РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ЗАДАНИЯ

**Задание 1**

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

<Z> => <S> $

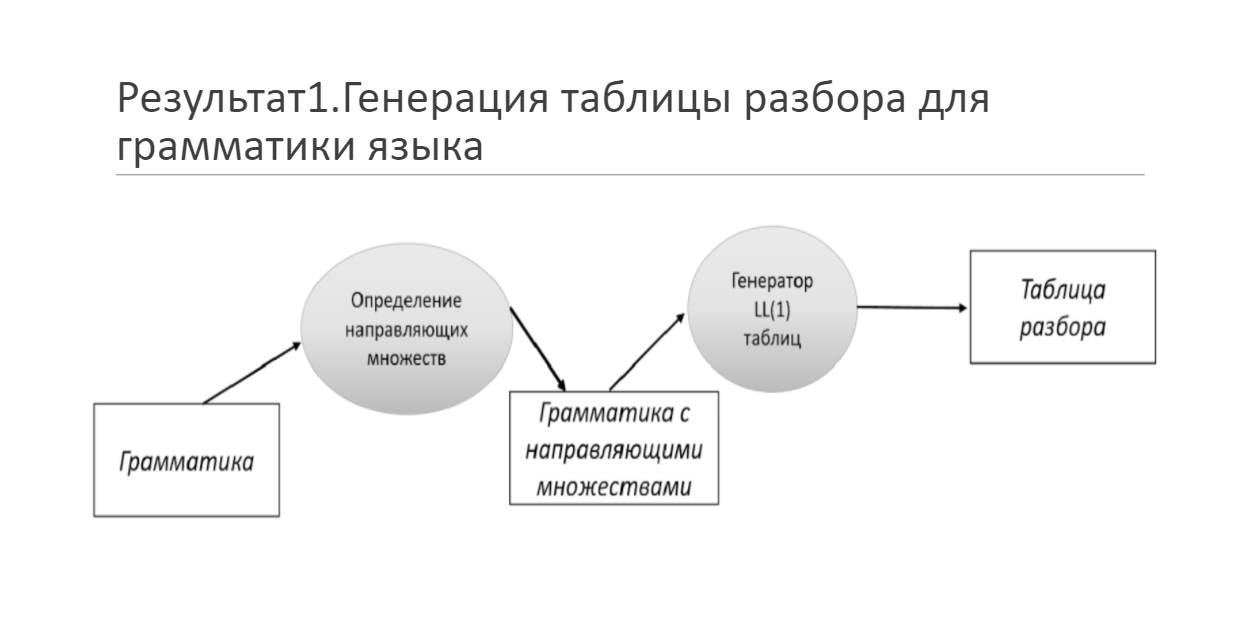
<S> => ( <S> )

<S> => e

Вызываем generator.exe input.txt output.txt

В выходном файле 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 |



### Задание 2

На вход syntaxAnalyser.exe подаётся входной файл:

((()))

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

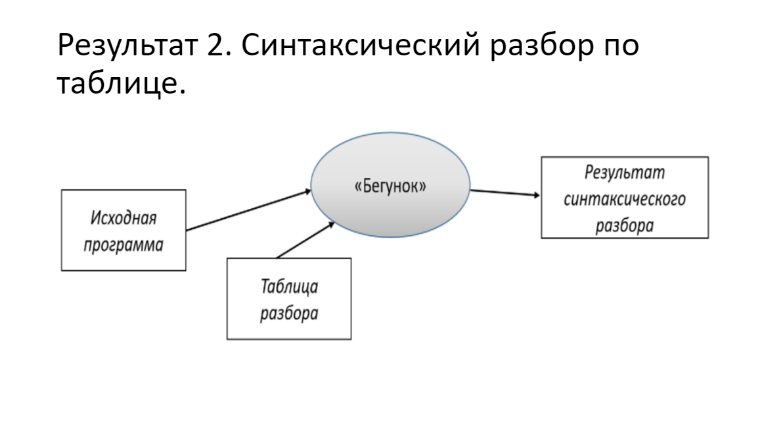
we are now at 8 position

we are now at 8 position

we are now at 8 position

we are now at 5 position

Stack is empty. Well done!



**Задание 3**

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

Последние 2 столбца являются направляюшими множествами

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

### Задание 4

Генератор использует лексер при представлении определённых символов их типами

В синтаксическом анализаторе он тоже применяется

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()) // TODO:

{

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;

}

}

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;

}

}