

1	2

Вариант 1

1. Дан класс *Scanner*, моделирующий автомат $\mathcal{U}$ с множеством **состояний** $\{A, B\}$. Метод *accept()* допускает цепочки языка $L(\mathcal{U})$.

```
#include<iostream>
#include<typeinfo>

using namespace std;

class Scanner {
    class State { public:
        virtual void operator()(char c)const =0;
    };

    class A: public State { public:
        void operator()(char c) const {
            if (cin.eof()) throw true; //допуск
            if (c=='b') throw stB;
            else if (c=='a') throw stA;
            else throw false;
        }
    };

    class B: public State { public:
        void operator()(char c) const {
            if (cin.eof()) throw false;
            if (c=='a') throw stA;
            else throw false;
        }
    };

    static A stA; // состояние A
    static B stB; // состояние B

    void next(State &s){//текущее состояние в s
        try { char c=cin.get();
            // текущий символ в c

            s(c); // переход из s по c
        }
        catch (A&) { next(stA); }
        catch (B&) { next(stB); }
    }
public:
    bool accept(){ try{ next(stA);}
        catch(bool b) {return b;}
        return false;
    }
}; //end of Scanner

Scanner::A Scanner::stA;
Scanner::B Scanner::stB;
```

а) Опишите функцию *main()*, анализирующую входную цепочку с помощью *Scanner*. *Ответ:*

б) Постройте конечный автомат $\mathcal{U}$ в виде ДС, указав начальное и заключительные состояния. *Ответ:*

в) Определите язык $L(\mathcal{U})$ формулой или словесно. *Ответ:*

г) Постройте эквивалентную автомату $\mathcal{U}$ левوليнейную грамматику G_{left} . *Ответ:*

д) Каким недостатком обладает реализация автомата с помощью класса *Scanner* с точки зрения прагматики использования механизма исключений? *Ответ:*

е) Добавьте в класс *Scanner* действия по печати названий состояний автомата, проходимых последовательно при допуске цепочки (полагая что метод *name()* из *typeinfo* возвращает имя соответствующего класса в виде строки).

2. По КС-грамматике с **одним** нетерминальным символом S построен анализатор, действующий по принципу рекурсивного спуска. Ниже дана его заготовка с одной пропущенной функцией.

```
#include <iostream>

void S();

struct alt1{ std::string s = "aSb"; };

namespace NS { struct alt2 {
                    std::string s = "cc";
                };
}

char c; // текущий символ-лексема
void gc() { std::cin>>c; }

//функция alt_analyze:

void S() {
    alt1 a1; NS::alt2 a2;
    if (c == 'a')
        alt_analyze (a1);
    else if (c == 'c')
        alt_analyze <NS::alt2>(a2);
}

int main() {
    try { gc(); S();
        if (std::cin.eof())
            std::cout << "\nOK!\n";
        else throw c;
    }
    catch (char) { std::cout << "\nNO!\n";}
}
```

а) Восстановить КС-грамматику G , по которой был построен анализатор.

Ответ:

б) Какой язык порождает грамматика G ?

Ответ: $L(G) =$

в) Описать недостающую функцию *alt_analyze* в соответствии с принципом рекурсивного спуска.

г) Корректен ли получившийся анализатор? Обоснуйте ответ.

Ответ:

д) Если грамматика G не является неукорачивающей, то с помощью алгоритма устранения пустых правых частей привести ее к неукорачивающему виду.

Ответ:

1	2

Вариант 2

1. Дан класс *Scanner*, моделирующий автомат $\mathcal{U}$ с множеством состояний $\{B, C\}$. Метод *accept()* допускает цепочки языка $L(\mathcal{U})$.

```
#include<iostream>
#include<typeinfo>

using namespace std;

class Scanner {
    class State { public:
        virtual void operator()(char c) const =0;
    };

    class B: public State { public:
        void operator()(char c) const {
            if (cin.eof()) throw true; //допуск
            if (c=='c') throw &stC;
            else if (c=='b') throw &stB;
            else throw false;
        }
    };

    class C: public State { public:
        void operator()(char c) const {
            if (cin.eof()) throw false;
            if (c=='b') throw &stB;
            else throw false;
        }
    };

    static B stB; // состояние B
    static C stC; // состояние C

    void next(State *s){
        // s указывает текущее состояние
        try{ char c=cin.get();
            // текущий символ в c

            (*s)(c); // переход из s по c
        }
        catch (B* b) { s=b; }
        catch (C* c) { s=c; }
        next(s);
    }

public:
    bool accept(){ try{ next(&stB);}
        catch(bool b) {return b;}
        return false;
    }
}; //конец класса Scanner

Scanner::B Scanner::stB;
Scanner::C Scanner::stC;
```

а) Опишите функцию *main()*, анализирующую входную цепочку с помощью *Scanner*. *Ответ:*

б) Постройте конечный автомат $\mathcal{U}$ в виде ДС, указав начальное и заключительные состояния. *Ответ:*

в) Определите язык $L(\mathcal{U})$ формулой или словесно. *Ответ:*

г) Постройте эквивалентную автомату $\mathcal{U}$ праволинейную грамматику G_{right} . *Ответ:*

д) Каким недостатком обладает реализация автомата с помощью класса *Scanner* с точки зрения прагматики использования механизма исключений? *Ответ:*

е) Добавьте в класс *Scanner* действия по печати названий состояний автомата, проходимых последовательно при допуске цепочки (полагая что метод *name()* из *typeinfo* возвращает имя соответствующего класса в виде строки).

2. По КС-грамматике с **одним** нетерминальным символом S построен анализатор, действующий по принципу рекурсивного спуска. Ниже дана его заготовка с одной пропущенной функцией.

```
#include <iostream>

void S();

struct alt1{ std::string s = "aSc"; };

namespace NS { struct alt2 {
                  std::string s = "cc";
                };
}

char c; // текущий символ-лексема
void gc() { std::cin>>c; }

//функция alt_analyze:

void S() {
    alt1 a1; NS::alt2 a2;
    if (c == 'a')
        alt_analyze (a1);
    else if (c == 'c')
        alt_analyze <NS::alt2>(a2);
}

int main() {
    try { gc(); S();
        if (std::cin.eof())
            std::cout << "\nOK!\n";
        else throw c;
    }
    catch (char) { std::cout << "\nNO!\n";}
}
```

- а) Восстановить КС-грамматику G , по которой был построен анализатор.

Ответ:

- б) Какой язык порождает грамматика G ?

Ответ: $L(G) =$

- в) Описать недостающую функцию *alt_analyze* в соответствии с принципом рекурсивного спуска.

- е) Корректен ли получившийся анализатор? Обоснуйте ответ.

Ответ:

- г) Если грамматика G не является неукорачивающей, то с помощью алгоритма устранения пустых правых частей привести ее к неукорачивающему виду.

Ответ: