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)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

# Лабораторная работа №1

по дисциплине: Теория автоматов и формальных языков
тема: Формальные грамматики. Выводы

Выполнил: студент ПВ-233
Мороз Роман Алексеевич

Проверил:
Рязанов Юрий Дмитриевич

Белгород 2025 г.

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

Вариант 9

1. S→SbSa
2. S→Sa
3. S→A
4. A→aS
5. A→aB
6. A→b
7. B→b
8. B→Aa

## Задания

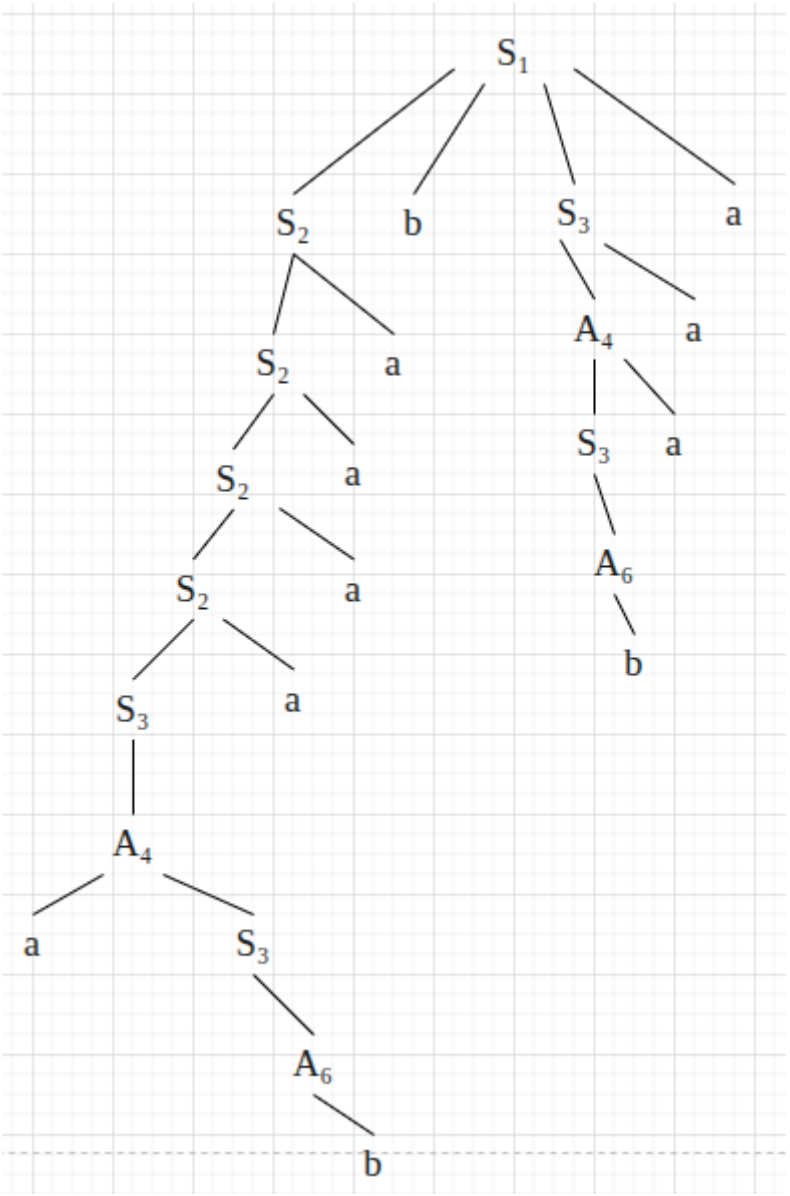
1. Найти терминальную цепочку α, |α| > 10, для которой существует не менее двух левых выводов в заданной КС-грамматике (см. варианты заданий). Записать различные левые выводы этой цепочки. Построить деревья вывода. Определить последовательности правил, применяемые при этих выводах.

