МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

КАФЕДРА ИиСП

ОТЧЕТ ПО ВЫПОЛНЕНИЮ   
ПРОГРАММЫ

**По дисциплине: ТАиФЯ**

Выполнили: студент 3 курса

специальности ПС

группы 31

Смирнов Фёдор Сергеевич

Подпись\_\_\_\_\_\_\_\_\_\_\_\_

Проверил: Зав. каф. ИиСП

Нехорошкова Л. Г.

Подпись\_\_\_\_\_\_\_\_\_\_\_

Дата\_\_\_\_\_\_\_\_\_\_

Йошкар-Ола

2021

## Содержание

**Содержание 2**

**СОДЕРЖАНИЕ ЗАДАНИЯ 3**

**РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ЗАДАНИЯ 5**

Задание 1 5

Задание 2 6

Задание 3 7

Задание 4 8

**ПРИЛОЖЕНИЯ 9**

ПРИЛОЖЕНИЕ 1 9

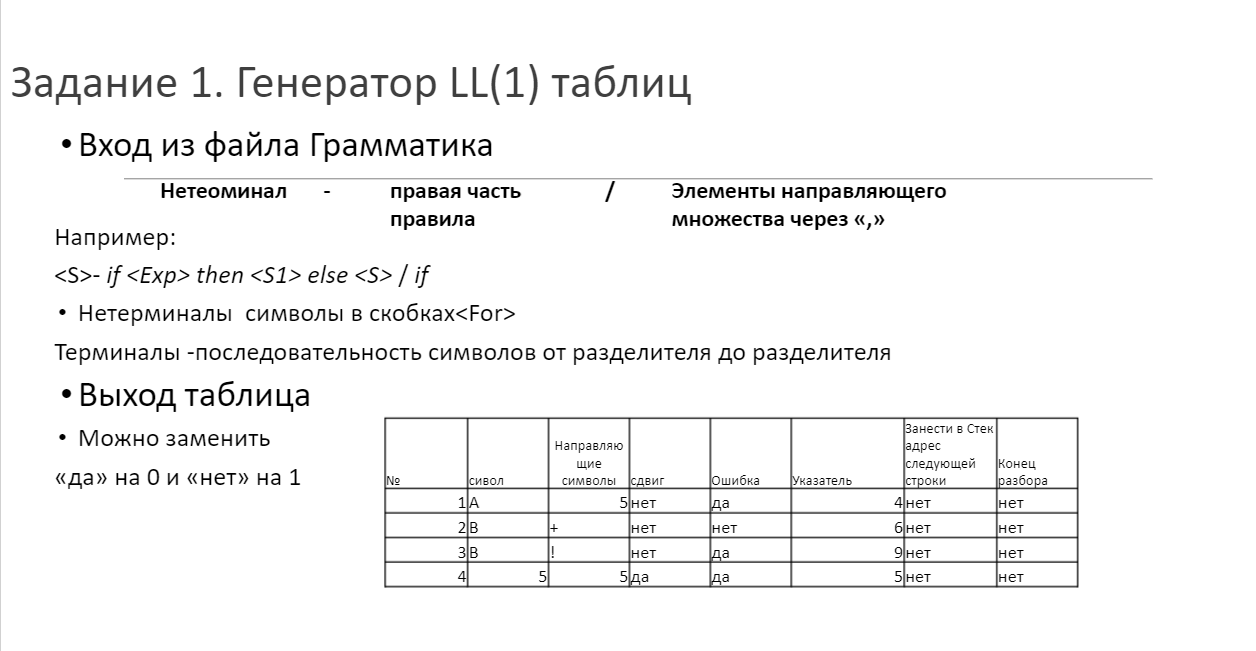
ПРИЛОЖЕНИЕ 2 11

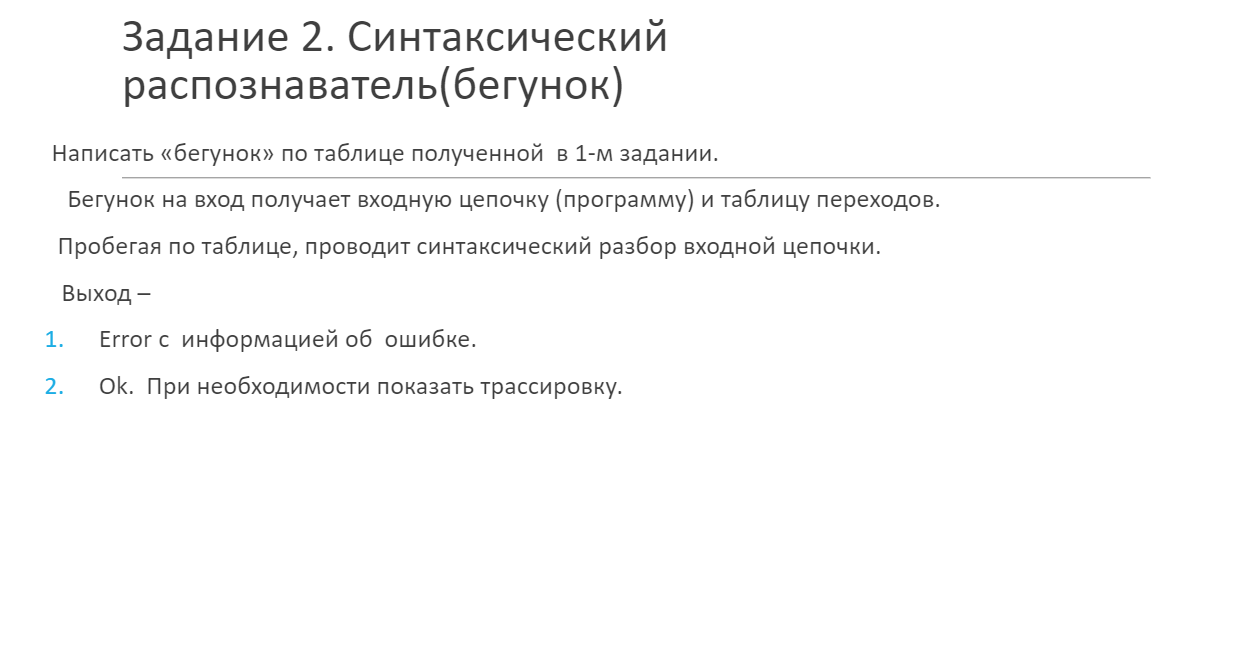
ПРИЛОЖЕНИЕ 3 13

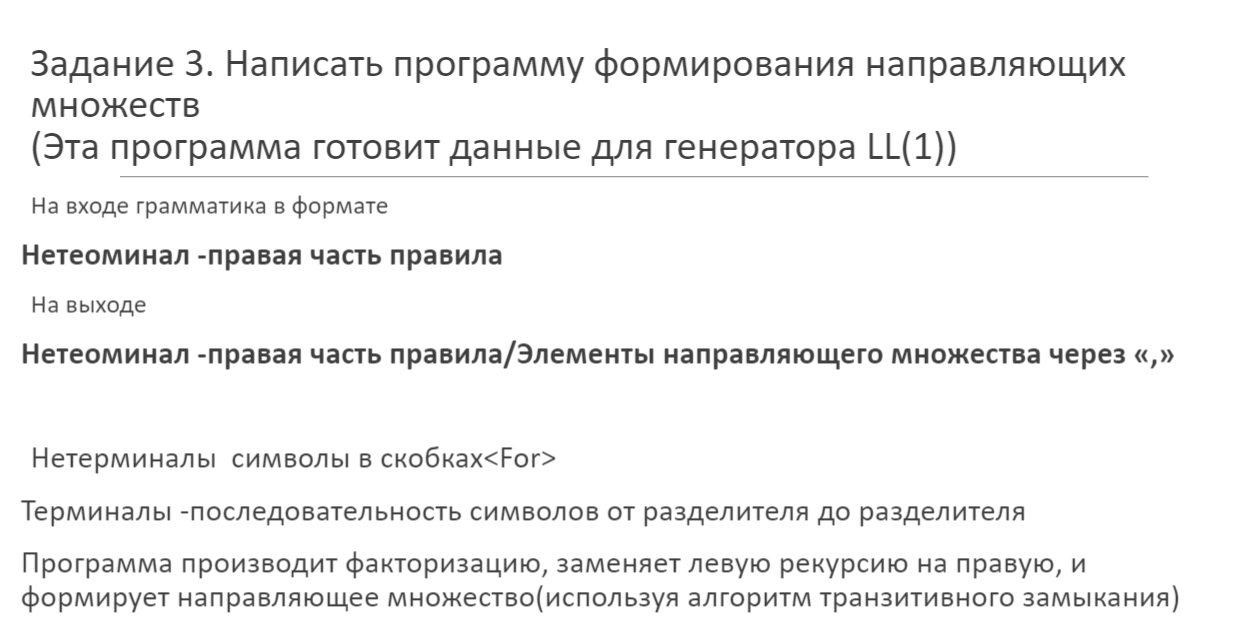
ПРИЛОЖЕНИЕ 4 16

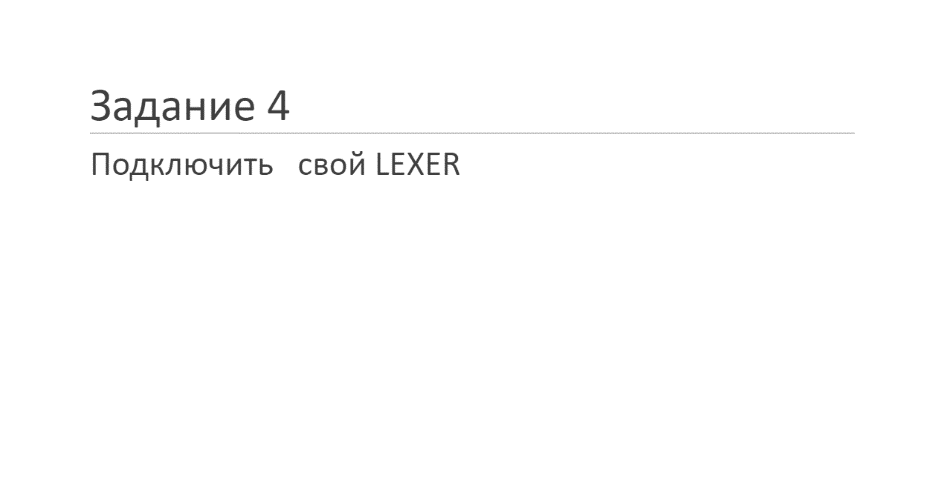
## СОДЕРЖАНИЕ ЗАДАНИЯ

Построение LL(1) генератора









РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ЗАДАНИЯ

**Задание 1**

На вход подается файл с грамматикой внутри (grammar.txt):

<Z> => <S> $

<S> => ( <S> )

<S> => e

Вызываем: generator.exe grammar.txt table.txt

В выходном файле(table.txt) получаем таблицу

| Number | Shift | Error | Pointer | Stack | End | Symbol | Characters |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 4 | 0 | 0 | Z | $ CloseParenthesis OpenParenthesis |
| 2 | 0 | 0 | 6 | 0 | 0 | S | OpenParenthesis |
| 3 | 0 | 1 | 9 | 0 | 0 | S | $ CloseParenthesis |
| 4 | 0 | 1 | 2 | 1 | 0 | S | OpenParenthesis $ CloseParenthesis |
| 5 | 0 | 1 | 0 | 0 | 1 | $ | $ |
| 6 | 1 | 1 | 7 | 0 | 0 | OpenParenthesis | OpenParenthesis |
| 7 | 0 | 1 | 2 | 1 | 0 | S | OpenParenthesis $ CloseParenthesis |
| 8 | 1 | 1 | 0 | 0 | 0 | CloseParenthesis | CloseParenthesis |
| 9 | 0 | 1 | 0 | 0 | 0 | e | $ CloseParenthesis |

Таблица 1

Методы в приложении 1

### Задание 2

На вход syntaxAnalyser.exe подаётся входной файл:

((()))

Вызываем: SyntaxAnalyzer.exe table.txt input.txt

На выходе получаем сообщение о том что цепочка верна

“Stack is empty. Well Done!”

Трассировка:

we are now at 1 position

we are now at 4 position

we are now at 2 position

we are now at 6 position

we are now at 7 position

we are now at 2 position

we are now at 6 position

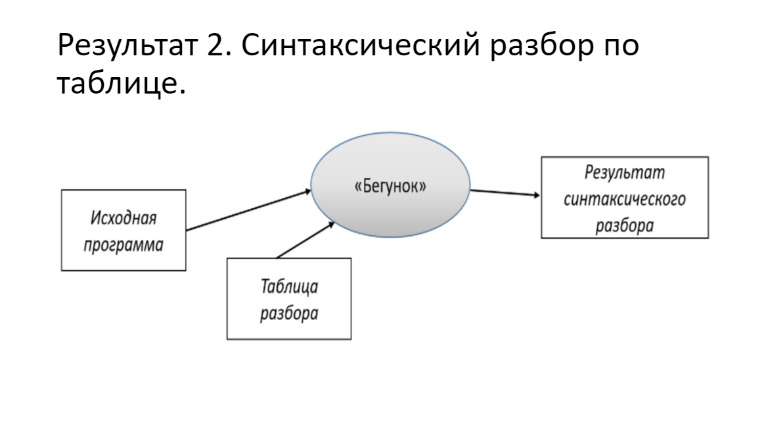
we are now at 7 position

we are now at 2 position

we are now at 6 position

we are now at 7 position

we are now at 3 position

we are now at 9 position

we are now at 8 position

we are now at 8 position

we are now at 8 position

we are now at 5 position

Stack is empty. Well done!

