

## Лексический анализ

### Разбор лабораторной работы 12

1. Задание лабораторной работы 12.

2. Реализация конструкторов.

Конструктор – метод (функция) структуры, имя которого совпадает с именем структуры (типа); метод ничего не возвращает; метод вызывается автоматически сразу после выделения памяти (в стеке или в куче) и предназначен для инициализации памяти.

Конструкторы бывают: по умолчанию, с параметрами, копирующими.

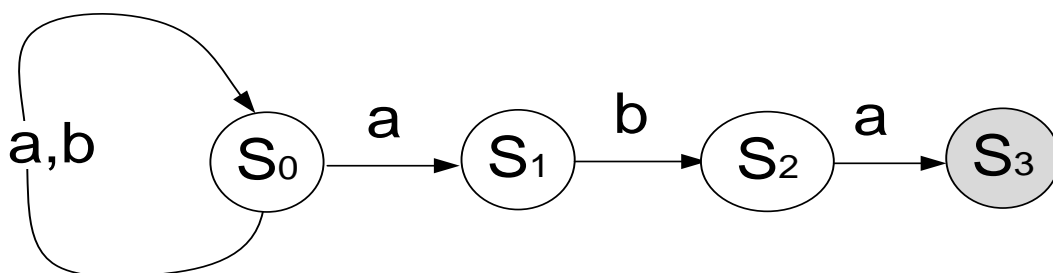
3. Реализация конструктора структуры **RELATION**

```

struct RELATION      // ребро:символ -> вершина графа переходов КА
{
    char symbol;      // символ перехода
    short nnode;      // номер смежной вершины
    RELATION (
        char c = 0x00,    // символ перехода
        short ns = NULL   // новое состояние
    );
};

RELATION::RELATION (char c, short nn)
{
    symbol = c;
    nnode = nn;
};

FST::FST fst1(
    "aaabbbaba",        // недетерминированный конечный автомат (a+b)*aba
                        // цепочка для распознавания
    4,                  // количество состояний
    FST::NODE(3, FST::RELATION('a', 0), FST::RELATION('b', 0), FST::RELATION('a', 1)),
    FST::NODE(1, FST::RELATION('b', 2)),
    FST::NODE(1, FST::RELATION('a', 3)),
    FST::NODE()
);
  
```



**Графом переходов** конечного автомата  $M = (S, I, \delta, s_0, F)$  называется ориентированный граф  $G = (S, E)$ , где  $S$  – множество вершин графа совпадает с множеством состояний конечного автомата,

$E$  – множество ребер (направленных линий, соединяющих вершины), ребро  $(s_i, s_j) \in E$ , если  $s_j \in \delta(s_i, a), a \in I \cup \lambda$ .

Метка ребра  $(s_i, s_j)$  – все  $a$ , для которых  $s_j \in \delta(s_i, a)$ .

Ребро графа, концы которого совпадают, то есть  $(s_i, s_i)$ , называется **петлей**

Два ребра, имеющие общую концевую вершину, то есть  $(s_i, s_k)$  и  $(s_k, s_j)$ , называются **смежными**.

Если имеется ребро  $(s_i, s_j) \in E$ , то говорят:

- $s_i$  – **предок**  $s_j$ .
- $s_i$  и  $s_j$  – **смежные**.
- вершина  $s_i$  **инцидентна** ребру  $(s_i, s_j)$ .
- вершина  $s_j$  **инцидентна** ребру  $(s_i, s_j)$ .