Цепочка: abaaaababa

**Последовательность правил:** (1, 2, 2, 2, 2, 3, 4, 3, 6, 3, 4, 3, 6)

Вывод 1: $S_1$ => $S_2$bSa => $S_2$abSa => $S_2$aabSa => $S_2$aaabSa => $S_3$aaaabSa => $A_4$aaaabSa => a$S_3$aaaabSa => a$A_6$aaaabSa => abaaaab$S_3$a => abaaaab$A_4$a => abaaaaba$S_3$a => abaaaaba$A_6$a => abaaaababa

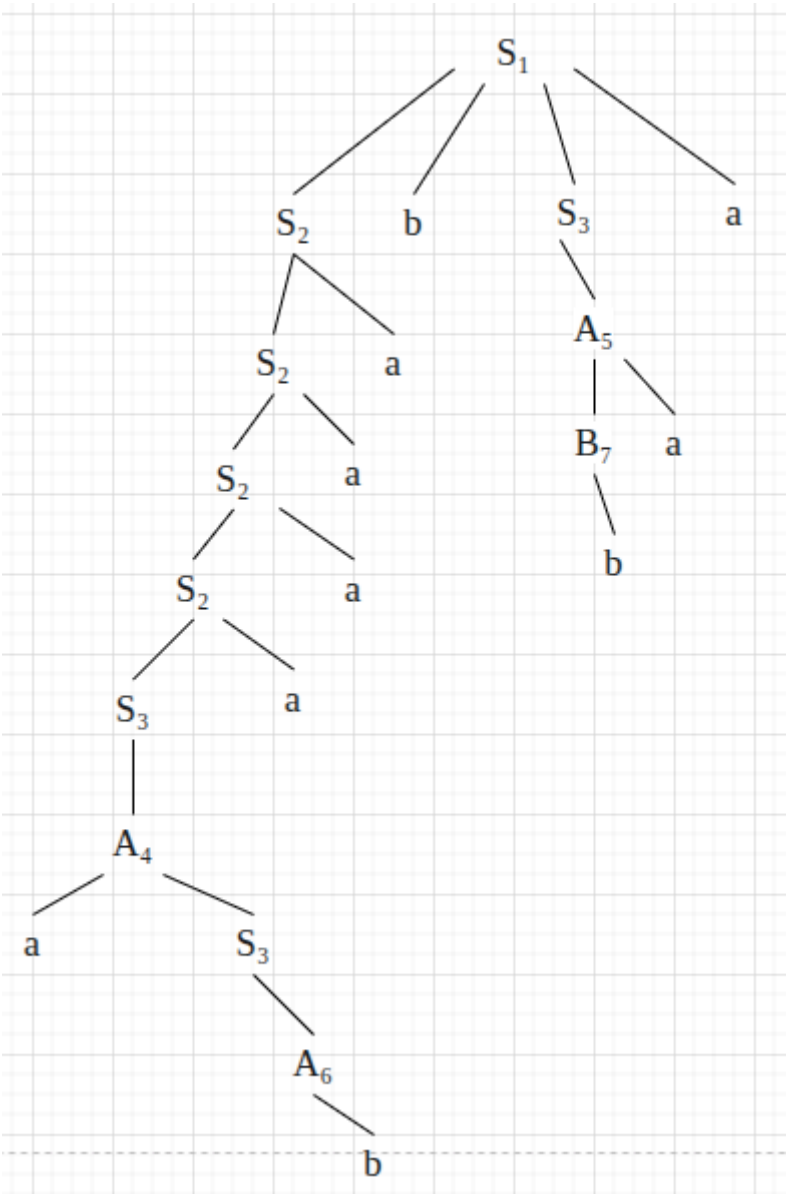
Дерево для вывода 1:




**Последовательность правил:** (1, 2, 2, 2, 2, 3, 4, 3, 6, 3, 5, 7)

Вывод 2: $S_1$ => $S_2$bSa => $S_2$abSa => $S_2$aabSa => $S_2$aaabSa => $S_3$aaaabSa => $A_4$aaaabSa => a$S_3$aaaabSa => a$A_6$aaaabSa => abaaaab$S_3$a => abaaaab$A_5$a => abaaaabaB$_7$a => abaaaababa

Дерево для вывода 2:




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.

