Разработка лексического анализатора

- Входной язык содержит операторы цикла **for (…; …; …) do …**, разделённые символом **;** (точка с запятой). Операторы цикла содержат идентификаторы, знаки сравнения **<**, **>**, **=**, десятичные числа с плавающей точкой (в обычной и экспоненциальной форме), знак присваивания (**:=**)

for(abc1:=2.;abc1<11.0E+1;abc1:=abc1>.1)do _id:=_id=34e-5;

Lexeme table:

```
KEYWORD: for
DELIMITER: (
IDENTIFIER: abc1
OPERATION: :=
NUMBER: 2.
DELIMITER: ;
IDENTIFIER: abc1
OPERATION: <
NUMBER: 11.0E+1
DELIMITER: ;
IDENTIFIER: abc1
OPERATION: :=
IDENTIFIER: abc1
OPERATION: >
NUMBER: .1
DELIMITER: )
KEYWORD: do
IDENTIFIER: _id
OPERATION: :=
IDENTIFIER: _id
OPERATION: =
NUMBER: 34e-5
```

```
for(abc1:=.;abc1<11.0E+1;abc1:=abc1>.1)do _id:=_id=34e-5;
                    Unknown character: .
                    Lexeme table:
                    KEYWORD: for
                    DELIMITER: (
                    IDENTIFIER: abc1
                    OPERATION: :=
                    DELIMITER: ;
                    IDENTIFIER: abc1
                    OPERATION: <
                    NUMBER: 11.0E+1
                    DELIMITER: ;
                    IDENTIFIER: abc1
                    OPERATION: :=
                    IDENTIFIER: abc1
                    OPERATION: >
                    NUMBER: .1
                    DELIMITER: )
                    KEYWORD: do
                    IDENTIFIER: _id
                    OPERATION: :=
                    IDENTIFIER: _id
                    OPERATION: =
                    NUMBER: 34e-5
```

for(ab%c1:=.;abc1<.E+1;abc1:=abc1>1.1)do _id:=_id=34e-5;

```
Unknown character: %
Unknown character: .
Unknown character: .
Lexeme table:
KEYWORD: for
DELIMITER: (
IDENTIFIER: ab
IDENTIFIER: c1
OPERATION: :=
DELIMITER: ;
IDENTIFIER: abc1
OPERATION: <
IDENTIFIER: E
NUMBER: +1
DELIMITER: ;
IDENTIFIER: abc1
OPERATION: :=
IDENTIFIER: abc1
OPERATION: >
NUMBER: 1.1
DELIMITER: )
KEYWORD: do
IDENTIFIER: _id
OPERATION: :=
IDENTIFIER: _id
OPERATION: =
NUMBER: 34e-5
```

Состояния, действия

Состояния	
H	Начальное состояние
ID	Идентификаторы
NM	Числа
ASGN	Знак присваивания (:=)
DLM	Разделители (;, (,), =, >, <)
ERR	Нераспознанные символы
Действия	
fgetc	чтение символа из файла
is_kword	проверка, является ли идентификатор ключевым словом
is_num	проверка на правильность записи числа
add_token	добавление токена в таблицу лексем

Регулярные выражения

Идентификатор — это последовательность букв и цифр. Первым символом должна быть буква; знак подчеркивания (_) считается буквой. Буквы нижнего и верхнего регистров различаются.

```
(_|a-zA-Z)(_|a-zA-Z|0-9)*
```

Вещественная константа с плавающей точкой состоит из целой части, десятичной точки, дробной части, символа e или E, целого показателя степени с необязательным знаком и необязательного суффикса типа — одной из букв f, F, l или L. Целая и дробная части состоят из последовательностей цифр. Может отсутствовать либо целая, либо дробная часть (но не обе сразу). Тип определяется суффиксом: F или f обозначают тип float, L или l — тип long double. При отсутствии суффикса принимается тип double.

```
(-|+)?((0-9)*.(0-9)+|(0-9)+.|(0-9)+)(e|E)(-|+)?(0-9)+(f|l|F|L)?
```

