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.

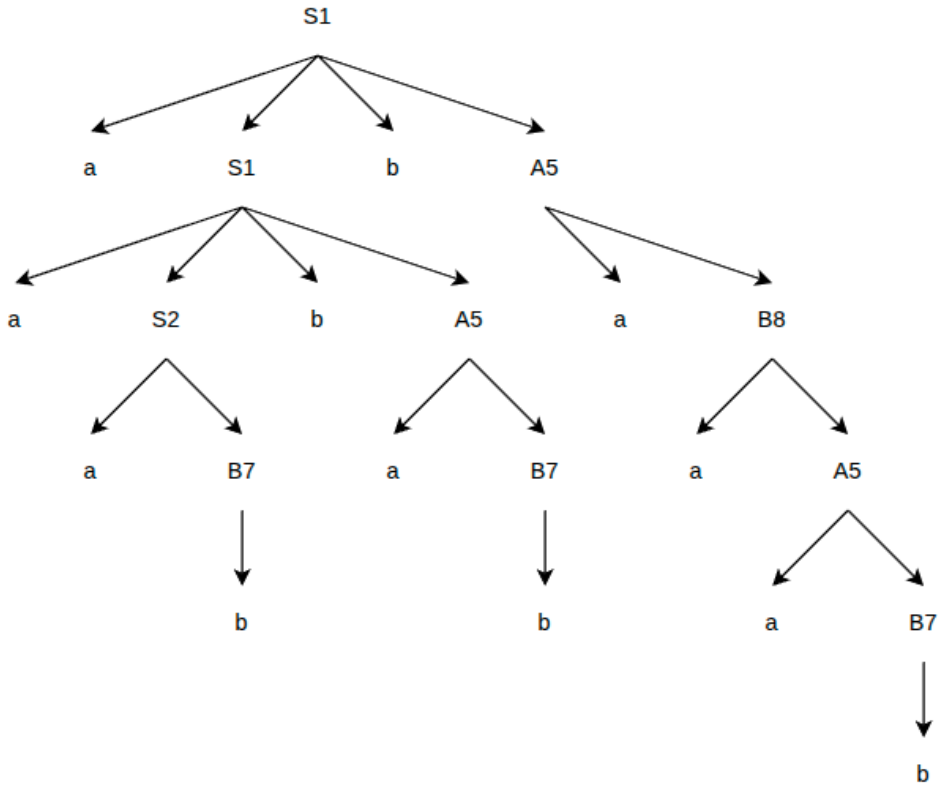
Вариант 8

1. S→aSbA
2. S→aB
3. S→A
4. A→aAbS
5. A→aB
6. A→S
7. B→b
8. B→aA

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.

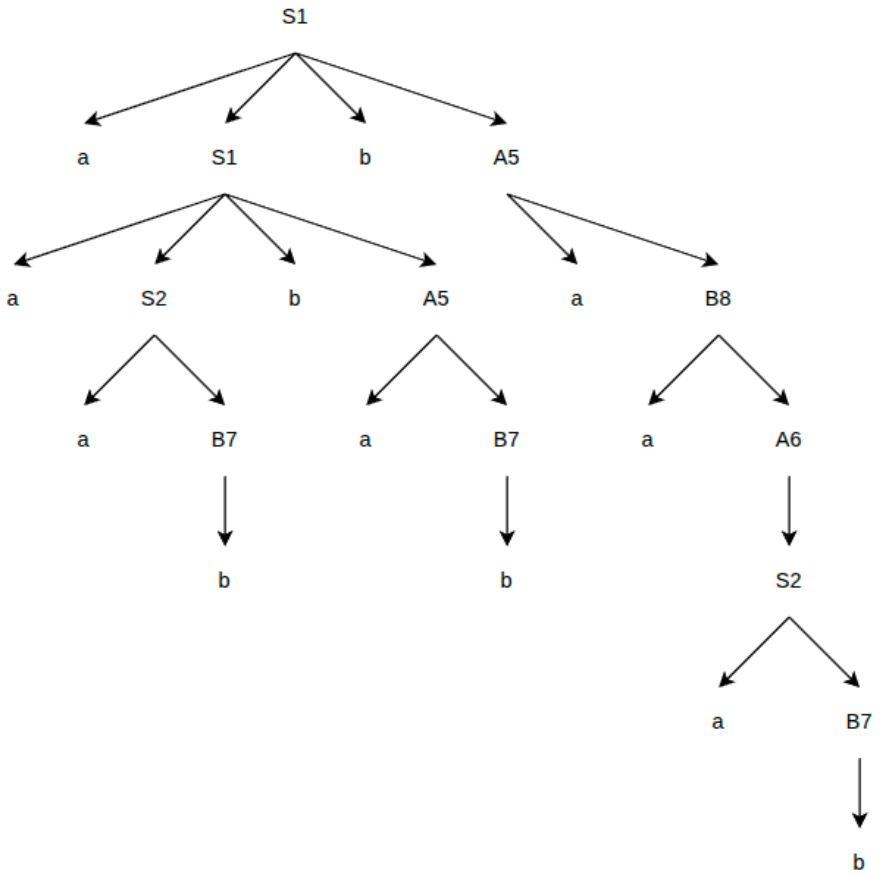
Левый вывод 1:
$S_1 => aS_1bA => aaS_2bAbA => aaaB_7bAbA => aaabbA_5bA => aaabbaB_7bA => aaabbabbA_5 => aaabbabbaB_8 => aaabbabbaaA_5 => aaabbabbaaaB_7 => aaabbabbaaab$

Терминальная цепочка: *aaabbabbaaab*
Последовательность правил: 1 1 2 7 5 7 5 8 5 7
Дерево вывода:



Левый вывод 2:
$S_1 => a\,S_1\,bA => aa\,S_2\,bAbA => aaa\,B_7\,bAbA => aaabb\,A_5\,bA => aaabba\,B_7\,bA => aaabbabb\,A_5 =>$
$aaabbabba\,B_8 => aaabbabbaa\,A_6 => aabbabbaa\,S_2 => aaabbabbaaa\,B_7 => aaabbabbaaab$

Терминальная цепочка: *aaabbabbaaab*
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 A в процессе вывода применяется правило с номером n, то в выводе и в линейной скобочной форме дерева вывода после нетерминала A 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:

```
1 1 2 7 5 7 5 8 5 7
q
S1 => aS1bA => aaS2bAbA => aaaB7bAbA => aaabbA5bA => aaabbaB7bA => aaabbabbA5 => aaabbabbaB8 => aaabbabbaaA5 => aaabbabbaaaB7 => aaabbabbaaab
(S1a(S1a(S2a(B7b))b(A5a(B7b)))b(A5a(B8a(A5a(B7b)))))
```

Результат выполнения программы, для п2:

```
1 1 2 7 5 7 5 8 6 2 7
q
S1 => aS1bA => aaS2bAbA => aaaB7bAbA => aaabbA5bA => aaabbaB7bA => aaabbabbA5 => aaabbabbaB8 => aaabbabbaaA6 => aaabbabbaaS2 => aaabbabbaaaB7 => aaabbabbaaab
(S1a(S1a(S2a(B7b))b(A5a(B7b)))b(A5a(B8a(A6(S2a(B7b))))))
```

3. Найти последовательность правил p, |p| > 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 p. Построить дерево вывода.

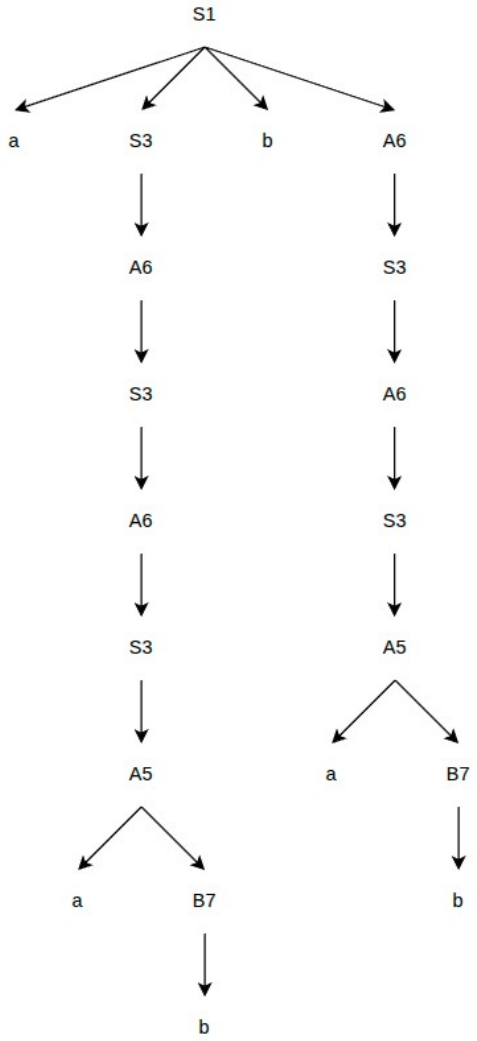
Записать левый и правый выводы, эквивалентные выводу v.

Произвольный: вывод:

$S_1 => a\,S_3\,bA => aAb\,A_6 => a\,A_6\,bS => aSb\,S_3 => a\,S_3\,bA => aAb\,a_6 => a\,A_6\,bS => aSb\,S_3 => a\,S_3\,bA => aAb\,A_5 => a\,A_5\,baB => aaBba\,B_7 => aa\,B_7\,bab => aabbab$

Последовательность правил p: 1 3 6 6 3 3 6 6 3 3 5 5 7 7
Терминальная цепочка: *aabbab*

Дерево вывода:



Левый вывод:

$S_1 => a\,S_3\,bA => a\,A_6\,bA => a\,S_3\,bA => a\,A_6\,bA => a\,S_3\,bA => a\,A_5\,bA => aa\,B_7\,bA => aabb\,A_6 =>$
$aabb\,S_3 => aabb\,A_6 => aabb\,S_3 => aabb\,A_5 => aabba\,B_7 => aabbab$

Последовательность правил: 1 3 6 3 6 3 5 7 6 3 6 3 5 7
Терминальная цепочка: *aabbab*

Правый вывод:

$S_1 => aSb\,A_6 => aSb\,S_3 => aSb\,A_6 => aSb\,S_3 => aSb\,A_5 => aSba\,B_7 => a\,S_3\,bab => a\,A_6\,bab => a\,S_3\,bab$
$=> a\,A_6\,bab => a\,S_3\,bab => a\,A_5\,bab => aa\,B_7\,bab => aabbab$

Последовательность правил: 1 6 3 6 3 5 7 3 6 3 6 3 5 7
Терминальная цепочка: *aabbab*

4. Написать программу, которая определяет, можно ли применить заданную последовательность правил p при выводе терминальной цепочки в заданной КС-грамматике и формирует линейную скобочную форму дерева вывода. Если последовательность правил p 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 A в процессе вывода применяется правило с номером n, то в линейной скобочной форме дерева вывода после нетерминала A 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 << "\nЛинейная скобочная форма дерева вывода:\n"
         << lin << "\n";
    cout << "\nПоследовательность 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:

```
13663366335577
S1 => aS3bA => aA6bA => aS6bA => aS3bS => aA3bS => aA6bA => aS6bA => aS3bS => aA3bS => aA5bA => aaB5bA => aaB7baB => aabbaB7 => aabbab

Линейная скобочная форма дерева вывода:
S1(aS3(A6(S3(A6(S3(A5(aB7(b)))))))bA6(S3(A6(S3(A5(aB7(b)))))))

Последовательность правил при левом выводе:
1 3 6 3 6 3 5 7 6 3 6 3 5 7
S1 => aS3bA => aA6bA => aS3bA => aA6bA => aS3bA => aA5bA => aaB7bA => aabbA6 => aabbS3 => aabbA6 => aabbS3 => aabbA5 => aabbaB7 => aabbab
```

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