**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**



ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

**Лабораторная работа №1**

по дисциплине: Теория автоматов и формальных языков

тема: «Формальные грамматики. Выводы»

Выполнил: ст. группы ПВ-233

Мовчан Антон Юрьевич

Проверили:

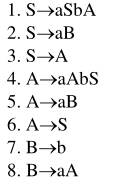
ст. пр. Рязанов Юрий Дмитриевич

Белгород 2025 г.

**Лабораторная работа №1**

Цель работы: изучить основные понятия теории формальных языков и грамматик.

**Вариант 8**

****

1. Найти терминальную цепочку α, |α| > 10, для которой существует не менее двух левых выводов в заданной КС-грамматике (см. варианты заданий). Записать различные левые выводы этой цепочки. Построить деревья вывода. Определить последовательности правил, применяемые при этих выводах.

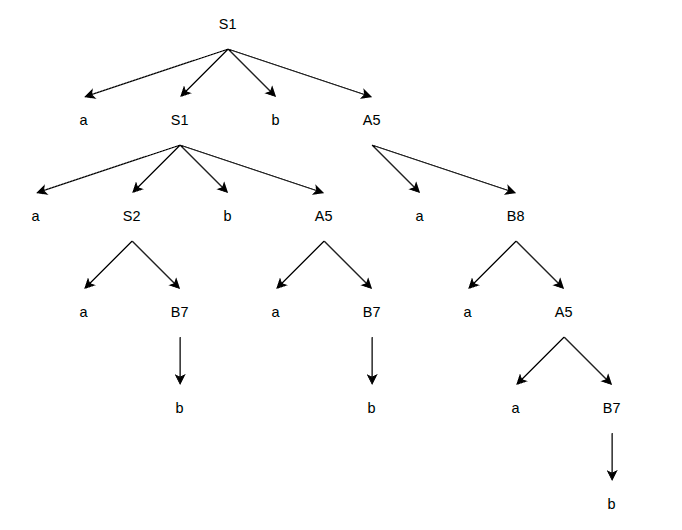
Левый вывод 1:

=> => => => => => => => => =>

Терминальная цепочка:

Последовательность правил: 1 1 2 7 5 7 5 8 5 7

Дерево вывода:



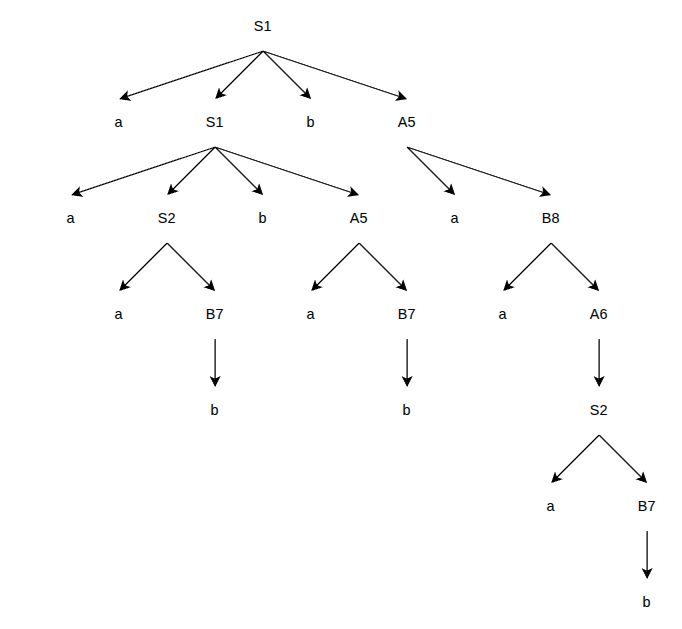
Левый вывод 2:

=> => => => => => => => => => =>

Терминальная цепочка:

Последовательность правил: 1 1 2 7 5 7 5 8 6 2 7

Дерево вывода:



2. Написать программу, которая определяет, можно ли применить заданную последовательность правил при левом выводе терминальной цепочки в заданной КС-грамматике, формирует левый вывод и линейную скобочную форму дерева вывода.

Обработать программой последовательности правил, полученные в п.1.

Примечание. Если к нетерминалу А в процессе вывода применяется правило с номером n, то в выводе и в линейной скобочной форме дерева вывода после нетерминала А должен быть символ с кодом n.

Исходный код:

#include *<bits/stdc++.h>*

**using** **namespace** **std**;

*/\* ---------- 1. Структура правила и инициализация грамматики ---------- \*/*

**struct** **Rule**

{

string lhs; *// левые члены (непустой символ)*

string rhs; *// правые члены*

int num; *// номер правила*

};

**static** **const** vector<Rule> rules = {

{"S", "aSbA", 1},

{"S", "aB", 2},

{"S", "A", 3},

{"A", "aAbS", 4},

{"A", "aB", 5},

{"A", "S", 6},

{"B", "b", 7},

{"B", "aA", 8}};

*/\* ---------- 2. Узел дерева вывода ---------- \*/*

**struct** **Node**

{

string symbol; *// символ*

bool terminal; *// true – терминал*

int ruleNum = -1; *// номер правила, которым он был развернут*

vector<Node \*> children; *// потомки*

Node(**const** string &s, bool t = true, int r = -1) : symbol(s), terminal(t), ruleNum(r) {}

};

*/\* ---------- 3. Поиск правила по номеру ---------- \*/*

**const** Rule \*findRule(int num)

{

**for** (**const** **auto** &r : rules)

**if** (r.num == num)

**return** &r;

**return** **nullptr**;

}

void printNode(Node \*n, string &out)

{

**if** (n->terminal)

{

out += n->symbol;

**return**;

}

out += '(' + n->symbol + to\_string(n->ruleNum);

**for** (Node \*c : n->children)

printNode(c, out);

out += ')';

}

string getCurrentState(vector<Node \*> &curList, int appliedRule)

{

bool added = false;

string derivation;

**for** (Node \*n : curList)

{

derivation += n->symbol;

**if** (!added && !n->terminal && (appliedRule != -1))

{

derivation += to\_string(appliedRule);

added = true;

}

}

**return** derivation;

}

*/\* ---------- 5. Главная функция ---------- \*/*

int main()

{

ios::sync\_with\_stdio(false);

cin.tie(**nullptr**);

*/\* ---------- ввод последовательности номеров правил ---------- \*/*

vector<int> seqRules;

int x;

**while** (cin >> x)

seqRules.push\_back(x);

*/\* ---------- инициализация дерева и списка текущих нод ---------- \*/*

Node \*root = **new** Node("S", false);

vector<Node \*> curList = {root};

bool ok = true;

**for** (int ruleNum : seqRules)

{

**const** Rule \*r = findRule(ruleNum);

**if** (!r)

{

ok = false;

**break**;

}

*/\* поиск левого самого первого нерегулярного символа \*/*

size\_t pos = 0;

**while** (pos < curList.size() && curList[pos]->terminal)

++pos;

**if** (pos == curList.size())

{

ok = false;

**break**;

}

Node \*target = curList[pos];

**if** (target->symbol != r->lhs)

{

ok = false;

**break**;

}

*/\* развернуть правило \*/*

*/\* строка левого вывода \*/*

string deriv = getCurrentState(curList, r->num);

cout << deriv << " => ";

target->ruleNum = r->num;

target->terminal = false;

target->children.clear();

*/\* создаём потомков и обновляем список текущих нод \*/*

vector<Node \*> newNodes;

**for** (char ch : r->rhs)

{

**if** (isupper(ch))

{ *// нетерминал*

Node \*child = **new** Node(string(1, ch), false, r->num);

target->children.push\_back(child);

newNodes.push\_back(child);

}

**else**

{ *// терминал*

Node \*child = **new** Node(string(1, ch), true, r->num);

target->children.push\_back(child);

newNodes.push\_back(child);

}

}

curList.erase(curList.begin() + pos, curList.begin() + pos + 1);

curList.insert(curList.begin() + pos, newNodes.begin(), newNodes.end());

}

*/\* ---------- проверка, что все стали терминалами ---------- \*/*

**for** (Node \*n : curList)

**if** (!n->terminal)

ok = false;

*/\* ---------- вывод результата ---------- \*/*

**if** (!ok)

{

cout << "Невозможно применить заданную последовательность правил.**\n**";