**Инцидентность** – понятие, используемое только в отношении ребра и вершины.

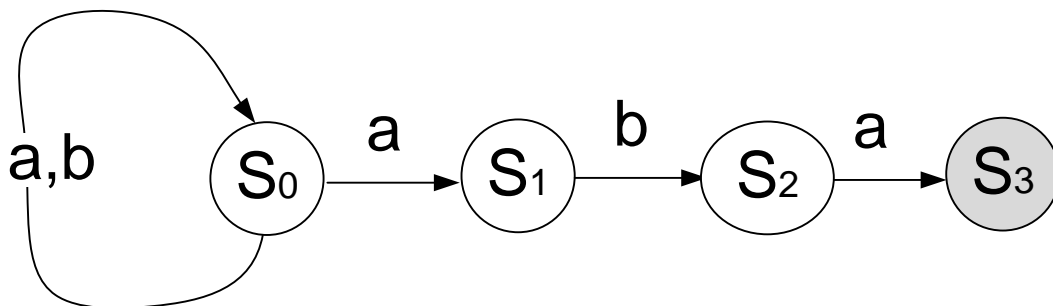
#### 4. Реализация конструктора структуры **NODE**

```
struct NODE // вершина графа переходов
{
    short n_relation; // количество инцидентных ребер
    RELATION *relations; // инцидентные ребра
    NODE();
    NODE (
        short n, // количество инцидентных ребер
        RELATION rel, ... // список ребер
    );
};

NODE::NODE() // по умолчанию
{
    n_relation = 0;
    RELATION *relations = NULL;
};

NODE::NODE (short n, RELATION rel, ...) // с параметрами
{
    n_relation = n;
    RELATION *p = &rel;
    relations = new RELATION[n];
    for(short i = 0; i < n; i++) relations[i] = p[i];
};

FST::FST fst1( // недетерминированный конечный автомат (a+b)*aba
    "aaabbbaba", // цепочка для распознавания
    4, // количество состояний
    FST::NODE(3, FST::RELATION('a', 0), FST::RELATION('b', 0), FST::RELATION('a', 1)),
    FST::NODE(1, FST::RELATION('b', 2)),
    FST::NODE(1, FST::RELATION('a', 3)),
    FST::NODE()
);
```

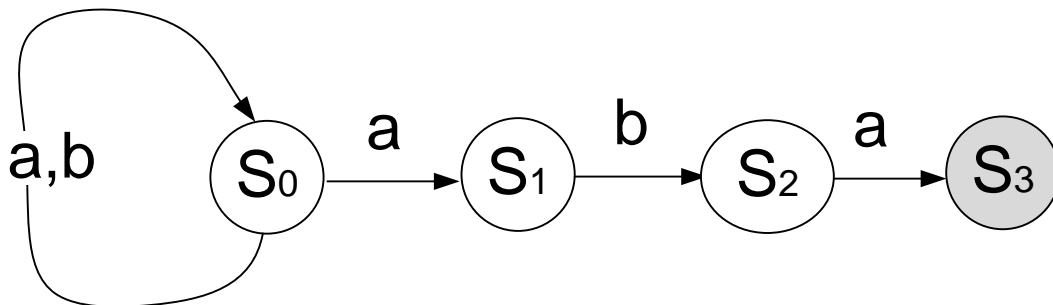


## 5. Реализация конструктора структуры FST

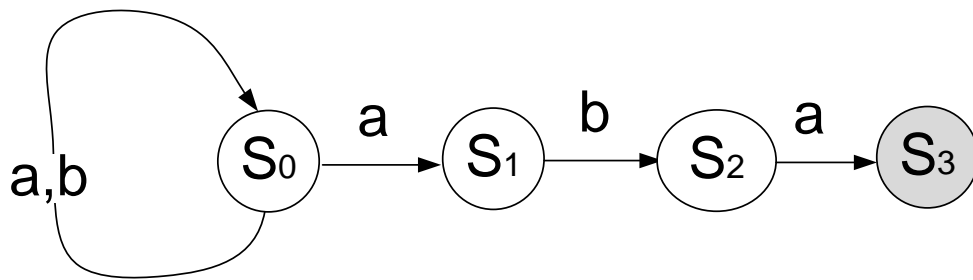
```
struct FST    // недетерминированный конечный автомат
{
    char* string;    // цепочка (строка, завершится 0x00 )
    short position;  // текущая позиция в цепочке
    short nstates;   // количество состояний автомата
    NODE* nodes;     // граф переходов: [0] - начальное состояние, [nstate-1] - конечное
    short* rstates;  // возможные состояния автомата на данной позиции
    FST(
        char* s,    // цепочка (строка, завершится 0x00 )
        short ns,   // количество состояний автомата
        NODE n, ... // список состояний (граф переходов)
    );
};
```

```
FST::FST(char* s,    short ns,    NODE n, ...)
{
    string = s;
    nstates = ns;
    nodes = new NODE[ns];
    NODE *p = &n;
    for (int k = 0; k < ns; k++) nodes[k] = p[k];
    rstates = new short[nstates];
    memset(rstates, 0xff, sizeof(short)*nstates);
    rstates[0] = 0;
    position = -1;
};
```

