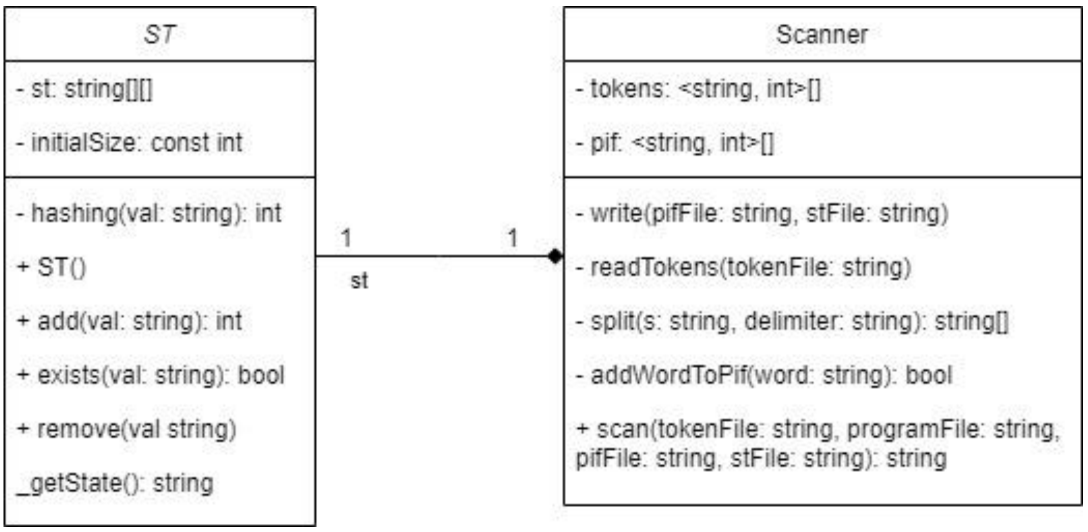
https://github.com/BogdanDumbravean/Formal-Languages-and-Compiler-Design/tree/main/Lab4%20Scanner

(I've done some changes to the ST: add and getState now return some values.)


ST
- st: string[][]
- initialSize: const int
- hashing(val: string): int
+ ST()
+ add(val: string): int
+ exists(val: string): bool
+ remove(val string)
_getState(): string
1
1
st
Scanner
- tokens: <string, int>[]
- pif: <string, int>[]
- write(pifFile: string, stFile: string)
- readTokens(tokenFile: string)
- split(s: string, delimiter: string): string[]
- addWordToPif(word: string): bool
+ scan(tokenFile: string, programFile: string, pifFile: string, stFile: string): string

```
class Scanner{
private:
```
// Writes in the given files the data from pif and the st respectively
// Input: pifFile - name of the file for the pif
//   stFile – name of the file for the symbol table
// Preconditions:
// Postconditions: the files from pifFile and stFile will be created if they don't exist, or they will be overwritten (if the method is called).
```
    void write(string pifFile, string stFile);
```

// Reads the tokens from tokenFile and puts them in a map
// Input: tokenFile - name of the file of the tokens
// Preconditions: file name must exist and tokens must be placed in a specific manner: id followed by token, first two pairs being for the identifiers and constants
// Postconditions: the tokens are inserted in a map
```
    void readTokens(string tokenFile);
```

// Splits a string in a vector of strings by using a delimiter
// Input: s - a string that needs to be split in multiple strings
//   delimiter – a string to be used as delimiter for splitting
// Output: - A vector of the strings that were before the string s, separated by the given delimiter
```
    vector<string> split(string s, string delimiter);
```

```
	// Separates a string using the table of tokens
	// Input: word - the string to be analyzed and inserted in the pif table
	// Preconditions: - tokens have already been read
			 - "word" doesn't contain whitespaces
	// Output: - true, if the given string is lexically correct
	//		 - false, otherwise
	// Postconditions:  the tokens from the given string have been added to
the pif table
	bool addWordToPif(string word);

public:
	// Scans the program given, using the tokens given and outputs the pif
and st, along with a returned value with more details about the possible errors
	// Input: tokenFile - name of the file of the tokens
	//	programFile – name of the file with the source code
	//	pifFile - name of the file for the pif
	//	stFile – name of the file for the symbol table
	// Preconditions: tokenFile and programFile exist and the tokens are
correctly written (as stated in the "readTokens" method preconditons)
	// Output: - a string stating that the program is lexically correct, or that it
has an error, with the line and group that have given the said error
	// Postconditions: the private variables have been changed
	//			pifFile and stFile have been created or modified
	string scan(string tokenFile, string programFile,
string pifFile, string stFile);
};


class ST {
private:
	// Performs a hashing on the given value
	// Input: val - a string that denotes the element on which to perform the
hashing
	// Output: - the value produced by hashing
	int hashing(string val);

public:
	// Default constructor. Initializes the size of the table
	// Postconditions: the hash table now has a set dimension
	ST();
	// Adds the given element to a position in the table
	// Input: val - a string that denotes the element to be added to the table
	// Output: – the position of the element in the hash table
	// Postcondition: if val was already in the table, it is not added a second
time
	int add(string val);

	// Checks if the given element already exists in the table
	// Input: val - a string that denotes the value to be verified if is in the
table
	// Output: - true if the element already exists
	//		- false otherwise
	bool exists(string val);

	// Removes the element with the given value from the table (if it exists)
	// Input: val - a string that denotes the element to be removed from the
table
	void remove(string val);

	// Prints to console the current inner state of the table
	// Output: - a string with all the elements of the table, each key on one
line
	string _getState();
};
```