Documentation

https://github.com/927TriponBriana/University-Projects/tree/main/Semester%205/FLCD

**SymbolTable class**

For my SymbolTable I chose to implement 2 separate HashTables, one for identifiers and one for constants. My HashTable is generic, representing the data type of the object that will be going to be inserted into the table. The variable “table” is an array of lists, that serves as the underlying data structure to store values. Each element of the array is a “bucket” that can hold multiple values.

hashFunction(Value value) – I computed the sum of the ASCII codes of the characters and made modulo the size of the HashTable;

add(Value value) – adds an element to the SymbolTable if the position computed by the hashFunction() is not occupied, otherwise it will compute a new position until it finds one that is not occupied;

contains(Value value) – checks if a given element exists in the SymbolTable;

getHashTable() – returns a list of the values from the SymbolTable;

getPosition(Value value) – return the position of a specific element from the SymbolTable;

getBucketPosition(Value value) – return the index of the bucket where the given element would be stored;

**MyScanner class**

The MyScanner class is responsible for creating the PIF and the ST, while checking if the input program is lexically correct or not. The PIF is a list of ScannedItems, which have a token type (String) which is the actual token, String const, Int const, id and a position (int) from the ST. For the tokens the position is considered to be -1.

verifyIfStringConstant():

- "^\"[a-zA-Z0-9_ .!]*\"" => excepts a double quote at the beginning, followed by a sequence of zero or unlimited combination of characters that can include lowercase letters, uppercase letters, digits, underscore, space, dot and ends with another double quote;
- is created a matcher using the given regex patters that is used to search for matches in the “pogram” string starting from the current index;
- checks if the regular expression pattern is found in the input string, if s match is found, puts in stringConst the entire matched string constant;
- attempts to add the string constant to a symbol table, if it's successful, it retrieves the position where the string constant is stored in the symbol table and adds an entry to the PIF, with the token type "String const" and the obtained position;
- if adding to the symbol table throws an exception, it means that the string constant already exists in the symbol table so it catches the exception and retrieves the position of the

existing string constant, then adds an entry to the PIF with the token type "String const" and the obtained position;

verifyIfIdentifier():

- "^([a-z][a-zA-Z0-9_]*)" => matches a sequence of characters that start with a lowercase letter and can be followed by zero or unlimited combination of lowercase or uppercase letters, digits, underscores.
- is created a matcher using the given regex patters that is used to search for matches in the “pogram” string starting from the current index;
- checks if the regular expression pattern is found in the input string, if a match is found, puts in identifier the entire matched string;
- checks if the first character of the identifier is not a digit using Character.isDigit(identifier.charAt(0)), if the first character is a digit, it returns false because identifiers cannot start with digits;
- attempts to add the identifier to a symbol table, if it's successful, it retrieves the position where the identifier is stored in the symbol table;
- if adding to the symbol table throws an exception, it means that the identifier already exists in the symbol table so it catches the exception and retrieves the position of the existing string constant;
- then adds an entry to the PIF with the token type "id" and the obtained position;

checkIfValid(String identifier, String substring):

- checks if the identifier is a reserved word;
- "^(?![0-9])[a-zA-Z_][a-zA-Z0-9_]*.*" => it should start with a letter (a-z or A-Z) or an underscore, The remaining characters can be letters, digits, or underscores, the (?![0-9]) part is a negative lookahead assertion that ensures the identifier doesn't start with a digit;
- if the identifier is not a reserved word and doesn't match the pattern, the code proceeds to check if the identifier already exists in the symbol table;

verifyIfIntConst():

- "^(0|[+-]?[1-9][0-9]*)(?![a-zA-Z_])" => matches 0 or a negative/positive integer number with a single digit from 1 to 9 followed by zero or unlimited digits from 0 to 9; (?![a-zA-Z_]) this is a negative lookahead assertion that ensures there are no letters (a-z or A-Z) or underscores (_) immediately following the matched integer constant;
- is created a matcher that searches for this pattern in the remaining program text starting from the current index;
- checks if the regular expression pattern is found in the input, if a match is found, puts in intConst the entire matched integer constant;
- attempts to add the integer constant to a symbol table, if it's successful, it retrieves the position where the integer constant is stored in the symbol table and adds an entry to the PIF, with the token type "Int const" and the obtained position;
- if adding to the symbol table throws an exception, it means that the integer constant already exists in the symbol table so it catches the exception and retrieves the position of the

existing integer constant, then adds an entry to the PIF with the token type "Integer const" and the obtained position;

verifyTokenList():

- extracts the next token from the remaining program text, which is separated by spaces;
- checks for multi-character operators first, if the word starts with an operator, it verifies that the remaining substring does not start with the same operator, which prevents matching individual characters, i f a match is found, the index is updated to skip the matched operator, and the operator is added to the Program Internal Form (PIF) with a position of -1;
- checks for reserved words, if the word starts with a reserved word, it defines a regex pattern to check for valid identifiers following reserved words, if the regex pattern matches, it returns false to avoid matching; it then updates the index to skip the matched reserved word and adds the reserved word to the PIF with a position of -1;
- checks for separators, if the word starts with a separator, it updates the index to skip the matched separator and adds the separator to the PIF with a position of -1;

nextToken():