Методы в приложении 2

### Задание 3

Направляющие множества мы получаем во время генерации таблицы (задание 1)

Сначала вызывается метод GetFormingGuideSets, на входе файл с грамматикой grammar.txt.

Метод FillingData парсит грамматику

Метод Forming убирает левую рекурсию добавлением нового правила

Метод AddingGuideCharacters добавляет направляющие множества

* если правая часть правила начинается с терминала, то множество направляющих символов состоит из одного этого терминала
* иначе правая часть начинается с нетерминала A, тогда множество направляющих символов есть First(A)

если при First(A) мы получаем пустой символ, то делаем follow(A) для нетерминального символа для левой части грамматики

избавление рекурсии происходит добавлением нового правила

S -> A $

A -> X | X Y Z

становится

S -> A $

A -> X B

B -> Y Z | ε

| Symbol | Characters |
| --- | --- |
| Z | $ CloseParenthesis OpenParenthesis |
| S | OpenParenthesis |
| S | $ CloseParenthesis |

Таблица 2

Методы в приложении 3

### Задание 4

Генератор использует лексер при представлении определенных символов их типами

В синтаксическом анализаторе он тоже применяется

Методы в приложении 4

## ВЫВОД

В течении семестра изучалась дисциплина ТАиФЯ, где были получены знания о синтаксических анализатора, методах разборов и сценариев использования, что в дальнейшем поможет в трудовой деятельности. Курс был непростым, но очень интересным. В подтверждение усвоения материала мною написан синтаксический анализатор типа LL(1), и соединен вместе с лексером, написанным нами ранее в предыдущей РГР.

## ПРИЛОЖЕНИЯ

**Приложение 1**

std::vector<OutputData> GetGenerateData(const std::vector<OutputDataGuideSets>& inputDatas)

{

std::vector<OutputData> outputDatas;

for (size\_t i = 0; i < inputDatas.size(); ++i)

{

OutputData outputData;

outputData.symbol = inputDatas[i].nonterminal;

outputData.guideCharacters = inputDatas[i].guideCharacters;

if (IsEmptyRule(inputDatas[i].terminals.front()) && IsCheckUniqueness(outputData.guideCharacters, TERMINAL\_END\_SEQUENCE))

{

outputData.guideCharacters.push\_back(TERMINAL\_END\_SEQUENCE);

}

outputData.pointer = i == 0 ? inputDatas.size() + 1 : outputDatas.back().pointer + inputDatas[i - 1].terminals.size();

outputDatas.push\_back(outputData);

}

for (size\_t i = 0; i < inputDatas.size(); ++i)

{

size\_t terminalsSize = inputDatas[i].terminals.size();

for (size\_t j = 0; j < terminalsSize; ++j)

{

bool isNext = j + 1 < terminalsSize;

OutputData outputData;

outputData.symbol = inputDatas[i].terminals[j];

outputData.guideCharacters = std::vector<std::string>{ outputData.symbol };

outputData.isShift = true;

outputData.pointer = outputDatas.size() + 2;

if (!isNext)

{

outputData.pointer = 0;

}

if (IsEndRule(outputData.symbol))

{

outputData.pointer = 0;

outputData.isShift = false;

outputData.isEnd = true;

}

else if (IsEmptyRule(outputData.symbol))

{

outputData.guideCharacters = inputDatas[i].guideCharacters;

outputData.pointer = 0;

outputData.isShift = false;

}

else

{

size\_t row = std::distance(inputDatas.begin(),

std::find\_if(inputDatas.begin(), inputDatas.end(), [&](const OutputDataGuideSets& data) { return data.nonterminal == outputData.symbol; }));

if (row < inputDatas.size())

{

std::vector<std::string> characters;

size\_t repeatCounter = std::count\_if(inputDatas.begin(), inputDatas.end(), [&](const OutputDataGuideSets& data) { return data.nonterminal == outputData.symbol; });

for (size\_t k = 0; k < repeatCounter; ++k)