*Код программы:*

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <memory>
#include <stdexcept>
#include <cctype>
#include <sstream>
#include <utility>

/**
* @struct Rule
* @brief Хранит одно правило грамматики.
*/
struct Rule {
    char non_terminal;
    std::string replacement;
};

/**
* @class TreeNode
* @brief Узел в дереве вывода.
*/
class TreeNode {
public:
    std::string value;
    int rule_number = 0; // Номер правила, примененного для создания этого узла
    std::vector<std::unique_ptr<TreeNode>> children;

    explicit TreeNode(std::string val) : value(std::move(val)) {}

    // Запрещаем копирование, чтобы избежать проблем с владением памятью
```

```
    TreeNode(const TreeNode&) = delete;
    TreeNode& operator=(const TreeNode&) = delete;
};

/**
* @class DerivationTree
* @brief Представляет и строит дерево вывода (реализует логику вашего класса Tree).
*/
class DerivationTree {
public:
    DerivationTree(char start_symbol) {
        root = std::make_unique<TreeNode>(std::string(1, start_symbol));
    }

    /**
     * @brief Находит самый левый узел для замены и добавляет к нему дочерние элементы.
     */
    void apply_rule(const Rule& rule, int rule_num) {
        TreeNode* node_to_expand = find_leftmost_expandable_node(root.get(), rule.non_terminal);
        if (!node_to_expand) {
            throw std::runtime_error("Ошибка дерева: Не найден подходящий нетерминал для
применения правила.");
        }

        node_to_expand->rule_number = rule_num;
        for (char symbol : rule.replacement) {
            node_to_expand->children.push_back(std::make_unique<TreeNode>(std::string(1,
symbol)));
        }
    }

    /**
     * @brief Генерирует линейную скобочную форму дерева (реализует логику getTree).
     */
```

```
    std::string get_bracket_form() const {
        std::stringstream ss;
        generate_bracket_form_recursive(root.get(), ss);
        return ss.str();
    }

private:
    std::unique_ptr<TreeNode> root;

    // Рекурсивный поиск самого левого нераскрытого нетерминала (аналог findNode)
    TreeNode* find_leftmost_expandable_node(TreeNode* current, char target_non_terminal) {
        if (!current) return nullptr;
        // Если узел - искомый нетерминал и у него еще нет дочерних узлов, мы нашли его.
        if (current->value.length() == 1 && current->value[0] == target_non_terminal &&
current->children.empty()) {
            return current;
        }
        // В противном случае, рекурсивно ищем в дочерних узлах слева направо.
        for (const auto& child : current->children) {
            TreeNode* result = find_leftmost_expandable_node(child.get(), target_non_terminal);
            if (result) {
                return result;
            }
        }
        return nullptr;
    }

    // Рекурсивная генерация скобочной формы
    void generate_bracket_form_recursive(const TreeNode* node, std::stringstream& ss) const {
        if (!node) return;

        ss << node->value[0];
        if (isupper(node->value[0])) { // Если это нетерминал
```

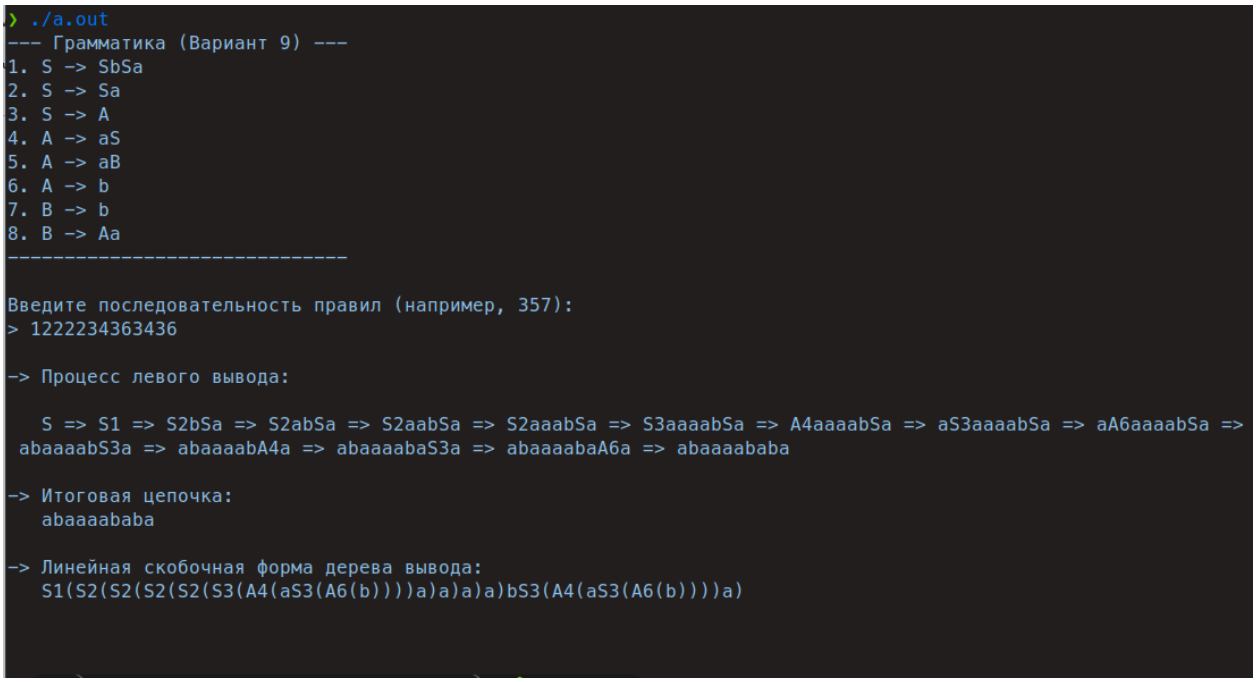
```
            ss << node->rule_number;
            if (!node->children.empty()) {
                ss << "(";
                for (const auto& child : node->children) {
                    generate_bracket_form_recursive(child.get(), ss);
                }
                ss << ")";
            }
        }
    }
};

/**
* @class DerivationProcessor
* @brief Управляет процессом левого вывода (реализует логику move).
*/
class DerivationProcessor {
public:
    explicit DerivationProcessor(std::string start_chain) : current_chain(std::move(start_chain))
{}

    /**
     * @brief Применяет правило к текущей цепочке, соблюдая левый вывод.
     */
    void apply_rule(const Rule& rule) {
        for (size_t i = 0; i < current_chain.length(); ++i) {
            if (current_chain[i] == rule.non_terminal) {
                current_chain.replace(i, 1, rule.replacement);
                return;
            }
        }
        throw std::runtime_error("Ошибка вывода: Не найден подходящий нетерминал '" +
std::string(1, rule.non_terminal) + "' для применения правила.");
    }
```

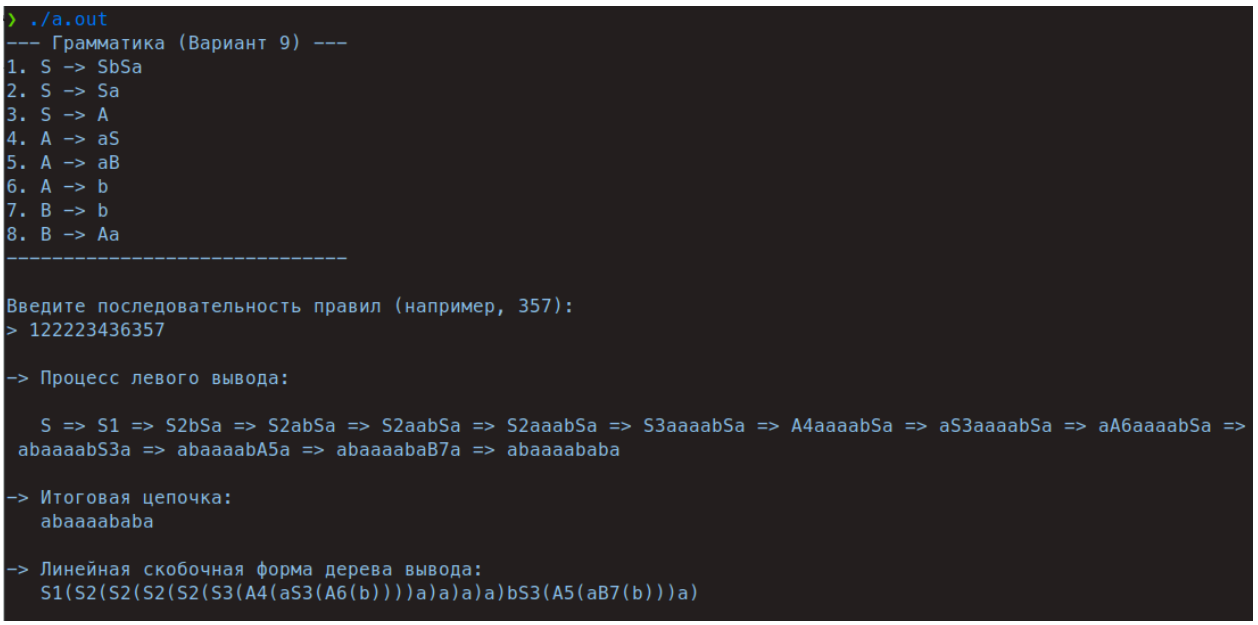
```
    const std::string& get_chain() const { return current_chain; }

private:
    std::string current_chain;
};

