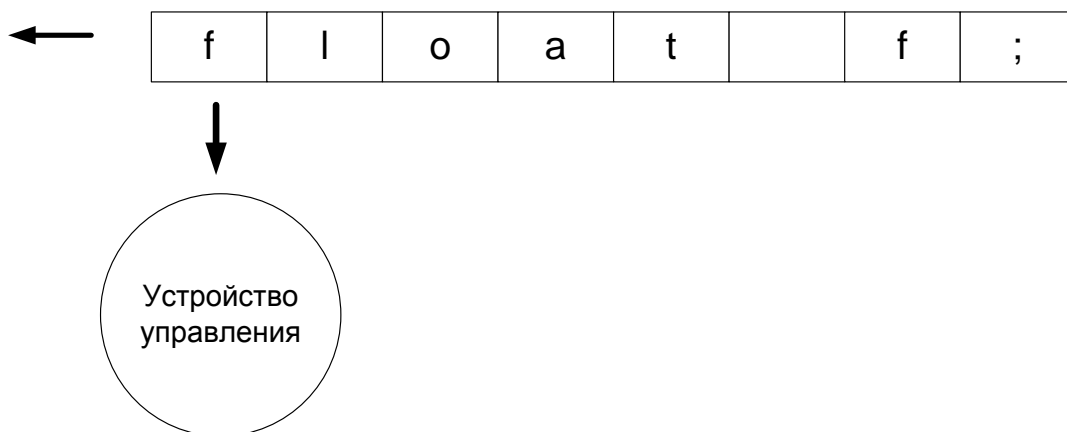


Лексический анализатор
Разбор лабораторной работы 16

1. Используйте материалы лекций 9-12.
2. Используйте результаты лабораторных работ 14-15.
3. Ознакомьтесь с заданием лабораторной работы 16.

Введение. С точки зрения лексического анализатора язык программирования – это набор лексем (токенов), которые распознаются (классифицируются) лексическим анализатором по шаблонам, описывающим вид соответствующих лексем. Множество слов, или цепочек символов, которые соответствуют шаблону, называют языком. Для описания шаблонов используются регулярные выражения. Язык программирования (на уровне лексического анализа) представляет собой регулярный язык, заданный регулярным выражением (язык типа 3 иерархии Хомского).

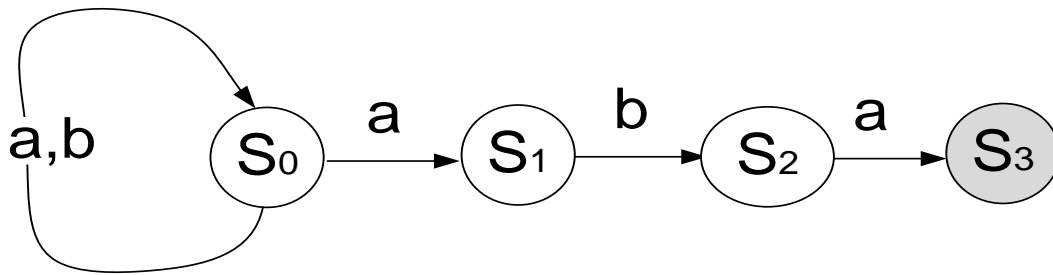
Распознаватель лексики языка – это алгоритм, позволяющий определить некоторое множество (входной язык). Простейший распознаватель состоит из входной ленты (входная цепочка символов) и управляющего устройства с конечной памятью:



Класс алгоритмов, соответствующих приведенной схеме, может быть записан в форме конечного автомата (КА).

Пусть входная цепочка описывается регулярным выражением $w \in (a + b)^* aba$

Граф конечного автомата, соответствующий этому регулярному выражению:



Определения:

Графом переходов конечного автомата $M = (S, I, \delta, s_0, F)$ называется ориентированный граф $G = (S, E)$,

где S – множество вершин графа, которое совпадает со 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) – это все a , для которых $s_j \in \delta(s_i, a)$.