Диаграмма состояний

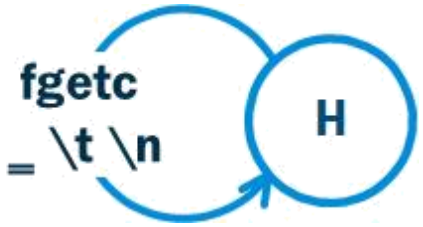


Диаграмма состояний

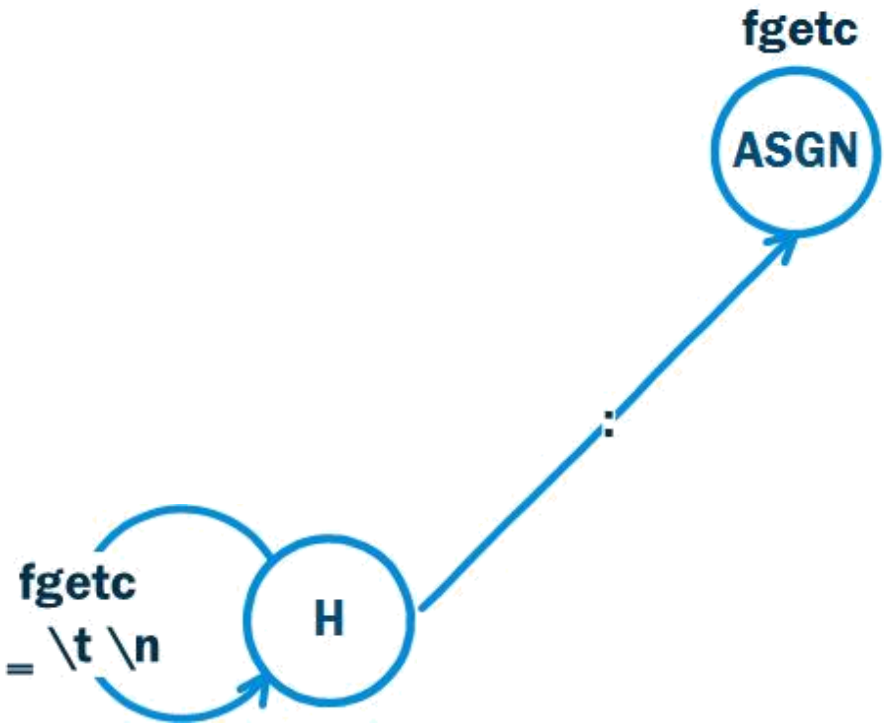


