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

«Формальные грамматики. Выводы»

по дисциплине

«Теория автоматов и формальных языков»

|  |  |
| --- | --- |
|  | Выполнила:  Ст. группы ПВ-31  Ковалев Павел |
|  | Проверил:  Рязанов Ю. Д.  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.  Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Белгород 2021 г.

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

**Задание №1**

Написать программу, выполняющую левый вывод в заданной КС грамматике.

**Исходные данные:** КС Грамматика.

**Выполнение:** на каждом шаге вывода отображается промежуточная цепочка (на первом шаге она представляет собой начальный не терминал, на последнем – терминальную цепочку) и правила, которые можно применить на данном шаге. Пользователь выбирает одно из предложенных правил и процесс повторяется.

**Результат:** терминальная цепочка, последовательность номеров правил, участвовавших в её выводе, дерево вывода в линейной скобочной форме (ЛСФ ДВ).

**Задание №2**

Выполнить левый (правый) вывод терминальной цепочки в заданной грамматике (см. варианты заданий п.1), построить дерево вывода. Определить, существует ли неэквивалентный вывод полученной цепочки и, если существует, представить его деревом вывода.

**Задание №3**

Написать программу, определяющую, можно ли применить заданную последовательность правил при левом выводе цепочки в заданной КС грамматике.

**Исходные данные:** КС-грамматика и последовательность номеров правил.

**Результат:** да или нет.

**Задание №4**

Для каждой последовательности правил (см. варианты заданий п.2) определить, можно ли её применить при левом (правом) выводе терминальной цепочки в заданной КС-грамматике, и, если можно, построить дерево вывода.

**Задание №5**

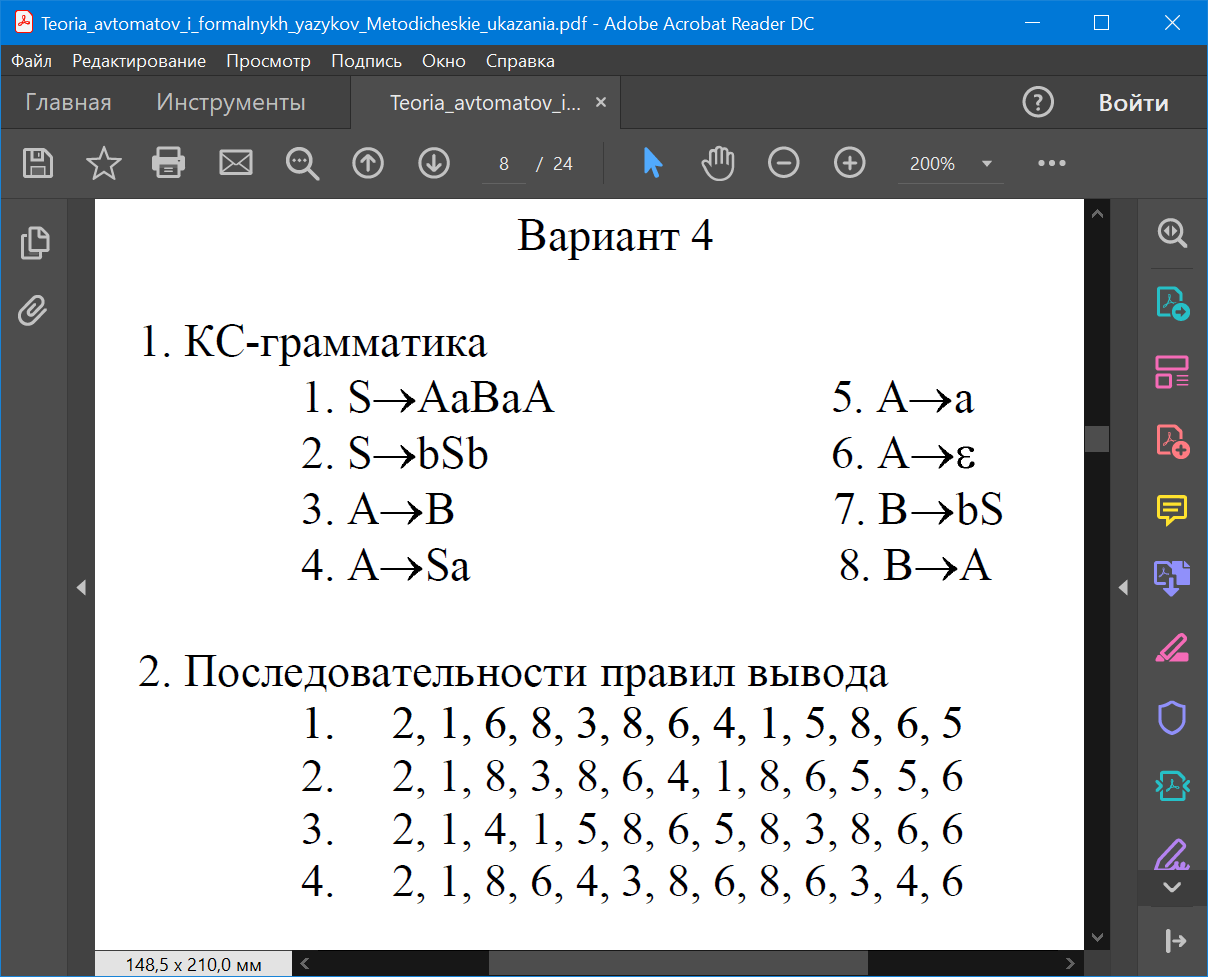
Написать программу, определяющую, можно ли применить заданную последовательность правил при выводе цепочки в заданной КС-грамматике.

**Исходные данные:** КС-грамматика; последовательность номеров правил.

**Результат:** да или нет

**Задание №6**

Для каждой последовательности правил (см. варианты заданий п.2) определить, можно ли её применить при выводе терминальной цепочки в заданной КС-грамматике, и, если можно, построить дерево вывода и записать эквивалентные левый и правый вывод.



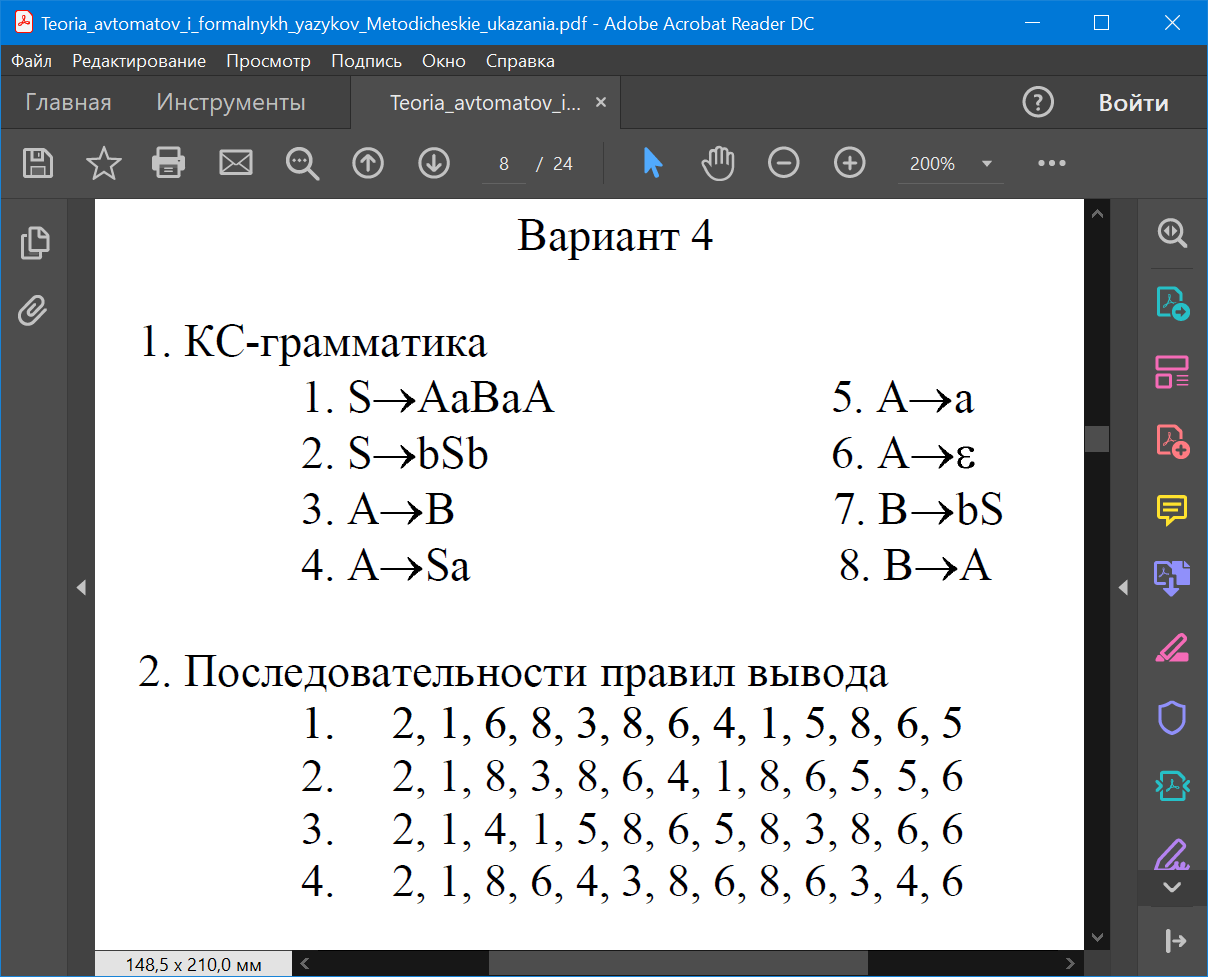
**Задание №2**

Выполнить левый (правый) вывод терминальной цепочки в заданной грамматике (см. варианты заданий п.1), построить дерево вывода. Определить, существует ли неэквивалентный вывод полученной цепочки и, если существует, представить его деревом вывода.

|  |  |
| --- | --- |
| **Левый вывод терминальной цепочки:** | **Неэквивалентный вывод** |
| **Правый вывод терминальной цепочки:** | **Неэквивалентный вывод** |

**Задание №4**

Для каждой последовательности правил (см. варианты заданий п.2) определить, можно ли её применить при левом (правом) выводе терминальной цепочки в заданной КС-грамматике, и, если можно, построить дерево вывода.