{

if (k < repeatCounter - 1)

{

outputDatas[row + k].isError = false;

}

for (const auto& character : outputDatas[row + k].guideCharacters)

{

if (IsCheckUniqueness(characters, character))

{

characters.push\_back(character);

}

}

}

outputData.isShift = false;

outputData.pointer = row + 1;

if (isNext)

{

outputData.isStack = true;

}

outputData.guideCharacters = characters;

}

}

outputDatas.push\_back(outputData);

}

}

return outputDatas;

}

void PrintResult(std::ostream& fileOutput, const std::vector<OutputData>& outputDatas)

{

fileOutput << "Number" << TAB << "Shift" << TAB << "Error" << TAB << "Pointer" << TAB << "Stack" << TAB << "End" << TAB << "Symbol" << TAB << TAB << "Characters" << std::endl;

for (size\_t i = 0; i < outputDatas.size(); ++i)

{

OutputData outputData = outputDatas[i];

size\_t counter = i + 1;

std::string symbol = IsNonterminal(outputData.symbol) ? SubstrNonterminal(outputData.symbol) : outputData.symbol;

fileOutput << counter << TAB << outputData.isShift << TAB << outputData.isError << TAB << outputData.pointer << TAB << outputData.isStack << TAB << outputData.isEnd << TAB << symbol << TAB;

PrintInfoVector(fileOutput, outputData.guideCharacters, SPACE);

fileOutput << std::endl;

}

}

**Приложение 2**

void RecursiveMethod(const std::vector<InputTableData>& inputTable, std::vector<OutputTableData>& outputTable, std::stack<size\_t>& stack,

const InputTableData& inputData, const std::vector<Lexeme>& lexemes, size\_t& index, bool isEnd)

{

std::cout << "we are now at " << inputData.number << " position" << "\n";

if (inputData.isEnd && stack.empty())

{

return;

}

size\_t nextIndex = index + 1;

if (inputData.isShift && nextIndex < lexemes.size())

{

++index;

}

if (nextIndex >= lexemes.size())

{

isEnd = true;

}

std::string currentSymbol = PrecariousLexemeTypeToString(lexemes[index].type);

// std::string currentSymbol = lexemes[index].lexeme;

if (inputData.isStack)

{

size\_t stackItem = inputData.number + 1;

outputTable.push\_back({ inputData.number, Action::Add, stackItem, currentSymbol });

stack.push(stackItem);

InputTableData newInputData = GetNewInputData(inputTable, currentSymbol, inputData.pointer, isEnd);

RecursiveMethod(inputTable, outputTable, stack, newInputData, lexemes, index, isEnd);

}

else

{

if (inputData.pointer == 0)

{

if (stack.empty())

{

throw std::runtime\_error("Error. Empty stack");

}

size\_t pointer = stack.top();

stack.pop();

outputTable.push\_back({ inputData.number, Action::Delete, pointer, currentSymbol });

InputTableData newInputData = GetNewInputData(inputTable, currentSymbol, pointer, isEnd);

RecursiveMethod(inputTable, outputTable, stack, newInputData, lexemes, index, isEnd);

}

else

{

InputTableData newInputData = GetNewInputData(inputTable, currentSymbol, inputData.pointer, isEnd);

RecursiveMethod(inputTable, outputTable, stack, newInputData, lexemes, index, isEnd);

}

}

}

**Приложение 3**

std::vector<OutputDataGuideSets> GetFormingGuideSets(std::istream& fileInput)

{

std::vector<std::string> nonterminals;

std::vector<std::string> terminals;

std::vector<InputData> inputDatas;

std::vector<OutputDataGuideSets> outputDatas;

FillingData(fileInput, inputDatas, nonterminals, terminals);

Forming(inputDatas, outputDatas, nonterminals, terminals);

AddingGuideCharacters(outputDatas, nonterminals, terminals);

return outputDatas;

}

void FillingData(std::istream& fileInput, std::vector<InputData>& inputDatas, std::vector<std::string>& nonterminals, std::vector<std::string>& terminals)

{

std::string line;

while (std::getline(fileInput, line))

{

std::istringstream iss(line);

std::string str;

InputData inputData;

bool isTerminal = false;

while (iss >> str)

{

if (str == DELIMITER)

{

isTerminal = true;

}

else if (isTerminal)

{

std::string newStr = str;

if (!IsNonterminal(str))

{

bool isEndSequence = IsEmptyRule(str) || IsEndRule(str);

if (!isEndSequence)