Диаграмма состояний

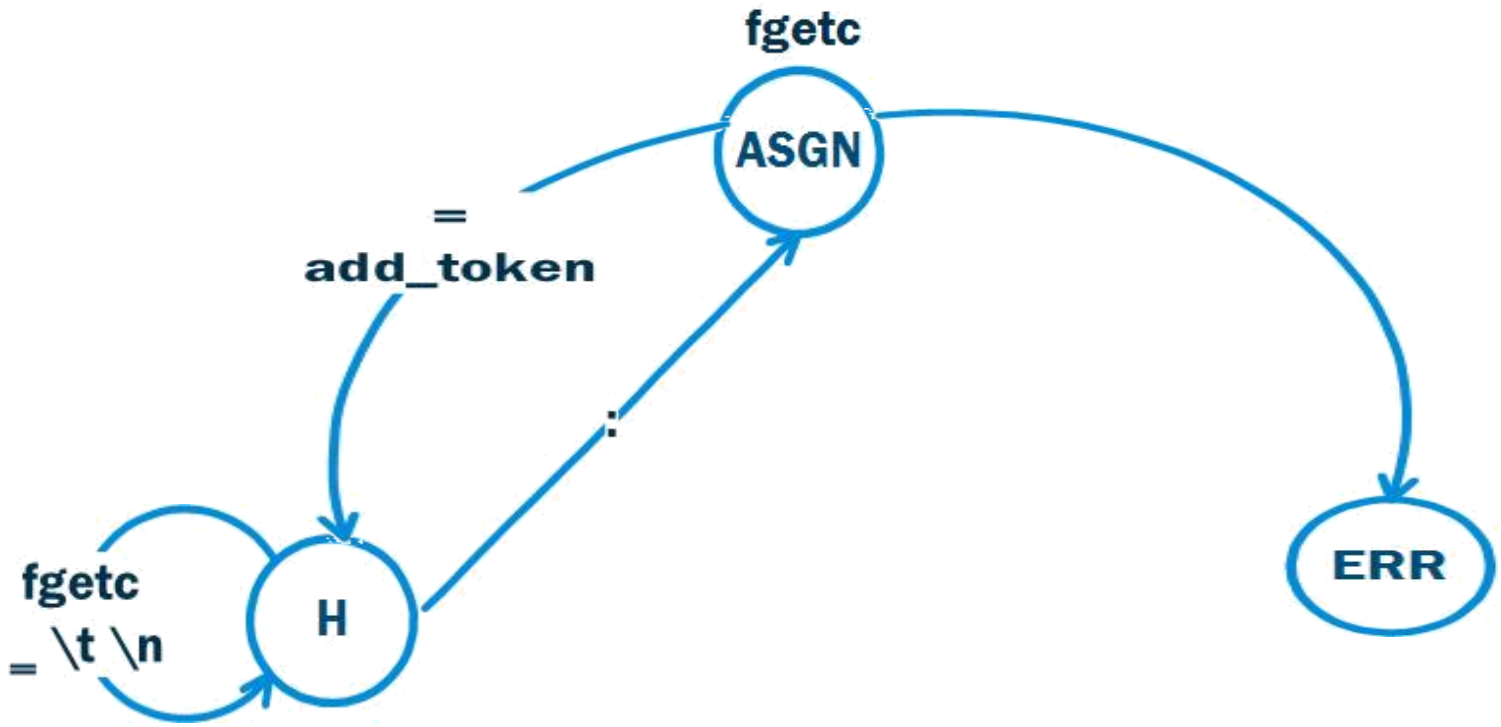


Диаграмма состояний

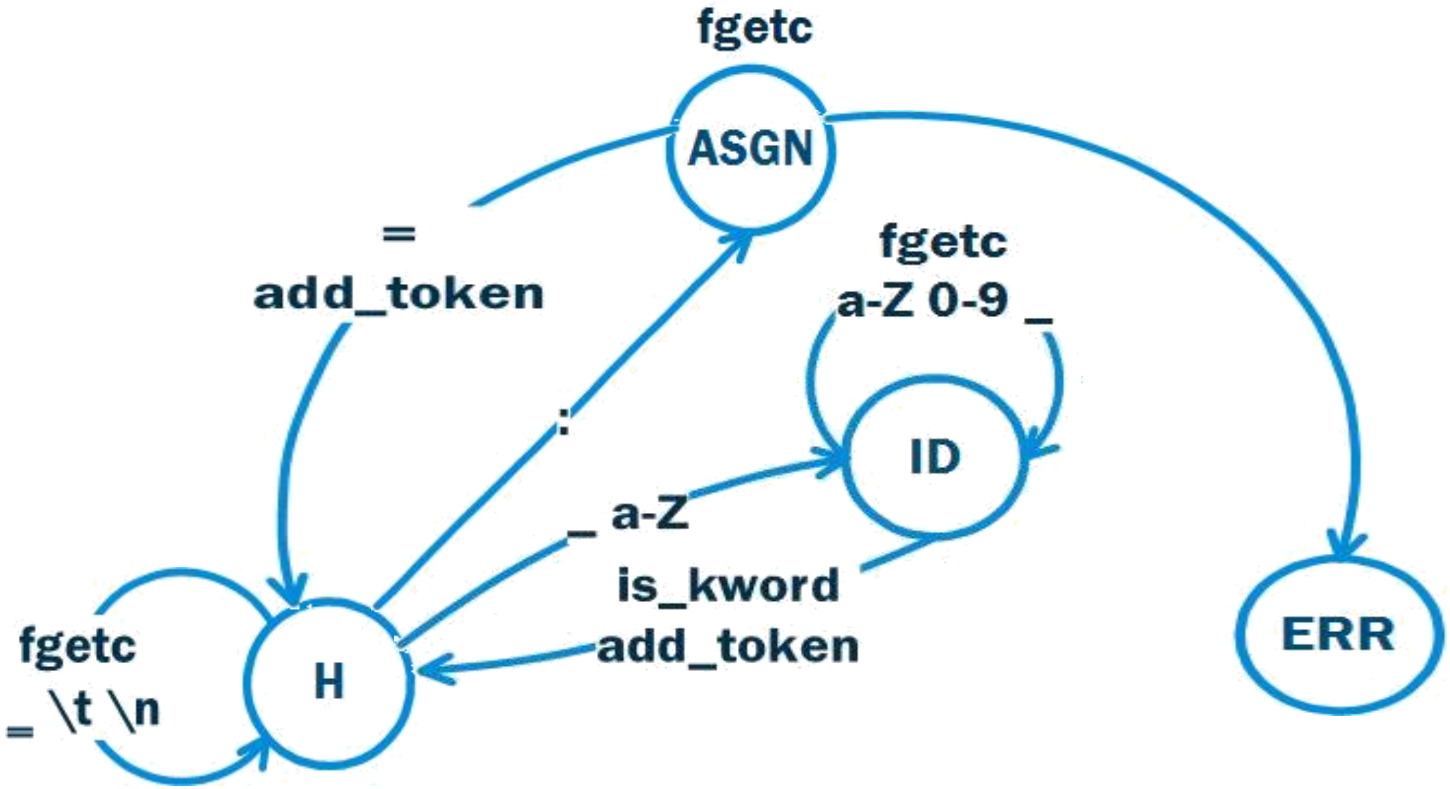