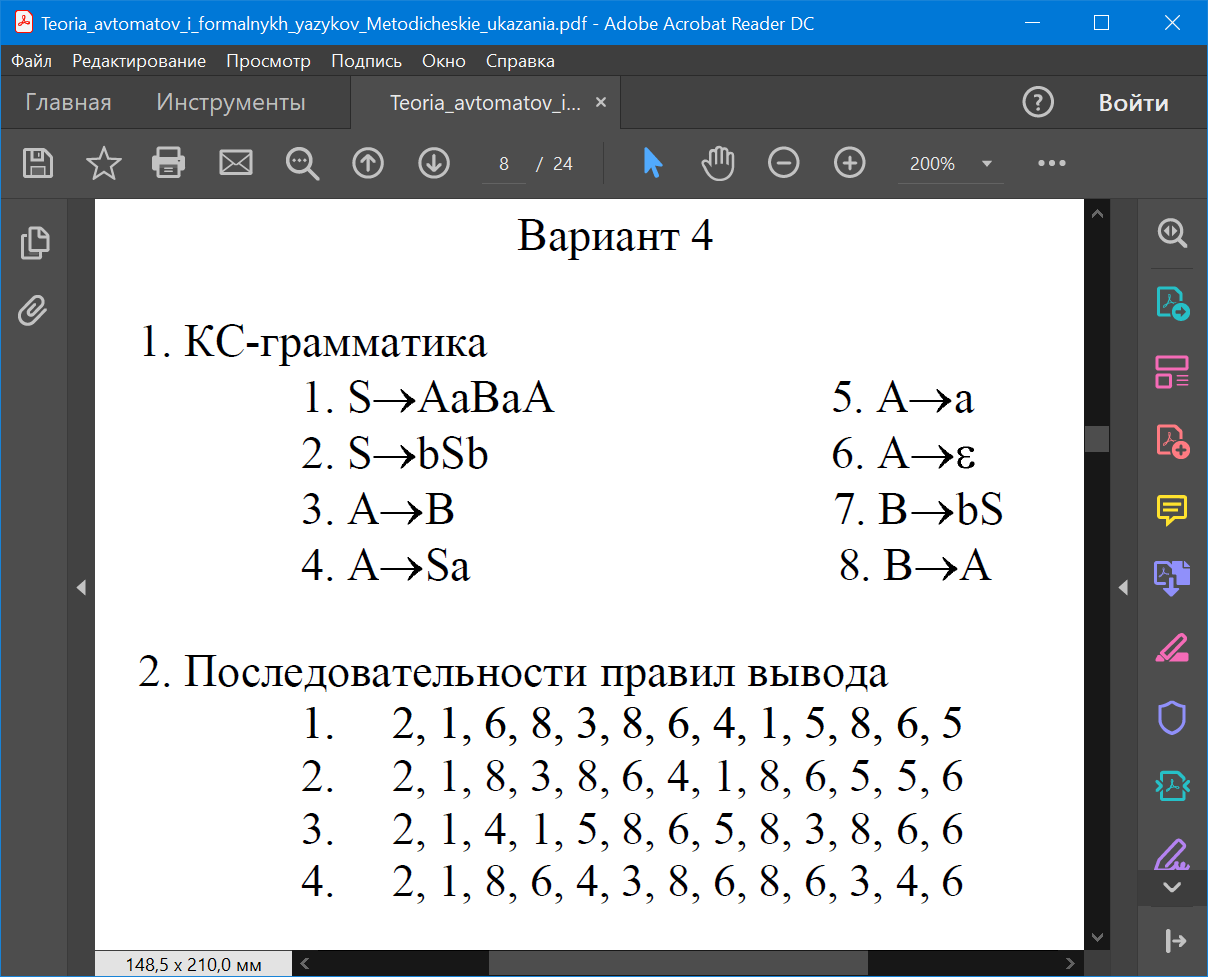


1. Можно применить
2. Нельзя применить
3. Можно применить
4. Нельзя применить

|  |
| --- |
| 1. **2,1,6,8,3,8,6,4,1,5,8,6,5** |
| **3) 2,1,4,1,5,8,6,5,8,3,8,6,6** |

**Задание №6**

Для каждой последовательности правил (см. варианты заданий п.2) определить, можно ли её применить при выводе терминальной цепочки в заданной КС-грамматике, и, если можно, построить дерево вывода и записать эквивалентные левый и правый вывод.



|  |  |
| --- | --- |
| **1-ую** последовательность МОЖНО применить при выводе терминальной цепочки | |
| **2,1,6,8,3,8,6,4,1,5,8,6,5** | |
| ЛЕВЫЙ ВЫВОД | ПРАВЫЙ ВЫВОД |

|  |
| --- |
| **2-ую** последовательность МОЖНО применить при выводе терминальной цепочки |
| **2,1,8,3,8,6,4,1,8,6,5,5,6** |
| ЛЕВЫЙ ВЫВОД    ПРАВЫЙ ВЫВОД |

|  |  |
| --- | --- |
| **3-тью** последовательность МОЖНО применить при выводе терминальной цепочки | |
| **2,1,4,1,5,8,6,5,8,3,8,6,6** | |
| ЛЕВЫЙ ВЫВОД | ПРАВЫЙ ВЫВОД |

|  |
| --- |
| **4-ую** последовательность НЕЛЬЗЯ применить при выводе терминальной цепочки |

