

Formal Languages and Compiler Design

Overview

You can find the source code of this assignment [here](#).

Representation

The chosen representation for the symbol table is a balanced binary search tree, implemented using a red black tree. A good overview of the red black tree data structure can be found [here](#).

Used Regex

- Separators Regex: `\,|;|:|\}|\\|\/|\{|\}|\\(|\\)|\\(|\\)$`
 - Generated depending on the input from tokens.in
- Operators Regex: `+|-|*|/|%|=|<|<=|==|>|>=|!=|!|and|or`
- Identifiers Regex: `^[a-zA-Z][a-zA-Z0-9]*$`
- Constants Regex:
 - Negative Integers: `^\-[1-9][0-9]*$`
 - Positive Integers: `^[1-9][0-9]*$`
 - Unsigned Integers: `^[1-9][0-9]*|0$`
 - Character: `^[a-zA-Z0-9]?$`
 - String: `^[a-zA-Z0-9]*$`

Documentation

SymbolTable

- `int retrievePosition(const std::string& element)`
 - Gets the position of the given element in the symbol table. If the element is not presented in the symbol table, then insert it first
 - In the context of the red black tree implementation of the symbol tree, the position of the element is defined as the position of the element in the sorted array

Scanner

- `Scanner(const std::string& inputFilePath, const std::string& tokensFilePath)`
- `void scan()`
 - Scans the input program, populates the PIF.out and ST.out files and prints a message to the standard output specifying if the program is lexically correct or not

- void scanTokens()
 - Scans and identifies the tokens
- std::string buildOrRegex(const std::unordered_set<std::string>& elements)
 - Builds a regex by taking each element from the given set and adding an or ("|") between each pair of elements
- void splitByOperators(std::string currentStr, int lineNumber)
 - Splits the given string by operators
- void splitByOperatorsRecursively(const std::string& multipleCharsOperatorsRegex, const std::string& singleCharOperatorsRegex, std::string currentStr, int lineNumber, bool isMultipleCharsStep = true)
 - Helper function for splitting the given string by operators
- void handleToken(std::string token, int lineNumber)
 - Handles the given token, by populating the updating the PIF and ST instances
- TokenType identifyToken(const std::string& token, int lineNumber)
 - Attempts to identify the given token
 - If the token cannot be identified, throws an exception

ProgramInternalForm

- ProgramInternalForm(const std::string& outFilePath)
- int push(const std::string& token, int position)
 - Adds a new (token, position) pair to the PIF
- ProgramInternalForm(const std::string& outFilePath)
- void getPifInFile()
 - Saves the current PIF in a file
- const std::vector<std::pair<std::string, int>> getPif()
 - Returns the PIF underlying data structure

RedBlackTree<K, Node>

- void insert(const K& key)
 - Inserts the element in the red black tree
- void remove(const K& key)
 - Removes the element from the red black tree
- Node* search(const K& key)
 - Searches the given key in the red black tree
 - If found, it returns a pointer to the corresponding node
- const K& minimum()
 - Returns the minimum in the red black tree
- const K& maximum()
 - Returns the maximum in the red black tree
- int size()
 - Returns the size of the red black tree
- Node* root()
 - Returns a pointer to the node corresponding to the root of the red black tree

Node<K>

- `const K& getKey()`
 - Returns the key corresponding to the node
- `Node* getParent()`
 - Returns the parent of the node
- `Node* getLeftChild()`
 - Returns the left child of the node
- `Node* getRightChild()`
 - Returns the right child of the node
- `bool getColor()`
 - Returns the color of the node

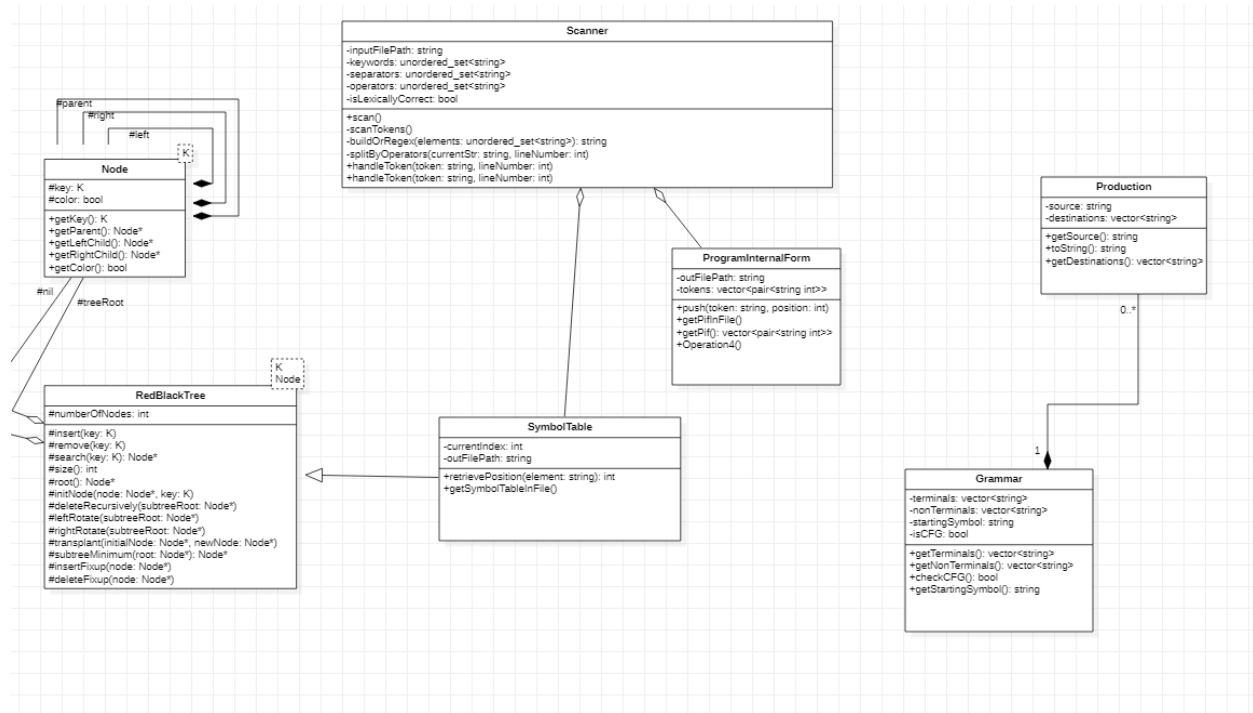
Production

- `std::string toString()`
 - return the string representation of the production
- `friend bool operator<(const Production &el, const Production &other)`
- `const std::string& getSource()`
- `const std::vector<std::string>& getDestinations()`

Grammar

- `void parse(const std::string &file_path)`
 - parse the input grammar
- `const std::vector<std::string> &getNonTerminals()`
- `const std::vector<std::string> &getTerminals()`
- `const std::vector<Production> &getProductions()`
- `const std::string &getStartingSymbol()`
- `bool checkCFG() const;`

Class Diagram



BNF for FA.in

letter ::= "a" | "b" | ... | "z"
 digit ::= "0" | "1" | ... | "9"
 character ::= digit | letter
 qualifier ::= letter character*
 transition ::= "(" qualifier "," character "->" qualifier ")"
 initialState ::= "q0 =" qualifier
 finalStates ::= "F = {" (qualifier "\n")+ "}"
 alphabet ::= "E = {" (character "\n")+ "}"
 states ::= "Q = {" (qualifier "\n")+ "}"
 transitions ::= "T = {" (transition "\n")+ "}"
 fa ::= states "\n"+ alphabet "\n"+ transitions "\n"+
 initialState "\n"+ finalStates