Диаграмма состояний

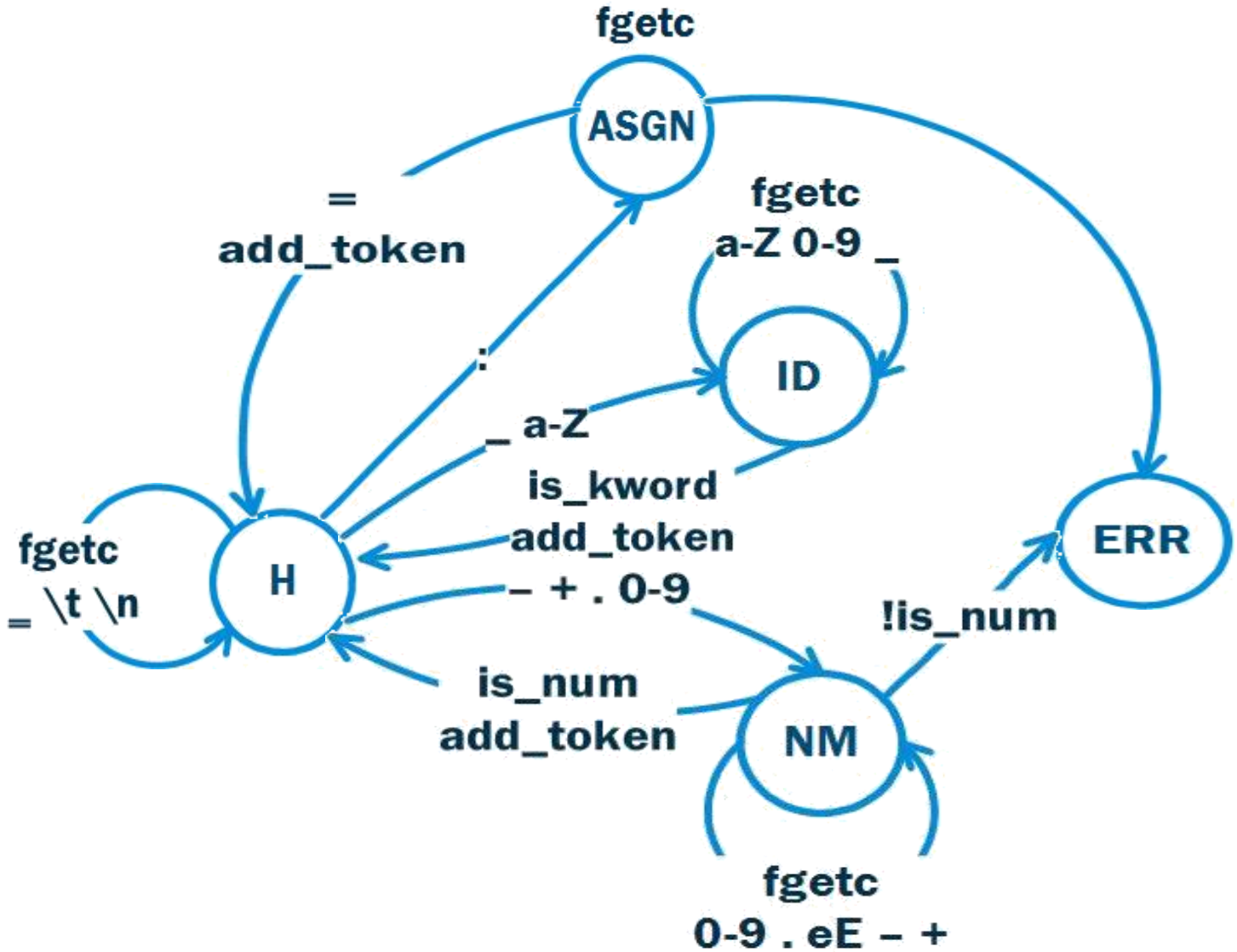
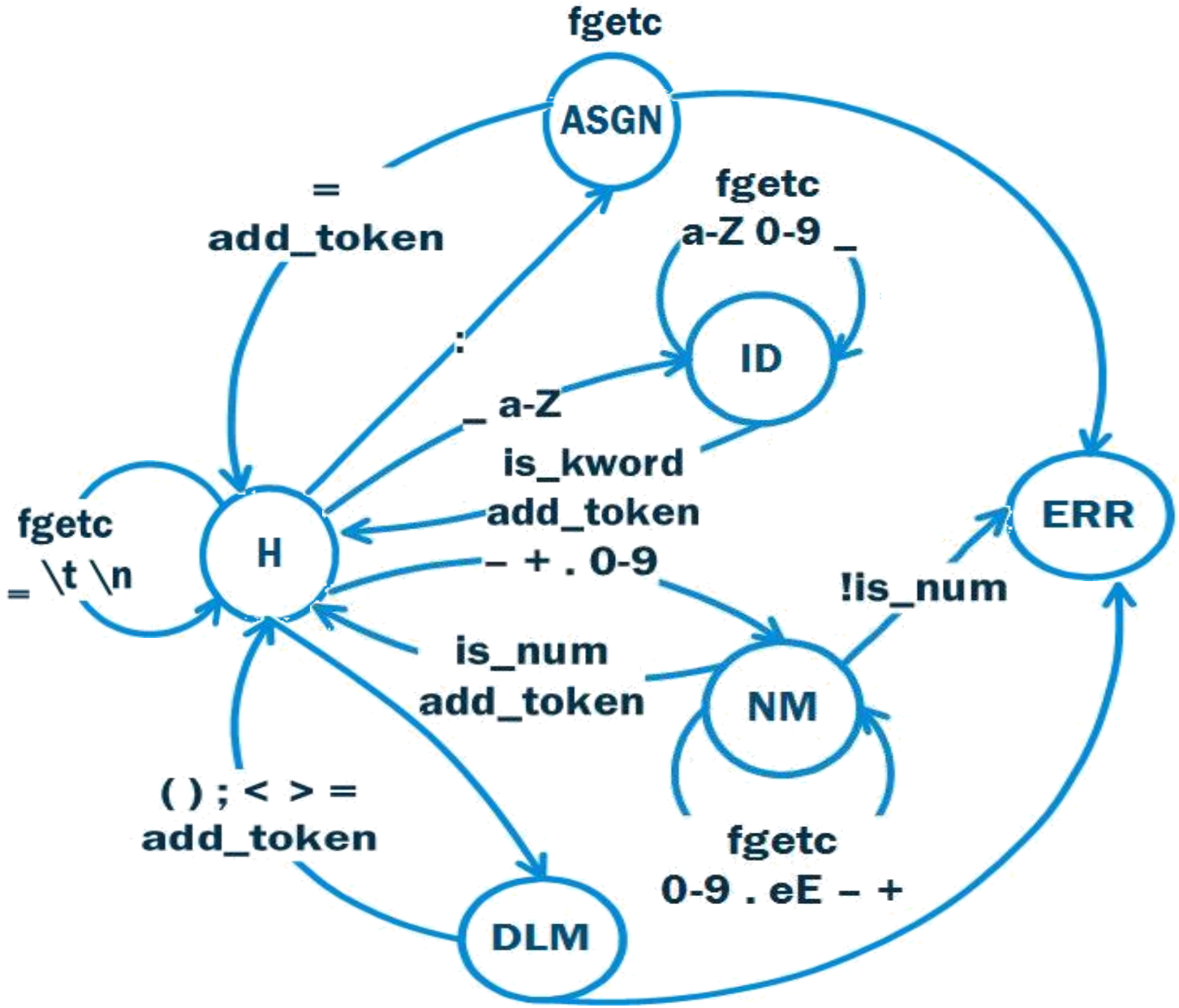