**Задание №1**

#include <iostream>

#include <string>

#include <vector>

#include <string>

using namespace std;

bool islow(string s);

string mod\_LSF(string LSF, int cnt);

int brack\_n(string s);

int Find\_up(string s, int cnt);

class KSRule

{

public:

char base;

string output;

KSRule(char base, string output)

: base(base), output(output) { }

};

class KSGramm

{

public:

vector<KSRule> gram;

vector<int> nums;

string LSF = "S";

string applyLeft(string chain, int num);

string handApplyLeft(string chain);

private:

vector<int> getRules(char t);

};

int main()

{

setlocale(0, "");

string s = "S";

KSGramm gram;

gram.gram.push\_back(KSRule('S', "AaBaA"));

gram.gram.push\_back(KSRule('S', "bSb"));

gram.gram.push\_back(KSRule('A', "B"));

gram.gram.push\_back(KSRule('A', "Sa"));

gram.gram.push\_back(KSRule('A', "a"));

gram.gram.push\_back(KSRule('A', ""));

gram.gram.push\_back(KSRule('B', "bS"));

gram.gram.push\_back(KSRule('B', "A"));

string s2 = "";

while (!islow(s))

{

s = gram.handApplyLeft(s);

}

cout << "\nТерминальная цепочка: " << s << endl;

cout << "Последовательность правил: ";

for(auto a : gram.nums)

cout << a+1 << " ";

cout << endl;

int cnt = brack\_n(gram.LSF), g, len = gram.LSF.size();

string snew;

for (int i = 0; i < cnt - 1; i++)

{

snew = mod\_LSF(gram.LSF, i + i + 1);

g = Find\_up(gram.LSF, i + 2);

gram.LSF.insert(g + 1, snew);

}

gram.LSF.erase(len, gram.LSF.size());

cout << "Дерево: " << gram.LSF << endl<< endl;

system("pause");

return 0;

}

int Find\_up(string s, int cnt)

{

for (int i = 0; i < s.size(); i++)

{

if (isupper(s[i]))

cnt--;

if (cnt == 0)

return i;

}

}

int brack\_n(string s)

{

int c = 0;

for (unsigned int i = 0; i < s.length(); i++)

if (s[i] == '(')

c++;

return c;

}

bool islow(string s)

{

for (int i = 0; i < s.size(); i++)

{

if (isupper(s[i]))

return false;

}

return true;

}

string mod\_LSF(string LSF, int cnt)

{

string sub;

for (int i = 0; i < LSF.size(); i++)

{

if (LSF[i] == '(')

cnt--;

if (cnt == -1)

if (LSF[i] == ')')

{

sub = sub + ')';

return sub;

}

else

sub = sub + LSF[i];

}

}

string KSGramm::applyLeft(string chain, int num)

{

for (int i = 0; i < chain.size(); i++)

if (chain[i] == gram[num].base)

{

chain.replace(i, 1, gram[num].output);

break;

}

return chain;

}

string KSGramm::handApplyLeft(string chain)

{

for (int i = 0; i < chain.size(); i++)

{

vector<int> rules = getRules(chain[i]);

if (rules.size() != 0)

{

cout << "Промежуточная цепочка:" << chain << endl << "Можно применить:" << endl;

for (auto a : rules)

{

cout << a+1 << ". " << chain[i] << "->" << gram[a].output << endl;

}

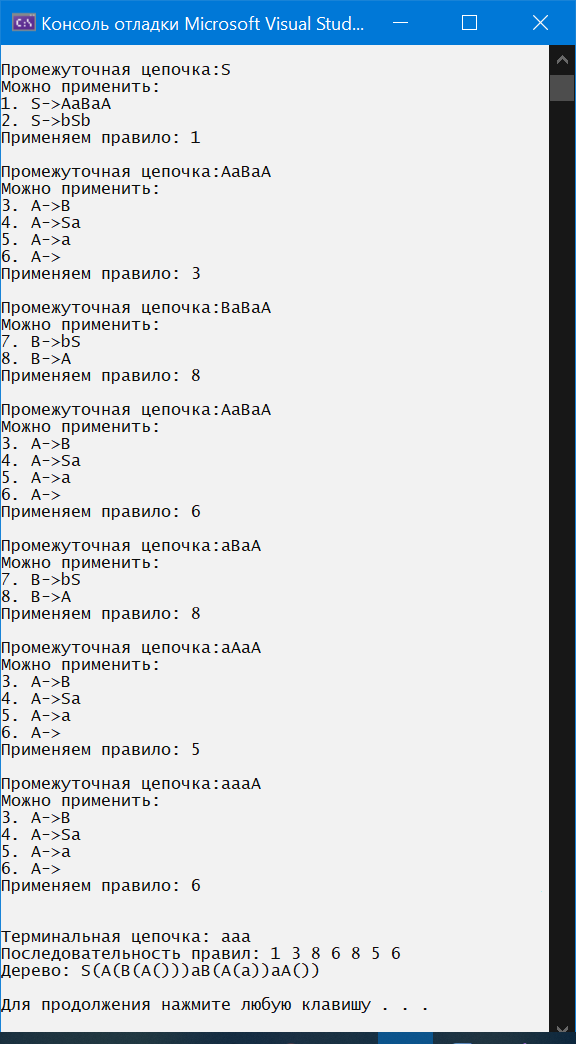
cout << "Применяем правило: ";

int r;

cin >> r;

r--;

cout << endl;

 nums.push\_back(r);

LSF = LSF + '(' + gram[r].output + ')';

chain = applyLeft(chain, r);

break;

}

}

return chain;

}

vector<int> KSGramm::getRules(char t)

{

vector<int> rules;

for (int i = 0; i < gram.size(); i++)

if (gram[i].base == t)

rules.push\_back(i);

return rules;

}

**Задание №3**

#include <iostream>

#include <string>

#include <vector>

#include <string>

using namespace std;

class KSRule

{

public:

char base;

string output;

KSRule(char base, string output)

: base(base), output(output) { }

};

class KSGramm

{

public:

vector<KSRule> gram;

vector<int> nums;

string applyLeft(string chain, int num);

bool checkLeftApply(string chain, vector<int> nums);

private:

vector<int> getRules(char t);

char getLeftNotTerminal(string chain);

};

void show\_vector(vector<int>& a)

{

for (vector<int>::iterator it = a.begin(); it != a.end(); ++it)

cout << \*it << " ";

cout << endl;

}

int main()

{

setlocale(0, "");

string s = "S";

KSGramm gram;

gram.gram.push\_back(KSRule('S', "AaBaA"));

gram.gram.push\_back(KSRule('S', "bSb"));

gram.gram.push\_back(KSRule('A', "B"));

gram.gram.push\_back(KSRule('A', "Sa"));

gram.gram.push\_back(KSRule('A', "a"));

gram.gram.push\_back(KSRule('A', ""));

gram.gram.push\_back(KSRule('B', "bS"));

gram.gram.push\_back(KSRule('B', "A"));

vector<int> rules = { 2, 1, 6, 8, 3, 8, 6, 4, 1, 5, 8, 6, 5 };

vector<int> rules2 = { 1, 0, 5, 7, 2, 7, 5, 3, 0, 4, 7, 5, 4 };

if (gram.checkLeftApply(s, rules2))

{

cout << "Последовательность правил: ";

show\_vector(rules);

cout << "Можно применить для левого вывода в заданной КС-грамматике" << endl << endl;

}

else

{

cout << "Последовательность правил: ";

show\_vector(rules);

cout << "Нельзя применить для левого вывода в заданной КС-грамматике" << endl << endl;

}

system("pause");

return 0;

}

string KSGramm::applyLeft(string chain, int num)

{

for (int i = 0; i < chain.size(); i++)

if (chain[i] == gram[num].base)