// --- Основная функция ---

void print_rules(const std::vector<Rule>& rules) {
    std::cout << "--- Грамматика (Вариант 9) ---\n";
    for(size_t i = 0; i < rules.size(); ++i) {
        std::cout << i + 1 << ". " << rules[i].non_terminal << " -> " << rules[i].replacement <<
"\n";
    }
    std::cout << "------------------------------\n\n";
}

int main() {
    // Используем вашу грамматику "Вариант 9"
    const std::vector<Rule> rules = {
        {'S', "SbSa"}, {'S', "Sa"},   {'S', "A"},
        {'A', "aS"},   {'A', "aB"},   {'A', "b"},
        {'B', "b"},    {'B', "Aa"}
    };

    print_rules(rules);

    std::cout << "Введите последовательность правил (например, 357):\n> ";
    std::string rule_sequence_str;
    std::cin >> rule_sequence_str;

    try {
```

```
        DerivationProcessor processor("S");

        DerivationTree tree('S');



        std::cout << "\n-> Процесс левого вывода:\n\n";

        std::string display_chain = "S";

        std::cout << "   " << display_chain;



        for (char rule_char : rule_sequence_str) {

            if (!isdigit(rule_char)) {

                throw std::runtime_error("Ошибка ввода: '" + std::string(1, rule_char) + "' не
является цифрой.");

            }

            int rule_index = rule_char - '1'; // '1' -> 0, '2' -> 1, ...

            if (rule_index < 0 || (size_t)rule_index >= rules.size()) {

                throw std::runtime_error("Ошибка ввода: Правила с номером " + std::string(1,
rule_char) + " не существует.");

            }

            const Rule& selected_rule = rules[rule_index];



            // Модифицируем отображаемую строку ПЕРЕД заменой

            bool rule_added = false;

            for(size_t i = 0; i < display_chain.length(); ++i) {

                if(isupper(display_chain[i])) {

                    display_chain.insert(i + 1, std::to_string(rule_index + 1));

                    rule_added = true;

                    break;

                }

            }

            std::cout << " => " << display_chain;



            // Применяем правило к дереву и реальной цепочке

            tree.apply_rule(selected_rule, rule_index + 1);

            processor.apply_rule(selected_rule);

            display_chain = processor.get_chain();
```

```
        }

        std::cout << " => " << processor.get_chain() << "\n\n";



        // Проверка результата

        const std::string final_chain = processor.get_chain();

        for (char c : final_chain) {

            if (isupper(c)) {

                std::cout << "[ПРЕДУПРЕЖДЕНИЕ] Итоговая цепочка не является терминальной. Найден
нетерминал '" << c << "'.\n";

                break;

            }

        }



        std::cout << "-> Итоговая цепочка:\n   " << final_chain << "\n\n";

        std::cout << "-> Линейная скобочная форма дерева вывода:\n   " << tree.get_bracket_form()
<< "\n\n";



    } catch (const std::runtime_error& e) {

        std::cerr << "\n[ПРОИЗОШЛА ОШИБКА]\n" << e.what() << "\n\n";

        return 1;

    }



    return 0;