Диаграмма состояний



Программная реализация

```
#define NUM_OF_KWORDS 2

char *keywords[NUM_OF_KWORDS] = {"for", "do"};

enum states {H, ID, NM, ASGN, DLM, ERR};
enum tok_names {KWORD, IDENT, NUM, OPER, DELIM};

struct token
{
    enum tok_names token_name;
    char *token_value;
};

struct lexeme_table
{
    struct token tok;
    struct lexeme_table *next;
};

struct lexeme_table *lt = NULL;

int lexer(char *filename);
int is_keyword(char *id);
int add_token(struct token *tok);
```

Программная реализация (продолжение)

```
int lexer(char *filename)
{
    FILE *fd;
    int c, err_symbol;
    struct token tok;

    if((fd = fopen(filename, "r")) == NULL)
    {
        printf("\nCannot open file %s.\n", filename);
        return -1;
    }

    enum states CS = H;
    c = fgetc(fd);
```

Программная реализация (продолжение)

```
while(!feof(fd)){
    switch(CS){
        case H:
        {
            while((c == ' ') || (c == '\t') || (c == '\n')){
                c = fgetc(fd);
            }
            if(((c >= 'A') && (c <= 'Z')) || ((c >= 'a') && (c <= 'z')) ||
                                                    (c == '_')){
                CS = ID;
            }else if(((c >= '0') && (c <= '9')) || (c == '.') || (c == '+') ||
                                                        (c == '-')){
                CS = NM;
            }else if(c == ':'){
                CS = ASGN;
            }else{
                CS = DLM;
            }
            break;
        }// case H
```

Программная реализация (продолжение)

```
case ASGN:
{
    int colon = c;
    c = fgetc(fd);
    if(c == '='){
       tok.token_name = OPER;
       if((tok.token_value =(char *)malloc(sizeof(2))) == NULL){
           printf("\nMemory allocation error in function \"lexer\"\n");
           return -1;
       }
       strcpy(tok.token_value, ":=");
       add_token(&tok);
       c = fgetc(fd);
       CS = H;
    }else{
       err_symbol = colon;
       CS = ERR;
    }
    break;
 }// case ASGN
```

Программная реализация (продолжение)

```
case DLM:
{
    if((c == '(') || (c == ')') || (c == ';')){
        tok.token_name = DELIM;
        if((tok.token_value = (char *)malloc(sizeof(1))) == NULL){
            printf("\nMemory allocation error in function \"lexer\"\n");
            return -1;
        }
        sprintf(tok.token_value, "%c", c);
        add_token(&tok);
        c = fgetc(fd);
        CS = H;
    }else if((c == '<') || (c == '>') || (c == '=')){
        tok.token_name = OPER;
        if((tok.token_value = (char *)malloc(sizeof(1))) == NULL){
            printf("\nMemory allocation error in function \"lexer\"\n");
            return -1;
        }
```

Программная реализация (продолжение)

```
            sprintf(tok.token_value, "%c", c);
            add_token(&tok);
            c = fgetc(fd);
            CS = H;
        }else{
            err_symbol = c;
            c = fgetc(fd);
            CS = ERR;
        }// if((c == '(') || (c == ')') || (c == ';'))
            break;
    }// case DLM
    case ERR:
    {
        printf("\nUnknown character: %c\n", err_symbol);
        CS = H;
        break;
    }
```

Программная реализация (продолжение)

```
case ID:
{
    int size = 0;
    char buf[256];
    buf[size] = c;
    size++;
    c = fgetc(fd);
    while(((c >= 'A') && (c <= 'Z')) || ((c >= 'a') && (c <= 'z')) ||
                        ((c >= '0') && (c <= '9')) || (c == '_')){
        buf[size] = c;
        size++;
        c = fgetc(fd);
    }
    buf[size] = '\0';
    if(is_keyword(buf)){
        tok.token_name = KWORD;
    }else{
        tok.token_name = IDENT;
    }
```

Программная реализация (продолжение)

```
        if((tok.token_value = (char *)malloc(strlen(buf))) == NULL){
            printf("\nMemory allocation error in function \"lexer\"\n");
            return -1;
        }
        strcpy(tok.token_value, buf);
        add_token(&tok);
        CS = H;
        break;
    } // case ID
```

Программная реализация (продолжение)

```
case NM:
 {



 }
```

Генератор лексических анализаторов

Генераторы лексических анализаторов

- **lex** – стандартный генератор в Unix
- **Flex** – альтернативный вариант классической утилиты lex (https://github.com/westes/flex, http://gnuwin32.sourceforge.net/packages/flex.htm)
- **Quex** – генератор лексических анализаторов для C/C++ (http://quex.sourceforge.net/)
- **RE2C** – генератор лексических анализаторов для C/C++ (http://re2c.org/)
- **JFLex** – генератор на Java (http://jflex.de/)
- **Oolex** – объектно-ориентированный генератор анализаторов на Java (http://luna2.informatik.uni-osnabrueck.de/alumni/bernd/oolex/index.html)
- **Lex под Windows** (http://www.bumblebeesoftware.com/)
- **alex** – генератор лексических анализаторов для Haskell (http://www.haskell.org/alex/)
- **leex** – генератор для Erlang (http://www.erlang.org/doc/man/leex.html)
- **FLEX/EIFFEL** – версия генератора лексических анализаторов для языка Eiffel (http://efsa.sourceforge.net/archive/kalberer/flex.htm)

Генератор лексических анализаторов (Lex / Flex)

- 1975 – М. Леск (Mike Lesk) и Э. Шмидт (Eric Schmidt) разработали lex, генератор лексических анализаторов
- 1987 – В. Пакссон (Vern Paxson) переписал версию lex, написанную на ratfor (расширение Фортрана), на Си и назвал ее flex (Fast Lexical Analyzer Generator)
- Сейчас Flex – https://github.com/westes/flex

Генератор лексических анализаторов (Lex / Flex)