- iterates through the input program, attempting to identify and process different token types while skipping spaces and separators, if a token doesn't match any expected patterns or if the end of the program is reached unexpectedly, it throws a ScannerException to indicate a lexical error.

readTokens():

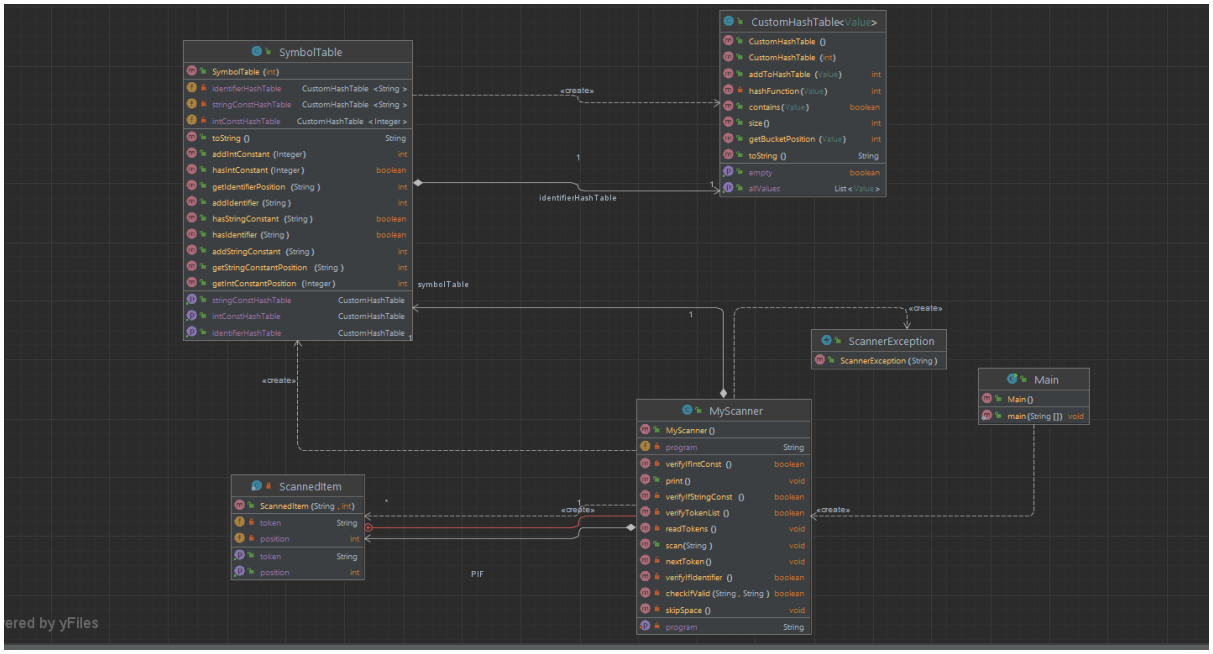
- reads the tokens from the file and categorizes them into reserved words, separators and operators

skipSpaces():

- skips over any leading white spaces(spaces, tabs, and newlines) in the program, updates the current line number when a newline character is encountered, and stops when it reaches the first non-whitespace character;

scan():

- reads the program from the given file and checks for the next token until the end of the program, then writes the PIF and the ST into the output files and prints “Lexically correct” if no issues were found or if an exception is caught, it displays the message of that exception.


SymbolTable
SymbolTable (int)
identifierHashTable CustomHashTable <String>
stringConstHashTable CustomHashTable <String>
intConstHashTable CustomHashTable <Integer>
toString () String
addIntConstant (Integer) int
hasIntConstant (Integer) boolean
getIdentifierPosition (String) int
addIdentifier (String) int
hasStringConstant (String) boolean
hasIdentifier (String) boolean
addStringConstant (String) int
getStringConstantPosition (String) int
getIntConstantPosition (Integer) int
stringConstHashTable CustomHashTable
intConstHashTable CustomHashTable
identifierHashTable CustomHashTable
«create»
1
identifierHashTable
CustomHashTable<Value>
CustomHashTable ()
CustomHashTable (int)
addToHashTable (Value) int
hashFunction (Value) int
contains (Value) boolean
size () int
getBucketPosition (Value) int
toString () String
empty boolean
allValues List<Value>
symbolTable
«create»
ScannerException
ScannerException (String)
Main
Main ()
main (String []) void
MyScanner
MyScanner ()
program String
verifyIfIntConst () boolean
print () void
verifyIfStringConst () boolean
verifyTokenList () boolean
readTokens () void
scan (String) void
nextToken () void
verifyIfIdentifier () boolean
checkIfValid (String, String) boolean
skipSpace () void
program String
ScannedItem
ScannedItem (String, int)
token String
position int
token String
position int
PIF
ered by yFiles