- ✓ Ребро графа называется **петлей**, если его концы совпадают, то есть ребро (s_i, s_i) является петлей.
- ✓ Два ребра называются **смежными**, если они имеют общую конечную вершину, то есть (s_i, s_k) и (s_k, s_j) имеют общую конечную вершину s_k .
- ✓ 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*.

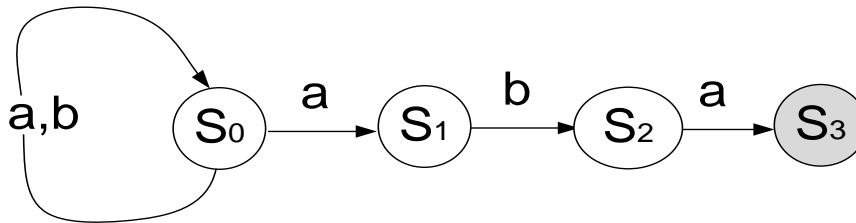
Определение:

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

Разновидности конструкторов: по умолчанию, с параметрами, копирующие конструкторы.

Распознавание допустимости входной цепочки. Реализация алгоритма с двумя массивами (см. лекцию 12).

4. Реализация конструктора структуры **RELATION**

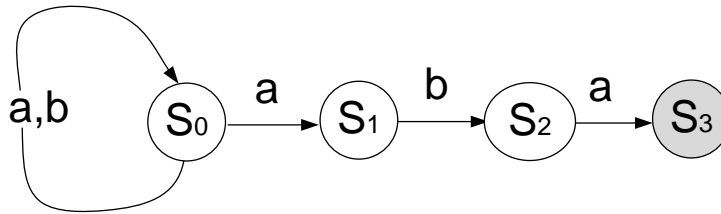


```
struct RELATION      // ребро:символ -> вершина графа переходов КА
{
    char symbol;      // символ перехода
    short nnode;      // номер смежной вершины
    RELATION (
        char c = 0x00,    // символ перехода
        short ns = NULL   // новое состояние
    );
};

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

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

5. 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
    );
};

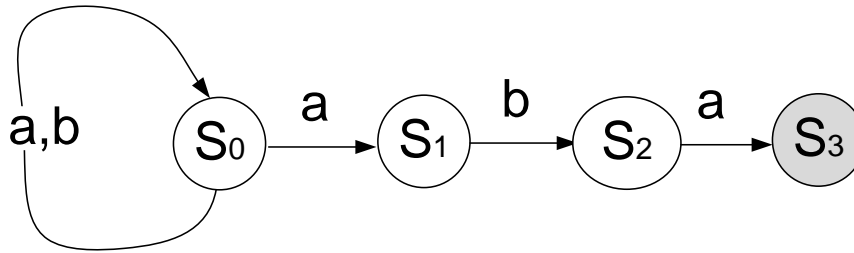
NODE::NODE() // по умолчанию
{
    n_relation = 0;
    RELATION *relations = NULL;
};

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()
);

```

6. 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)
    );
};
  
```

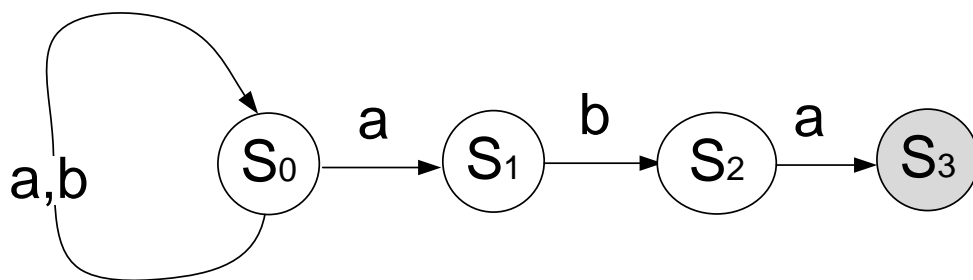
```