}
```

*Пример работы программы:*




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).

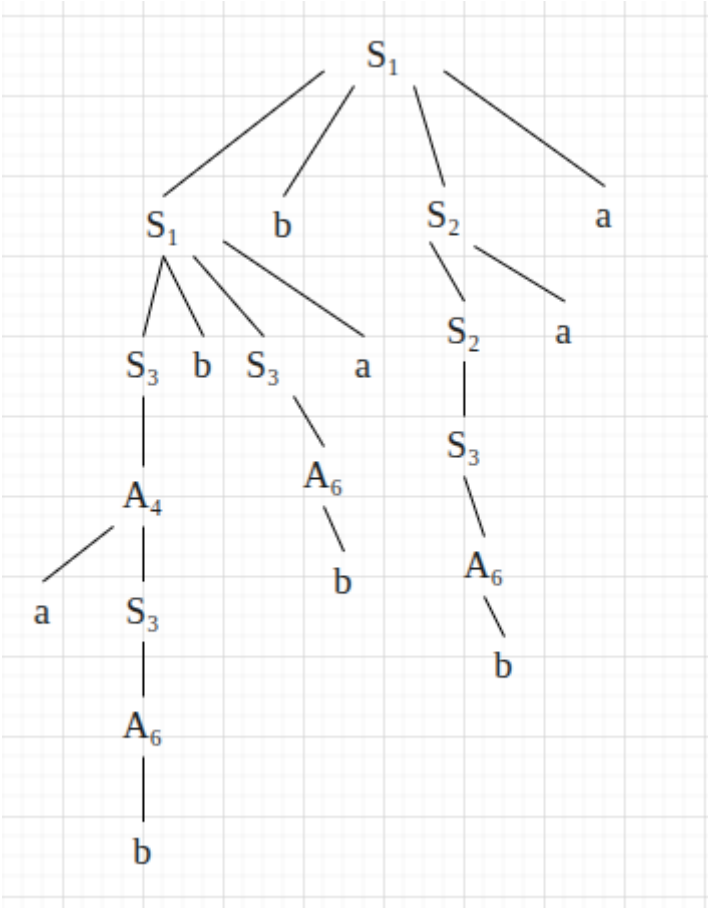
Записать вывод v, в процессе которого применяется последовательность правил p. Построить дерево вывода. Записать левый и правый выводы, эквивалентные выводу v.

**1. Произвольный вывод v**

**Последовательность правил** (1, 1, 3, 6, 3, 4, 3, 6, 2, 2, 3, 6)

**Итоговая цепочка:** abbababbaaa

$S_1 \Rightarrow S_1bSa \Rightarrow SbS_3abSa \Rightarrow SbA_6abSa \Rightarrow S_3bbabSa \Rightarrow A_4bbabSa \Rightarrow aS_3bbabSa \Rightarrow aA_6bbabSa \Rightarrow abbbabS_2a \Rightarrow abbbabS_2aa \Rightarrow abbbabA_3aaa \Rightarrow abbbabA_6aaa \Rightarrow abbababbaaa$




**2. Эквивалентный левый вывод**

**Последовательность правил:** (1, 1, 3, 4, 3, 6, 3, 6, 2, 2, 3, 6)
**Итоговая цепочка:** abbababbaaa

$S_1 \Rightarrow S_1bSa \Rightarrow A_3bSabSa \Rightarrow aS_4bSabSa \Rightarrow aA_3bSabSa \Rightarrow ab_6bSabSa \Rightarrow abA_3abSa \Rightarrow abba_6bSa \Rightarrow abbabS_2a \Rightarrow abbabS_2aa \Rightarrow abbabA_3aaa \Rightarrow abbbabA_6aaa \Rightarrow abbababbaaa$

**3. Эквивалентный правый вывод**

**Последовательность правил:** (1, 2, 2, 3, 6, 1, 3, 6, 3, 4, 3, 6)
**Итоговая цепочка:** abbababbaaa

$S_1 \Rightarrow SbS_2a \Rightarrow SbS_2aa \Rightarrow SbA_3aaa \Rightarrow Sbb_6aaa \Rightarrow S_1bbaaa \Rightarrow SbS_3abbaaa \Rightarrow SbA_6abbaaa \Rightarrow S_3bbabbaaa \Rightarrow A_4bbabbaaa \Rightarrow aS_3bbabbaaa \Rightarrow aA_6bbabbaaa \Rightarrow abbbababbaaa$

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.

*Код программы:*

```
#include <iostream>
#include <vector>
#include <string>


typedef struct node{
   node **el;
   int countEl;
   char *characters;
   int size;
   char *rules;
}node;


class Tree {
   node *root;
   void findNode(node *n, bool &end, node *&ans, char c, char p);
   Tree() = default;
public:
   Tree(char c);
   void setNode(char *array, char c, char p);
   void getTree(node *n, std::vector<char> &str, int &index);
   ~Tree();
```