{

std::stringstream strm(str);

Lexer lexer(strm);

const auto [type, lexeme, lineNum, linePos] = lexer.GetLexeme();

newStr = LexemeTypeToString(type);

// newStr = lexeme;

}

if (IsCheckUniqueness(terminals, newStr))

{

terminals.push\_back(isEndSequence ? NONTERMINAL\_END\_SEQUENCE : newStr);

}

}

inputData.terminals.push\_back(newStr);

}

else

{

inputData.nonterminal = str;

if (IsCheckUniqueness(nonterminals, str))

{

nonterminals.push\_back(str);

}

}

}

inputDatas.push\_back(inputData);

}

}

void AddingGuideCharacters(std::vector<OutputDataGuideSets>& outputDatas,

const std::vector<std::string>& nonterminals, const std::vector<std::string>& terminals)

{

std::vector<PairStringVectorPair> transitions;

std::vector<PairStringBool> characters;

std::for\_each(nonterminals.begin(), nonterminals.end(), [&](std::string str) { characters.emplace\_back(str, false); });

std::for\_each(terminals.begin(), terminals.end(), [&](std::string str) { characters.emplace\_back(str, false); });

std::for\_each(nonterminals.begin(), nonterminals.end(), [&](std::string str) { transitions.emplace\_back(str, characters); });

BuildingFirstRelationship(outputDatas, transitions, characters);

BuildingFirstPlusRelationship(transitions);

ValidateTransitions(transitions);

for (auto & outputData : outputDatas)

{

outputData.guideCharacters = GetGuideCharsByRule(outputDatas, outputData);

}

}

void BuildingFirstRelationship(const std::vector<OutputDataGuideSets>& outputDatas, std::vector<PairStringVectorPair>& transitions, const std::vector<PairStringBool>& characters)

{

for (auto& outputData : outputDatas)

{

size\_t row = std::distance(transitions.cbegin(), GetIteratorFindIfVector(transitions, outputData.nonterminal));

size\_t column = std::distance(characters.cbegin(), GetIteratorFindIfVector(characters, outputData.terminals.front()));

if ((row < transitions.size()) && (column < characters.size()))

{

if (IsEmptyRule(outputData.terminals.front()))

{

SearchStartingTerminalsEmptyRules(outputDatas, outputData.nonterminal, outputData.nonterminal, transitions, characters);

}

else

{

transitions[row].second[column].second = true;

}

}

}

}

void BuildingFirstPlusRelationship(std::vector<PairStringVectorPair>& transitions)

{

for (auto it = transitions.rbegin(); it != transitions.rend(); ++it)

{

for (size\_t j = 0; j < transitions.size(); ++j)

{

if ((\*it).second[j].second)

{

size\_t row = std::distance(transitions.cbegin(), GetIteratorFindIfVector(transitions, (\*it).second[j].first));

for (size\_t k = 0; k < transitions[row].second.size(); ++k)

{

if (transitions[row].second[k].second)

{

(\*it).second[k].second = true;

}

}

}

}

}

}

void PrintResultGuideSets(std::ostream& output, const std::vector<OutputDataGuideSets>& outputDatas)

{

for (const auto& outputData : outputDatas)

{

output << outputData.nonterminal << SPACE << DELIMITER << SPACE;

PrintInfoVector(output, outputData.terminals, SPACE);

output << SPACE << "/" << SPACE;

PrintInfoVector(output, outputData.guideCharacters, SPACE);

output << std::endl;

}

}

**Приложение 4**

int main(int argc, char\* argv[])

{

if (argc != 4)

{

std::cout << "Invalid input, should be: <exe> <table> <sentence> <output>" << std::endl;

return 1;

}

std::ifstream table(argv[1]);

std::ifstream sentence(argv[2]);

std::ofstream output(argv[3]);

if (!table.is\_open() || !sentence.is\_open())

{

std::cerr << "This file does not exist" << std::endl;

return 1;

}

try

{

std::vector<InputTableData> tableIn;

std::vector<OutputTableData> tableOut;

std::vector<Lexeme> lexemes = GetAllLexemes(sentence);

if (!lexemes.empty())

{

if (auto& last = lexemes.back(); last.type == LexemeType::EndOfFile)

{

last.lexeme = END\_CHAIN;

}

}

InitInputTable(table, tableIn);

MakeProcess(tableIn, tableOut, lexemes);

PrintResult(output, tableOut);

}

catch (const std::exception& ex)

{

std::cerr << ex.what() << std::endl;

}

}