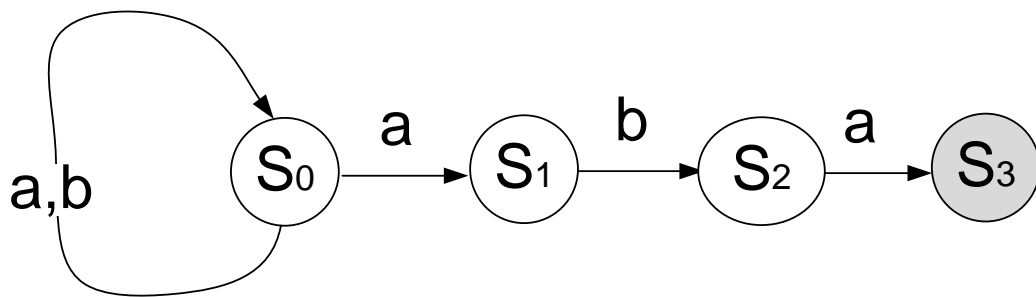
```
FST::FST fst1(                // недетерминированный конечный автомат (a+b)*aba
    "aaabbbaba",              // цепочка для распознавания
    4,                        // количество состояний
    FST::NODE(3, FST::RELATION('a', 0), FST::RELATION('b', 0), FST::RELATION('a',1)),
    FST::NODE(1, FST::RELATION('b', 2)),
    FST::NODE(1, FST::RELATION('a', 3)),
    FST::NODE()
);
```



## 6. Алгоритм разбора



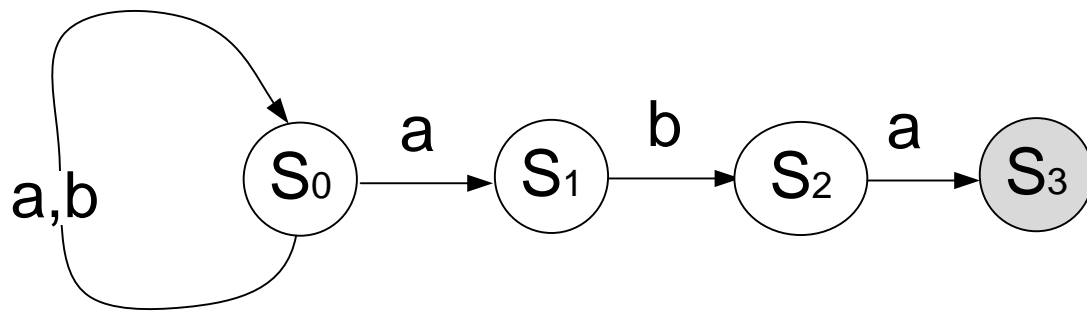
aabbabaaba	{S0}	
abbabaaba	{S0,S1}	
bbabaaba	{S0,S1}	
babaaba	{S0,S2}	
abaaba	{S0}	
baaba	{S0,S1}	
aaba	{S0,S2}	
aba	{S0,S1,S3}	
ba	{S0,S1}	
a	{S0,S2}	
	{S0, <b>S3</b> }	<b>Успешный разбор</b>



aabbababba	{S <sub>0</sub> }
abbababba	{S <sub>0</sub> , S <sub>1</sub> }
bbababba	{S <sub>0</sub> , S <sub>1</sub> }
bababba	{S <sub>0</sub> , S <sub>2</sub> }
ababba	{S <sub>0</sub> }
babba	{S <sub>0</sub> , S <sub>1</sub> }
abba	{S <sub>0</sub> , S <sub>2</sub> }
bba	{S <sub>0</sub> , S <sub>1</sub> , S <sub>3</sub> }
ba	{S <sub>0</sub> , S <sub>2</sub> }
a	{S <sub>0</sub> }