```
    node * getRoot();
};


Tree::Tree(char c){
    root = new node();
    root->characters = new char[2];
    root->characters[0] = c;
    root->characters[1] = '\0';
    root->size = 1;
    root->countEl = 0;
    root->el = nullptr;
    root->rules = new char[root->size];
}


void Tree::findNode(node *n, bool &end, node *&ans, char c, char p){
    if(end){
        return;
    }
    if(n->countEl == 0){
        for(int i = 0; i < n->size; i++){
            if(n->characters[i] < 97 && n->characters[i] == c){
                end = true;
                n->el = new node*[n->size];
                n->countEl = n->size;
                for(int j = 0; j < n->size; j++){
                    if(i!=j){
                        n->el[j] = nullptr;
                    } else{
                        n->el[j] = new node();
                        ans = n->el[j];
                        n->rules[j] = p;
                    }
```

```
            }

            return;

        }

    }

}

else{

    for(int i = 0; i < n->size; i++){

        if(n->characters[i] < 97 && n->el[i] == nullptr && !end && n->characters[i] == c){

            end = true;

            n->el[i] = new node();

            ans = n->el[i];

            n->rules[i] = p;

            return;

        }

        else if(n->el[i] != nullptr){

            if(!end) {

                findNode(n->el[i], end, ans, c, p);

            }

        }

    }

}

}



void Tree::setNode(char *array, char x, char p){

char *c = array;

int count = 0;

while (*c!= '\0'){

    count++;

    c++;

}

node *curr = nullptr;

bool end = false;
```

```
    findNode(root, end, curr, x, p);

    if(curr) {

        curr->characters = new char[count + 1];

        for(int i = 0 ; i < count; i++){

            curr->characters[i] = array[i];

        }

        curr->characters[count] = '\0';

        curr->size = count;

        curr->countEl = 0;

        curr->el = nullptr;

        curr->rules = new char[count];

    }

}



void Tree::getTree(node *n, std::vector<char> &str, int &index){

    for(int i = 0; i < n->size; i++){

        str.push_back(n->characters[i]);

        if(n->characters[i] < 97){

            str.push_back(n->rules[i]);

            str.push_back('(');

        }

        if(n->el != nullptr && n->el[i] != nullptr){

            getTree(n->el[i], str, index);

            str.push_back(')');

        }

    }

}



void deleteTree(node *n){

    if (n == nullptr) return;

    for(int i = 0; i < n->size; i++){

        if(n->el != nullptr && n->el[i] != nullptr){
```

```
            deleteTree(n->el[i]);
            delete n->el[i];
        }
    }
    delete[] n->el;
    delete[] n->characters;
    delete[] n->rules;
}

Tree::~Tree(){
    deleteTree(root);
}

node * Tree::getRoot(){
    return root;
}

typedef struct P{
    std::string x;
    std::string y;
}P;

int move(P p, char *&str, int &capacity, int &size){
    char current = '\0';
    for(int i = 0; i < size; i++){
        if(str[i] < 96 && str[i] == p.x[0]){
            current = str[i];
            break;
        }
    }
    if(current == '\0'){
        std::cout << "\nНетерминалы не найдены";
```

```
        return 1;
    }

    int new_len = size + p.y.length() - p.x.length();
    char *tempStr = new char[new_len + 1];
    bool change = false;
    int newSize = 0;
    for(int i = 0; i < size; i++){
        if(!change && str[i] == current){
            for(char c : p.y){
                tempStr[newSize++] = c;
            }
            change = true;
        }
        else{
            tempStr[newSize++] = str[i];
        }
    }

    if(new_len > capacity){
        delete[] str;
        capacity = new_len * 2;
        str = new char[capacity+1];
    }

    for(int i = 0; i < newSize; i++){
        str[i] = tempStr[i];
    }
    size = newSize;
    delete[] tempStr;
    return 0;
}
```

```
char *setRule(char *c, char rule, int &size, int &capacity, char x){
   char *res = new char[capacity];
   char *begin = c;
   int flag = 1;
   int sizeRes = 0;
   while(*begin != '\0'){
       res[sizeRes++] = *begin;
       if(*begin < 97 && flag && x == *begin){
           flag = 0;
           res[sizeRes++] = rule;
       }
       begin++;
   }
   res[sizeRes] = '\0';
   return res;
}


char *solve(Tree &tree, char buf[], const std::vector<char> *v = nullptr){


   std::string sequence_str = "";
   if (v != nullptr) {
       for (char ch : *v) {
           sequence_str += ch;
       }
       // Копируем в buf, так как остальная часть кода его использует
       for(size_t i = 0; i < sequence_str.length(); ++i) {
           buf[i] = sequence_str[i];
       }
       buf[sequence_str.length()] = '\0';
   } else {
       sequence_str = buf;
```

```
}



P p1 = {"S", "SbSa"};

P p2 = {"S", "Sa"};

P p3 = {"S", "A"};

P p4 = {"A", "aS"};

P p5 = {"A", "aB"};

P p6 = {"A", "b"};

P p7 = {"B", "b"};

P p8 = {"B", "Aa"};

P rules[] = {p1, p2, p3, p4, p5, p6, p7, p8};


char *ans = new char[21];

ans[0] = 'S';

int size = 1;

int capacity = 20;


std::cout << "S";


for(char c : sequence_str){

    std::cout << " => ";


    if(c > '8' || c < '1'){

        std::cout << "\nНеверное обозначение: " << c << std::endl;

        exit(1);

    }


    int rule_idx = c - '1';


    ans[size] = '\0';

    char *result = setRule(ans, c, size, capacity, rules[rule_idx].x[0]);
```

```
        std::cout << result;
        delete[] result;

        if (v == nullptr) {
             tree.setNode(&rules[rule_idx].y[0], rules[rule_idx].x[0], c);
        }

        if(move(rules[rule_idx], ans, capacity, size) == 1){
            exit(1);
        }
    }

    ans[size] = '\0';
    std::cout << " => " << ans;

    return ans;
}

int main() {
    std::cout << "Введите последовательность правил\n> ";
    char buf[100];
    std::cin >> buf;

    std::cout << "\n============================================================\n";
    Tree tree = Tree('S');
    char *ans = solve(tree, buf);
    std::cout << "\n\n";

    char *begin = ans;
    while (*begin != '\0'){
        if(*begin < 97){
```

```
            std::cout << "[ОШИБКА] Последовательность правил не привела к терминальной цепочке,
существует нетерминал: " << *begin << std::endl;

            delete[] ans;

            return 1;

        }

        begin++;

    }



    std::cout << "============================================================\n";



    std::vector<char> res;

    int index = 0;

    tree.getTree(tree.getRoot(), res, index);



    std::cout << "-> Линейная скобочная форма дерева вывода:\n    ";

    for(char c : res){

        std::cout << c;

    }

    std::cout << "\n\n";



    std::vector<char> ruleV;



    std::cout << "\n-> Последовательность правил для левого вывода:\n";



    for(char c : res){

        if(c <= '9' && c >= '1') {

            std::cout << c;

            ruleV.push_back(c);

        }

    }



    std::cout << "\n\n-> Левый вывод: \n";
```

```
    Tree dummy_tree('S');

    char buf2[100];

    char* ans2 = solve(dummy_tree, buf2, &ruleV);



    std::cout << "\n=======================================================\n\n";



    delete[] ans;

    delete[] ans2;

    return 0;

}
```

*Пример работы программы:*

```
113634362236

=======================================================
   S => S1bSa => S1bSabSa => SbA3abSa => Sbb6abSa => A3bbabSa => aS4bbabSa => aA3bbabSa => ab6bbabSa => abbab
S2a => abbabS2aa => abbabA3aaa => abbbab6aaa => abbababbaaa

=======================================================
-> Линейная скобочная форма дерева вывода:
   S1(S1(A3(aS4(A3(b6)))bA3(b6)a)bS2(S2(A3(b6))a)a)

-> Последовательность правил для левого вывода:
   113436362236

-> Левый вывод:
   S => S1bSa => S1bSabSa => A3bSabSa => aS4bSabSa => aA3bSabSa => ab6bSabSa => abA3abSa => abb6abSa => abbab
S2a => abbabS2aa => abbabA3aaa => abbbab6aaa => abbababbaaa
=======================================================
```

**Вывод:** изучили основные понятия теории формальных языков и грамматик.