{

chain.replace(i, 1, gram[num].output);

break;

}

return chain;

}

bool KSGramm::checkLeftApply(string chain, vector<int> nums)

{

for (auto num : nums)

if (getLeftNotTerminal(chain) == gram[num].base)

chain = applyLeft(chain, num);

else

return false;

return true;

}

vector<int> KSGramm::getRules(char t)

{

vector<int> rules;

for (int i = 0; i < gram.size(); i++)

if (gram[i].base == t)

rules.push\_back(i);

return rules;

}

char KSGramm::getLeftNotTerminal(string chain)

{

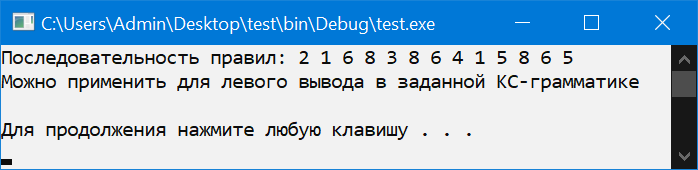
for (auto a : chain)

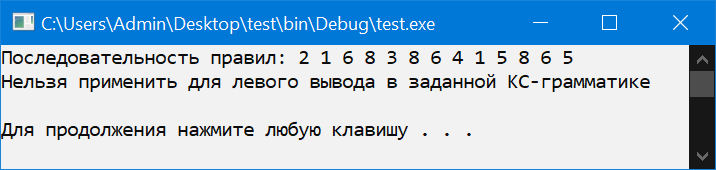
if (getRules(a).size() != 0)

return a;

return ' ';

}





**Задание №5**

#include <iostream>

#include <string>

#include <vector>

#include <string>

using namespace std;

class KSRule

{

public:

char base;

string output;

KSRule(char base, string output)

: base(base), output(output) { }

};

class KSGramm

{

public:

vector<KSRule> gram;

vector<int> nums;

string applyLeft(string chain, int num);

bool checkApply(string chain, vector<int> nums);

};

void show\_vector(vector<int>& a)

{

for (vector<int>::iterator it = a.begin(); it != a.end(); ++it)

cout << \*it << " ";

cout << endl;

}

int main()

{

setlocale(0, "");

string s = "S";

KSGramm gram;

gram.gram.push\_back(KSRule('S', "AaBaA"));

gram.gram.push\_back(KSRule('S', "bSb"));

gram.gram.push\_back(KSRule('A', "B"));

gram.gram.push\_back(KSRule('A', "Sa"));

gram.gram.push\_back(KSRule('A', "a"));

gram.gram.push\_back(KSRule('A', ""));

gram.gram.push\_back(KSRule('B', "bS"));

gram.gram.push\_back(KSRule('B', "A"));

vector<int> rules = { 2, 1, 8, 6, 3, 8, 6, 4, 1, 5, 8, 6, 5 };

vector<int> rules2 = { 1, 0, 7, 5, 2, 7, 5, 3, 0, 4, 7, 5, 4 };

if (gram.checkApply(s, rules2))

{

cout << "Последовательность правил: ";

show\_vector(rules);

cout << "Можно применить в заданной КС-грамматике" << endl << endl;

}

else

{

cout << "Последовательность правил: ";

show\_vector(rules);

cout << "Нельзя применить в заданной КС-грамматике" << endl << endl;

}

system("pause");

return 0;

}

string KSGramm::applyLeft(string chain, int num)

{

for (int i = 0; i < chain.size(); i++)

if (chain[i] == gram[num].base)

{

chain.replace(i, 1, gram[num].output);

break;

}

return chain;

}

bool KSGramm::checkApply(string chain, vector<int> nums)

{

char c;

int pos, cnt = 0;

for (auto num : nums)

{

c = gram[num].base;

pos = chain.find(c);

cout << c << " [pos: " << pos+1 << "] " << chain << endl;

if (pos == -1)

return false;

chain = applyLeft(chain, num);

cout << "chain " << chain << endl << endl;

cnt++;

