Chis Sergiu
Claudiu Carare
gr: 932/1

Final lab parsing

Github link: https://github.com/SergiuChis/FLCD/tree/lab5
(The branch is named ‘lab5’ because this was the first lab for parsing. It contains labs from 5 to 7)

The parsing process is done by 3 classes: Grammar, ProductionRule and LL_ParsingTable.

The Grammar class is represented as follows:

```
class Grammar
{
public:
    std::list<std::string> nonTerminals;
    std::list<std::string> terminals;
    std::list<ProductionRule> productionRules;
    std::string startSymbol;
    std::map<std::string, std::list<std::string>> firstForAllNonTerminals;

    Grammar(std::string FilePath);

private:

    void pushProductionRule(ProductionRule Pr);
    void popProductionRule(ProductionRule Pr);

    void readFromFile(std::string FilePath);
    void eliminateDuplicates(std::list<std::string>& List);
public:
    std::list<std::string> firstRecursive(std::string NonTerminal);
    std::list<ProductionRule> getProductionRulesRightSide(std::string NonTerminal);
    std::map<std::string, std::list<std::string>> getFirstForAllNonTerminals();
public:
    std::list<std::string> follow(std::string NonTerminal);


public:
    std::list<ProductionRule> getProductionRulesFor(std::string NonTerminal);
    std::list<std::string> first(std::string NonTerminal);
    bool checkCFG();

    std::string nonTerminalsToString();
    std::string terminalsToString();
    std::string productionRulesToString();

    ProductionRule getProductionRuleFor(std::string NonTerminal, std::string Terminal);
};
```

- class attributes:
  - nonTerminals (a list of all the nonterminals)
  - terminals (a list of all the terminals)
  - productionRules (a list of all the production rules of the grammar)
  - startSymbol (the start symbol of the grammar)
  - firstForAllNonTerminals (a dictionary that maps the nonterminals to all their possible starting terminals)

- class methods:
  - pushProductionRule(ProductionRule Pr)
    - adds a new production rule to the grammar
  - popProductionRule(ProductionRule Pr)
    - deletes the given production rule
  - readFromFile(string FilePath)
    - reads the grammar from the given file path
  - eliminateDuplicates(list List)
    - helper method to eliminate all duplicates from the given list
  - firstRecursive(string NonTerminal)
    - recursively search for all the terminals that the NonTerminal could start with
  - getProductionRulesRightSide(string NonTerminal)
    - get all production rules from the right side of the NonTerminal
  - getFirstForAllNonTerminals()
    - calculate first on all non terminals and return a map with the result
  - follow(string NonTerminal)
    - calculate the terminals that could appear after the given NonTerminal
  - getProductionRulesFor(string NonTerminal)
    - get all the production rules that have NonTerminal on the left side
  - first(string NonTerminal)
    - wrapper for the firstRecursive method
  - checkCFG()
    - returns true if the grammar is context free

The ProductionRule class is just a structure that represents one production rule. It is represented as folows:

```
class ProductionRule
{
public:
    std::list<std::string> leftSide;
    std::list<std::list<std::string>> rightSide;

    ProductionRule(std::list<std::string> LeftSide, std::list<std::list<std::string>> RightSide);
    ProductionRule();

    bool isEmpty();

    std::string toString();

    bool operator==(const ProductionRule& rhs) {
        std::string result1, result2;
        result1 = this->toString();
        ProductionRule temp = ProductionRule(rhs.leftSide, rhs.rightSide);
        result2 = temp.toString();
        return result1 == result2;
    };
};
```

ProductionRule contains a list of symbols for the left side, and a list of lists of symbols for the right side.
Example: A - > B C | D
leftSide would be [A]
rightSide would be [[B, C], [D]]

The LL_ParsingTable class handles the creation of the parsing table. It is represented as follows:

```
class LL_ParsingTable
{
public:
    std::list<std::string> nonTerminals;
    std::list<std::string> terminals;

    std::map<std::pair<std::string, std::string>, ProductionRule> table;

    LL_ParsingTable(Grammar Gr);

    std::string toString();

private:
    void buildTable(Grammar Gr);
};
```

The most important method of this class is buildTable(Grammar Gr), which traverses all the possible pairs of NonTerminal – Terminal, and gets the proper ProductionRule for each pair.