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;
};
  
```

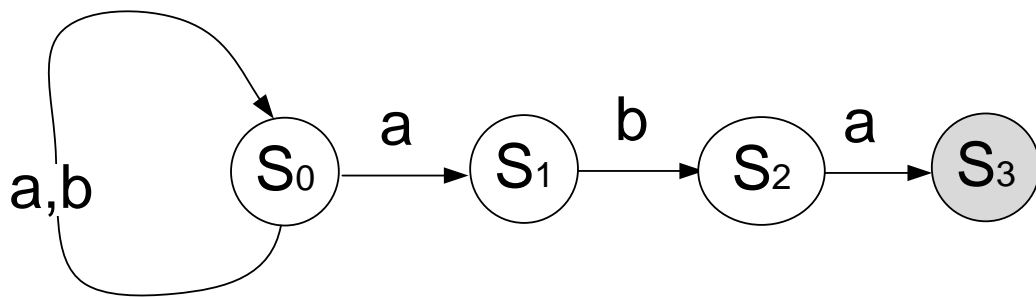
```

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()
);
  
```

7. Алгоритм разбора цепочек недерминированным конечным автоматом
 $(a + b)^*aba$



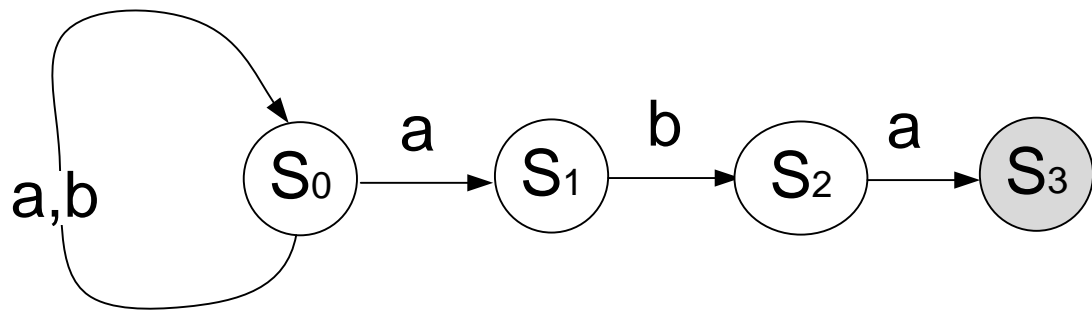
aabbabaaba	{S0}
abbabaaba	{S0,S1}
bbabaaba	{S0,S1}
babaaba	{S0,S2}
abaaba	{S0}
baaba	{S0,S1}
aaba	{S0,S2}
aba	{S0,S1,S3}
ba	{S0,S1}
a	{S0,S2}
	{S0, S3 Успешный разбор }



aabbababba	{S0}
abbababba	{S0,S1}
bbababba	{S0,S1}
bababba	{S0,S2}
ababba	{S0}
babba	{S0,S1}
abba	{S0,S2}
bba	{S0,S1,S3}
ba	{S0,S2}
a	{S0}

{S0,S1}

Ошибка разбора



aabbxbabba

$\{S_0\}$

abbxbabba

$\{S_0, S_1\}$

bbxbabba

$\{S_0, S_1\}$

bxbabba

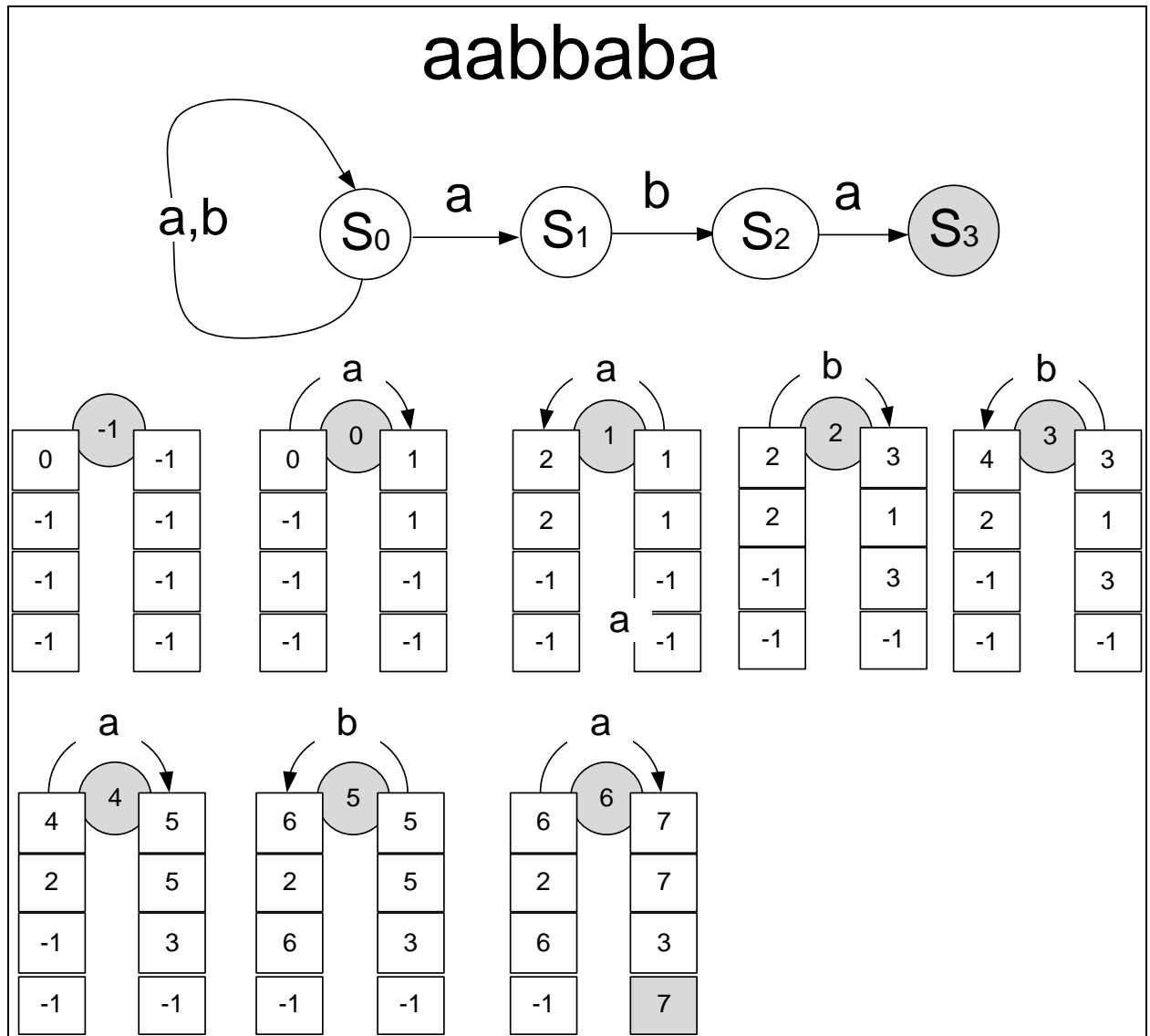
$\{S_0, S_2\}$

xbabba

$\{\}$

Ошибка разбора

8. Алгоритм разбора на двух массивах



Инициализация `rstates[nstates]` – массива возможных состояний автомата в данной позиции, выполняется конструктором с параметрами `FST::FST(char *s, short ns, NODE n, ...); :`

```
rstates = new short[nstates];
memset(rstates, 0xFF, sizeof(short)*nstates);
rstates[0] = 0;
position = -1; // чтобы с 0 начинать в execute
```

Элементы вспомогательного массива заполняются значениями -1.

На каждом шаге `position` меняем массивы местами.

Если на данном шаге существует переход по метке ребра из состояния i в состояние j , то в j -ый элемент массива `FST::FST.rstates` заносим значение `position + 1`.

Если на заключительном шаге в последнем элементе массива получено значение, равное длине цепочки, то её разбор закончился успешно.

9. Реализация функции **execute** (распознавание цепочки символов)

```
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)) // выолнить разбор
    std::cout<<"Цепочка "<< fst2.string << " распознана"<< std::endl;
else std::cout<<"Цепочка "<< fst2.string << " не распознана"<< std::endl;
```