}

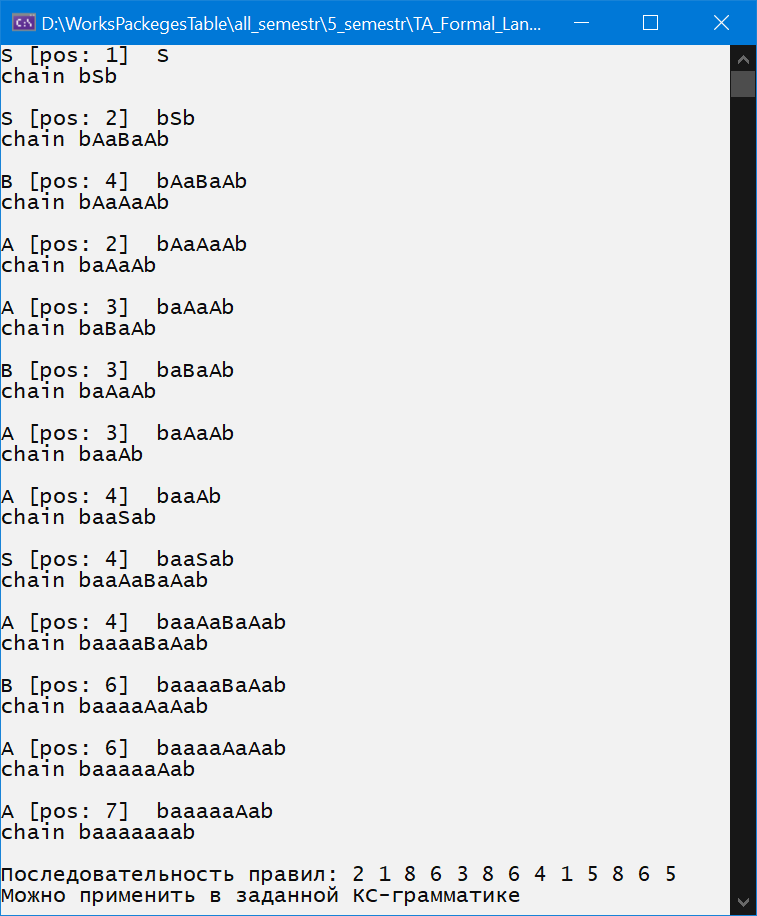
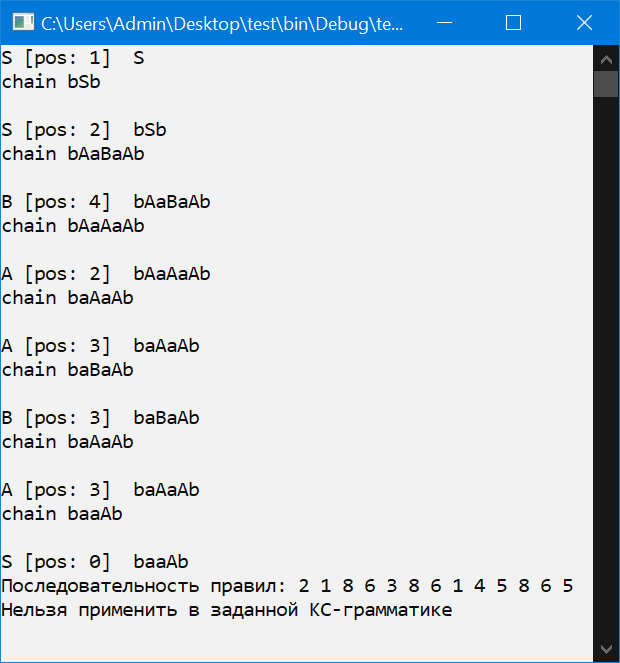
if (cnt == nums.size())

return true;

else

return false;

}



**Вывод**: были изучены основные понятия теории формальных языков и грамматик. Приобретены навыки программирования алгоритмов составления терминальных цепочек и составления линейно-скобочной формы вывода деревьев.