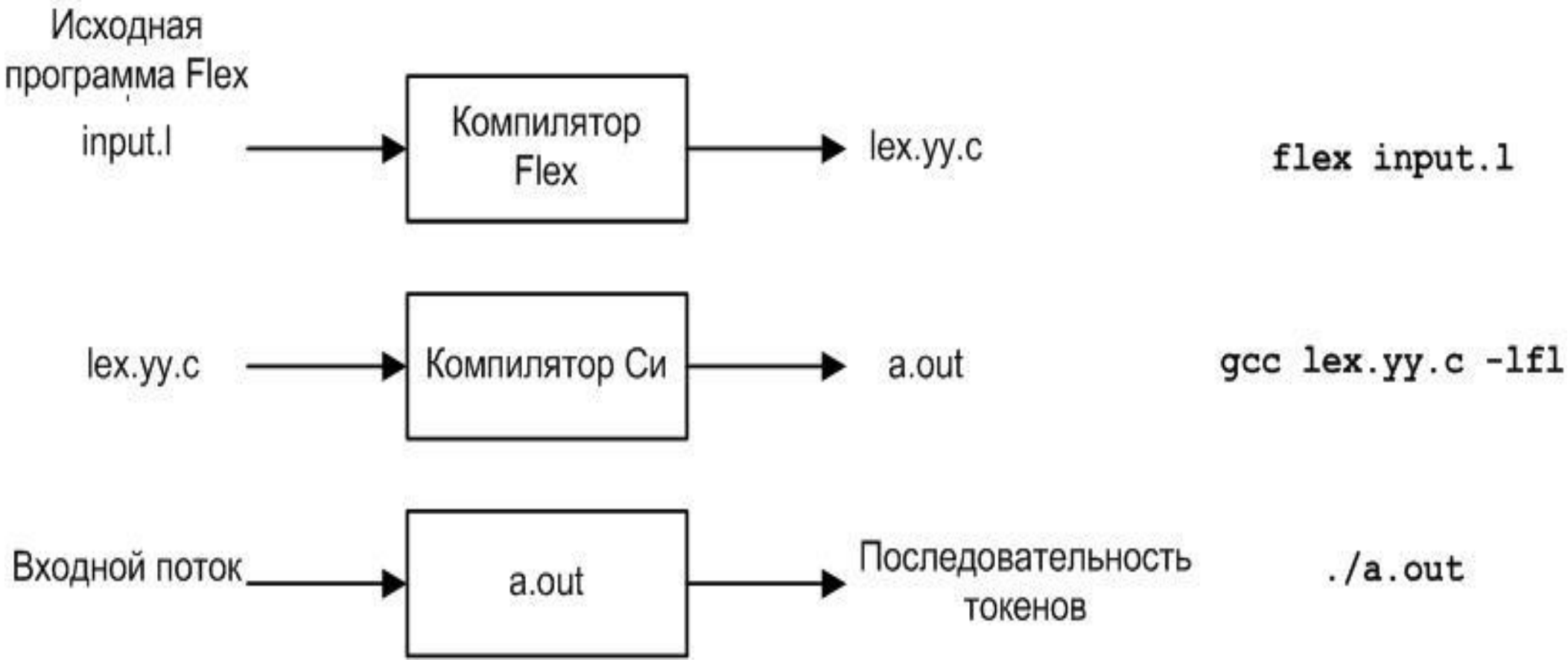
1. Levine J.R. Flex & bison. – O'Reilly Media, Inc., 2009. – 274 p.
   Режим доступа: http://web.iitd.ac.in/~sumeet/flex__bison.pdf

2. Niemann T. A Compact Guide To Lex & Yacc. Режим доступа:
   http://epaperpress.com/lexandyacc/download/LexAndYaccTutorial.pdf

3. Hubert B. Lex и YACC в примерах. Режим доступа: http://rus-linux.net/lib.php?name=/MyLDP/algol/lex-yacc-howto.html

4. Lesk M.E., Schmidt E. (July 21, 1975). Lex – A Lexical Analyzer Generator // UNIX TIME-SHARING SYSTEM:UNIX PROGRAMMER'S MANUAL, Seventh Edition, Volume 2B. bell-labs.com. Режим доступа:
   http://epaperpress.com/lexandyacc/download/lex.pdf

Генератор лексических анализаторов (Lex / Flex)



Структура программы на языке Flex

Объявления
%%
Правила трансляции
%%
Вспомогательные функции

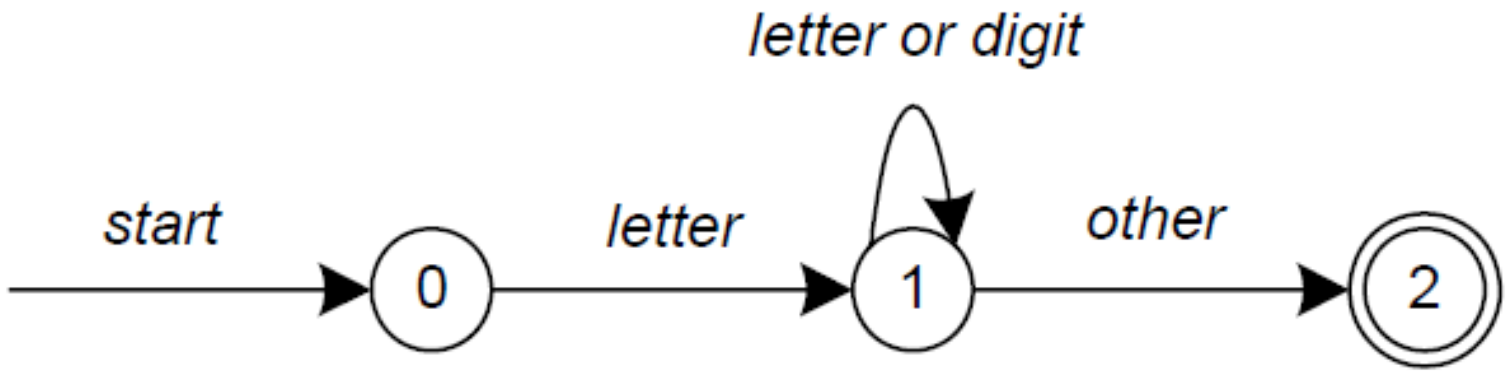
Объявления переменных, именованные константы, регулярные определения

