

Объектная модель детерминированного конечного автомата.

При конструировании лексического анализатора вы будете использовать модель детерминированного конечного автомата, представленную классом `tFSM`. Класс определен в файле `fsm.h`.

```
class tFSM{  
public:  
// типы  
    typedef char tSymbol;  
    typedef unsigned char tState;  
    typedef std::set<tState> tStateSet;  
// конструктор  
    tFSM(){};
```

Конструктор создает "пустой" автомат, в котором нет ни одного состояния.

```
// функции-члены
```

```
void add(tState from, tSymbol c, tState to);
```

В таблицу переходов автомата добавляется одна команда (from,c)->to, при этом таблица автоматически расширяется так, чтобы вместить и состояние from, и состояние to. Для полного определения таблицы, нужно записать в программе столько вызовов функции add, сколько есть команд перехода. Порядок задания команд значения не имеет. Две команды вида (i,a)->j и (i,b)->j задать невозможно, в таблицу попадет только одна. Модель обеспечивает детерминированность автомата.

```
void final(tState st);
```

Объявляет состояние st заключительным.

```
int apply(const tSymbol* input);
```

Применяет автомат ко входной цепочке input, заданной в форме C-строки.

Функция подсчитывает количество символов, которое автомат прочитал до остановки. Если автомат остановился в заключительном состоянии, то выдается это количество символов, в противном случае - 0. Таким образом, функция позволяет выделить во входной

цепочке самый длинный допустимый префикс. Вся цепочка принадлежит языку автомата, если ее длина совпадает со значением функции apply.

```
size_t size()const{return table.size();}
```

Выдает количество состояний автомата.

```
private:
```

```
// представление детерминированного конечного  
// автомата
```

```
typedef std::map<tSymbol,tState> tTransMap;
```

```
typedef std::vector<tTransMap> tStateTable;
```

```
tStateTable table; //таблица состояний
```

```
tStateSet finals; //множество заключительных  
// состояний
```

```
};
```

```
// функции-помощники
```

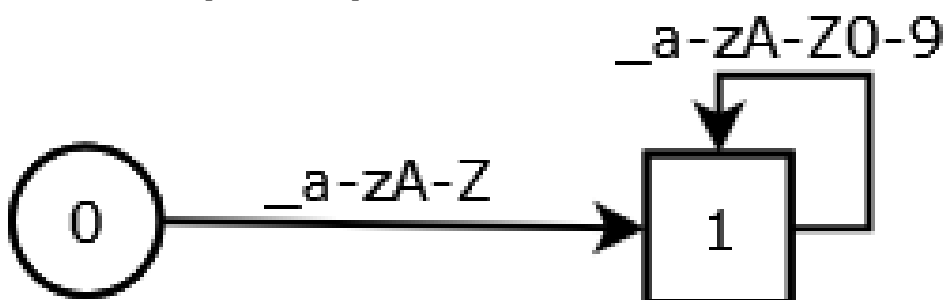
```
void addstr(tFSM& fsm,  
           tFSM::tState from, const tFSM::tSymbol *str,  
           tFSM::tState to);
```

Добавляет в таблицу набор команд перехода из from в to для каждого символа C-строки str.

```
void addrange(tFSM& fsm,  
             tFSM::tState from, tFSM::tSymbol first,  
             tFSM::tSymbol last, tFSM::tState to);
```

Добавляет в таблицу набор команд перехода из from в to для каждого символа диапазона от first до last (включительно).

Рассмотрим модель автомата, распознающего токен «идентификатор C++» .



При построении диаграммы использовано соглашение о том, что все переходы из состояния i в состояние j

изображаются одной дугой, рядом с которой записаны символы перехода, '-' между двумя символами обозначает диапазон. Если '-' сам является символом перехода, его следует записать первым или последним.

Тестовое приложение строится из файла cppid.cpp .

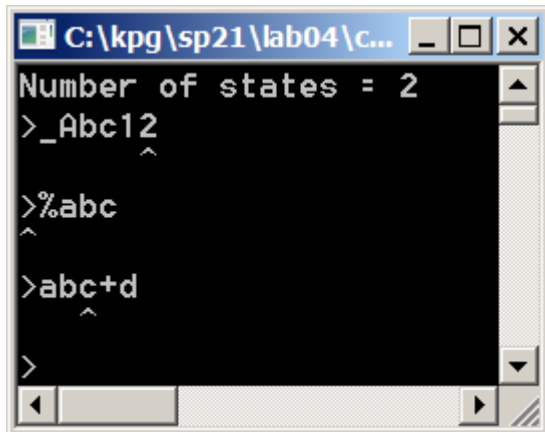
```
//          cppid.cpp
#include <iostream>
#include <iomanip>
#include "fsm.h"
using namespace std;

int main()
{
    tFSM fsm;
    ///////////////////////////////////
    // Постоить автомат
    addstr(fsm,0,"_",1);
    addrange(fsm,0,'A','Z',1);
    addrange(fsm,0,'a','z',1);
    addstr(fsm,1,"_",1);
    addrange(fsm,1,'A','Z',1);
    addrange(fsm,1,'a','z',1);
    addrange(fsm,1,'0','9',1);
    fsm.final(1);
    ///////////////////////////////////

    cout << "Number of states = " << fsm.size()
         << "\n";

    while(true)
    {
        char input[81];
        cout << ">";
        cin.getline(input,81);
        if(!*input) break;
        int res = fsm.apply(input);
        cout << setw(res?res+1:0) << "^"
             << endl;
    }
    return 0;
}
```

Приложение в цикле запрашивает тестовые цепочки и применяет к ним автомат. Результат отображается в форме маркера '^', который отмечает длину допустимой части цепочки, выданную функцией apply. Работа приложения завершается при вводе пустой цепочки.



```
C:\kpg\sp21\lab04\c...
Number of states = 2
>_Abc12
  ^
>%abc
^
>abc+d
  ^
>
```