{S<sub>0</sub>, S<sub>1</sub>}

Ошибка разбора



aabbxbabba

$\{S_0\}$

abbxbabba

$\{S_0, S_1\}$

bbxbabba

$\{S_0, S_1\}$

bxbabba

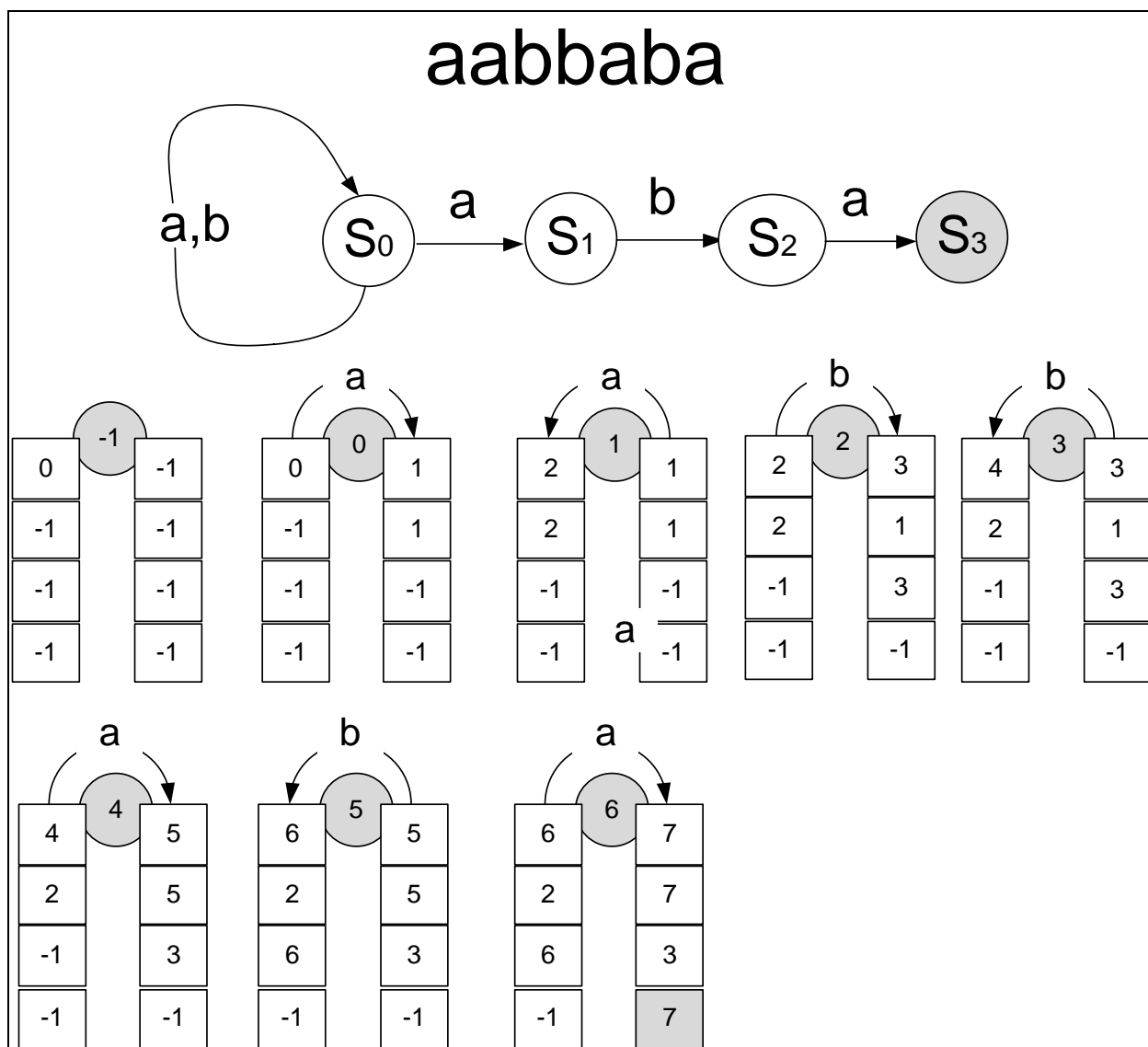
$\{S_0, S_2\}$

xbabba

$\{\}$

**Ошибка разбора**

## 7. Реализация алгоритма разбора





## 8. Реализация функции **execute**

```
bool execute(          // выполнить распознавание цепочки
    FST& fst           // недетерминированный конечный автомат
);

bool step(FST& fst, short* &rstates) // один шаг автомата
{
    bool rc = false;
    std::swap(rstates, fst.rstates); // смена массивов
    for(short i = 0; i < fst.nstates; i++)
    {
        if (rstates[i] == fst.position)
            for(short j = 0; j < fst.nodes[i].n_relation; j++)
            {
                if (fst.nodes[i].relations[j].symbol == fst.string[fst.position])
                {
                    fst.rstates[fst.nodes[i].relations[j].nnode] = fst.position+1;
                    rc = true;
                }
            };
    };
    return rc;
};

bool execute(FST& fst) // выполнить распознавание цепочки
{
    short* rstates = new short[fst.nstates]; memset(rstates, 0xff, sizeof(short)*fst.nstates);
    short lstring = strlen(fst.string);
    bool rc = true;
    for (short i = 0; i < lstring && rc; i++)
    {
        fst.position++; // продинули позицию
        rc = step(fst, rstates); // один шаг автомата
    };
    delete[] rstates;
    return (rc?(fst.rstates[fst.nstates-1] == lstring):rc);
};

if (FST::execute(fst2)) // выполнить разбор
    std::cout<<"Цепочка "<< fst2.string << " распознана"<< std::endl;
else
    std::cout<<"Цепочка "<< fst2.string << " не распознана"<< std::endl;
```