**return** 0;

}

*/\* линейная скобочная форма \*/*

string deriv = getCurrentState(curList, -1);

cout << deriv << '\n';

string linear;

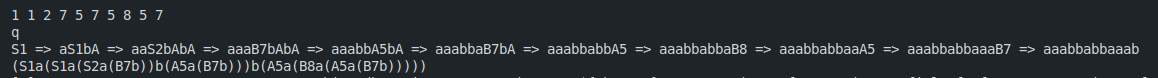
printNode(root, linear);

cout << linear << '\n';

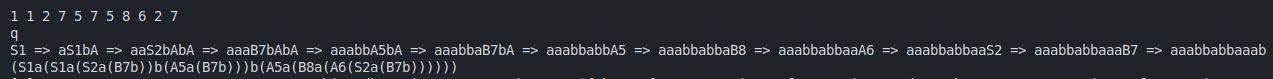
**return** 0;

}

Результат выполнения программы, для п1:



Результат выполнения программы, для п2:



3. Найти последовательность правил р, |р| > 10, которую можно применить при произвольном выводе терминальной цепочки, но нельзя применить при левом или правом выводе в заданной КС-грамматике (см. варианты заданий).

Записать вывод v, в процессе которого применяется последовательность правил р. Построить дерево вывода.

Записать левый и правый выводы, эквивалентные выводу v.

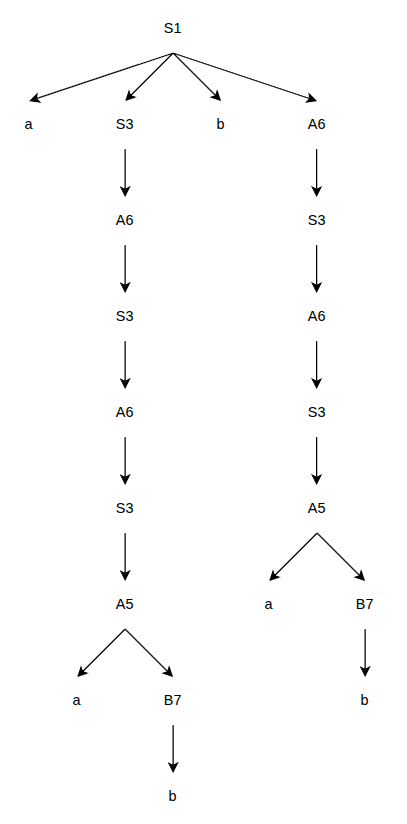
**Произвольный: вывод:**

=> => => => => => => => => => => => => =>

Последовательность правил p: 1 3 6 6 3 3 6 6 3 3 5 5 7 7

Терминальная цепочка:

Дерево вывода:



**Левый вывод:**

=> => => => => => => => => => => => => =>

Последовательность правил: 1 3 6 3 6 3 5 7 6 3 6 3 5 7

Терминальная цепочка:

**Правый вывод:**

=> => => => => => => => => => => => => =>

Последовательность правил: 1 6 3 6 3 5 7 3 6 3 6 3 5 7

Терминальная цепочка:

4. Написать программу, которая определяет, можно ли применить заданную последовательность правил р при выводе терминальной цепочки в заданной КС-грамматике и формирует линейную скобочную форму дерева вывода. Если последовательность правил р можно применить при выводе v терминальной цепочки, то программа должна вывести последовательность правил, применяемую при левом выводе,

эквивалентном выводу v.

Обработать программой последовательность правил, найденную в п.3.

Примечание. Если к нетерминалу А в процессе вывода применяется правило с номером n, то в линейной скобочной форме дерева вывода после нетерминала А должен быть символ с кодом n.

Исходный код:

#include *<bits/stdc++.h>*

**using** **namespace** **std**;

*// Грамматика:*

*// 1. S -> a S b A*

*// 2. S -> a B*

*// 3. S -> A*

*// 4. A -> a A b S*

*// 5. A -> a B*

*// 6. A -> S*

*// 7. B -> b*

*// 8. B -> a A*

**struct** **Node**

{

char sym; *// 'S','A','B' для нетерминалов или 'a','b' для терминалов*

int rule; *// номер правила, использованного для раскрытия узла, -1 если не раскрыт (лист)*

vector<shared\_ptr<Node>> ch;

Node(char s = 0) : sym(s), rule(-1) {}

};

bool is\_nonterm(char c) { **return** c >= 'A' && c <= 'Z'; }

bool is\_terminal(char c) { **return** !is\_nonterm(c); }

map<int, pair<char, string>> productions;

shared\_ptr<Node> deep\_copy(**const** shared\_ptr<Node> &root)

{

**if** (!root)

**return** **nullptr**;

**auto** n = make\_shared<Node>(root->sym);

n->rule = root->rule;

**for** (**auto** &c : root->ch)

n->ch.push\_back(deep\_copy(c));

**return** n;

}

void collect\_leaves(**const** shared\_ptr<Node> &node, vector<shared\_ptr<Node>> &leaves)

{

**if** (node->ch.empty())

leaves.push\_back(node);

**else**

**for** (**auto** &c : node->ch)

collect\_leaves(c, leaves);

}

void expand\_leaf(shared\_ptr<Node> leaf, int rule\_num, **const** string &rhs)

{

leaf->rule = rule\_num;

leaf->ch.clear();

**for** (char c : rhs)

leaf->ch.push\_back(make\_shared<Node>(c));

}

bool leaves\_all\_terminals(**const** shared\_ptr<Node> &root)

{

vector<shared\_ptr<Node>> leaves;

collect\_leaves(root, leaves);

**for** (**auto** &l : leaves)

**if** (is\_nonterm(l->sym))

**return** false;

**return** true;

}

string linear\_form(**const** shared\_ptr<Node> &node)

{

**if** (node->ch.empty())

**return** string(1, node->sym);

string res;

res.push\_back(node->sym);

**if** (node->rule != -1)

res += to\_string(node->rule);

res += "(";

**for** (**auto** &c : node->ch)

res += linear\_form(c);

res += ")";

**return** res;

}

void collect\_leftmost\_seq(**const** shared\_ptr<Node> &node, vector<int> &seq)

{

**if** (node->ch.empty())

**return**;

**if** (node->rule != -1)

seq.push\_back(node->rule);

**for** (**auto** &c : node->ch)

collect\_leftmost\_seq(c, seq);

}

*// --- Новый: вывод текущей цепочки ---*

void print\_current\_chain(int step, **const** shared\_ptr<Node> &tree, int ruleNum)

{

vector<shared\_ptr<Node>> leaves;

collect\_leaves(tree, leaves);

bool added = false;

**for** (**auto** &l : leaves)

{

cout << (l->sym);

**if** (!added && is\_nonterm(l->sym))

{

cout << ruleNum;

added = true;

}

}

cout << " => ";

}

*// DFS с выводом цепочки на каждом шаге*

bool dfs\_try(**const** shared\_ptr<Node> &tree, **const** vector<int> &seq, int idx, shared\_ptr<Node> &result\_tree)

{

**if** (idx == (int)seq.size())

{

**if** (leaves\_all\_terminals(tree))

{

result\_tree = tree;

**return** true;

}

**else**

**return** false;

}

int rule\_num = seq[idx];

**if** (productions.find(rule\_num) == productions.end())

**return** false;

char lhs = productions[rule\_num].first;

string rhs = productions[rule\_num].second;

vector<shared\_ptr<Node>> leaves;

collect\_leaves(tree, leaves);

shared\_ptr<Node> newtree;

bool anyAttempt = false;

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

{

**if** (leaves[i]->sym == lhs)

{

print\_current\_chain(i, tree, rule\_num);

anyAttempt = true;

newtree = deep\_copy(tree);

vector<shared\_ptr<Node>> newleaves;

collect\_leaves(newtree, newleaves);

expand\_leaf(newleaves[i], rule\_num, rhs);

**if** (dfs\_try(newtree, seq, idx + 1, result\_tree))

**return** true;

}

}

**if** (!anyAttempt)

**return** false;

**return** false;

}

void leftmost\_derivation(**const** vector<int> &seq)

{

**auto** tree = make\_shared<Node>('S');

**for** (int rule\_num : seq)

{

*// собираем листья*

vector<shared\_ptr<Node>> leaves;

collect\_leaves(tree, leaves);

*// ищем левыйmost нетерминал, который подходит под правило*

char lhs = productions[rule\_num].first;

string rhs = productions[rule\_num].second;

bool applied = false;

**for** (**auto** &l : leaves)

{

**if** (l->sym == lhs)

{

expand\_leaf(l, rule\_num, rhs);

applied = true;

**break**;

}

}

**if** (!applied)

{

cout << "Правило " << rule\_num << " не может быть применено. Прерывание.**\n**";

**return**;

}

*// вывод текущей цепочки*

bool used = false;

**for** (**auto** &l : leaves)

{

cout << l->sym;

**if** (!used && is\_nonterm(l->sym))

{

cout << rule\_num;

used = true;

}

}

cout << " => ";

}

*// вывод финальной цепочки*

vector<shared\_ptr<Node>> leaves;

collect\_leaves(tree, leaves);

**for** (**auto** &l : leaves)

cout << l->sym;

cout << "**\n**";

}

int main()

{

ios::sync\_with\_stdio(false);

cin.tie(**nullptr**);

productions[1] = {'S', string("aSbA")};

productions[2] = {'S', string("aB")};

productions[3] = {'S', string("A")};

productions[4] = {'A', string("aAbS")};

productions[5] = {'A', string("aB")};

productions[6] = {'A', string("S")};

productions[7] = {'B', string("b")};

productions[8] = {'B', string("aA")};

string line;

**if** (!getline(cin, line))

**return** 0;

**while** (!line.empty() && isspace(line.back()))

line.pop\_back();

**while** (!line.empty() && isspace(line.front()))

line.erase(line.begin());

vector<int> seq;

**if** (line.find(' ') != string::npos)

{

stringstream ss(line);

int x;

**while** (ss >> x)

seq.push\_back(x);

}

**else**

{

**for** (char c : line)

{

**if** (isdigit(c))

seq.push\_back(c - '0');

**else** **if** (!isspace(c))

{

cerr << "Неожиданный символ во входе: '" << c << "'**\n**";

**return** 0;

}

}

}

**if** (seq.empty())

{

cout << "Пустая последовательность правил. Выход.**\n**";

**return** 0;

}

**auto** start = make\_shared<Node>('S');

shared\_ptr<Node> result\_tree = **nullptr**;

bool ok = dfs\_try(start, seq, 0, result\_tree);

**if** (!ok)

{

cout << "Последовательность правил \*\*НЕ\*\* может быть применена для получения терминальной цепочки в заданной грамматике.**\n**";

**return** 0;

}

string lin = linear\_form(result\_tree);

vector<shared\_ptr<Node>> leaves;

collect\_leaves(result\_tree, leaves);

string v;

**for** (**auto** &l : leaves)

v.push\_back(l->sym);

vector<int> left\_seq;

collect\_leftmost\_seq(result\_tree, left\_seq);

cout << v << "**\n**";

cout << "**\n**Линейная скобочная форма дерева вывода:**\n**"

<< lin << "**\n**";

cout << "**\n**Последовательность правил при левом выводе:**\n**";

**for** (size\_t i = 0; i < left\_seq.size(); ++i)

{

**if** (i)

cout << " ";

cout << left\_seq[i];

}

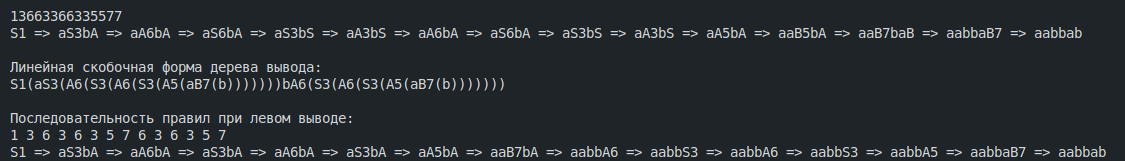
cout << "**\n**";

leftmost\_derivation(left\_seq);

**return** 0;

}

Результат выполнения программы:



Вывод: в ходе выполнения л.р. я изучил основные понятия теории формальных языков и грамматик.