Шаблон { Действие }

Дополнительные функции, используемые в действиях

Как работает Flex

```
letter(letter|digit)*
```



Метасимволы, используемые в Flex

.	Любой символ, кроме \n
\n	Новая строка
*	Повторение 0 или более раз
+	Повторение 1 или более раз
?	Повторение 0 или 1 раз
^	Соответствует началу строки, если указан в начале регулярного выражения. В квадратных скобках соответствует исключению символа из диапазона
$	Конец строки
a | b	a или b
(ab)+	Группировка (повторение последовательности ab 1 или более раз)
“a+b”	Строка a+b
[ ]	Класс (диапазон) символов
{ }	A{1,3} соответствует повторению буквы A от одного до трех раз 0{5} соответствует 00000 {reg_def} – обращение к регулярному определению

Примеры

`abc`	abc
`abc*`	ab abc abcc abccc …
`abc+`	abc abcc abccc …
`a(bc)+`	abc abcbc abcbcbc …
`a(bc)?`	a abc
`[abc]`	a или b или c
`[a-z]`	любая буква в диапазоне от a до z
`[a\-z]`	a или – или z
`[-az]`	– или a или z
`[A-Za-z0-9]+`	любая буква или цифра, повторенная 1 или более раз
`[  \t\n]+`	один или несколько пробелов, знаков табуляции или перехода на новую строку
`[^ab]`	кроме a и b
`[a^b]`	a или ^ или b
`[a|b]`	a или | или b
`a|b`	a или b

Самый короткий файл на языке Flex

```
%%
```

Пример

```
%%

.

\n
```

Пример

```
%%
. ECHO;
\n ECHO;
%%
int main(void)
{
    yylex();
    return 0;
}
```

Вывод данных

```
.    ECHO;

#define ECHO fwrite( yytext, yyleng, 1, yyout )
```

```
%option nodefault
```

Функции, макросы, предопределённые переменные в Flex

`int yylex(void)`	вызов лексического анализатора, возвращает токен
`char *yytext`	распознанная строка
`yyleng`	длина распознанной строки
`yylval`	значение, ассоциирующееся с токеном
`FILE *yyout`	выходной файл (по умолчанию `stdout`)
`FILE *yyin`	входной файл (по умолчанию `stdin`)
`ECHO`	вывод распознанной строки в `yyout`

Правила выбора лексем в Lex

1. Предпочтение всегда отдается более длинному префиксу

2. Если наибольший возможный префикс соответствует двум и более шаблонам, предпочтение отдается шаблону, указанному в программе первым

```
"+"       { return ADD; }
"="       { return ASSIGN; }
"+="      { return ASSIGNADD; }

"if"      { return KEYWORDIF; }
"else"    { return KEYWORDELSE; }
[a-zA-Z_][a-zA-Z0-9_]* { return IDENTIFIER; }
```

Начальные состояния

- *Начальные состояния* позволяют контролировать, когда и какие шаблоны можно использовать

- INITIAL – начальное состояние, которое всегда присутствует в программе Flex

- BEGIN – макрос, позволяющий переключаться между состояниями

- *Исключающие* начальные состояния (%x)

- *Включающие* начальные состояния (%s)

Разработка лексического анализатора с помощью Flex

- Входной язык содержит операторы цикла **for (…; …; …) do …**, разделённые символом **;** (точка с запятой). Операторы цикла содержат идентификаторы, знаки сравнения **<**, **>**, **=**, десятичные числа с плавающей точкой (в обычной и экспоненциальной форме), знак присваивания (**:=**)

Регулярные выражения

Идентификатор — это последовательность букв и цифр. Первым символом должна быть буква; знак подчеркивания (_) считается буквой. Буквы нижнего и верхнего регистров различаются.

```
(_|a-zA-Z)(_|a-zA-Z|0-9)*
```

Вещественная константа с плавающей точкой состоит из целой части, десятичной точки, дробной части, символа e или E, целого показателя степени с необязательным знаком и необязательного суффикса типа — одной из букв f, F, l или L. Целая и дробная части состоят из последовательностей цифр. Может отсутствовать либо целая, либо дробная часть (но не обе сразу). Тип определяется суффиксом: F или f обозначают тип float, L или l — тип long double. При отсутствии суффикса принимается тип double.

```
(-|+)?((0-9)*.(0-9)+|(0-9)+.|(0-9)+)(e|E)(-|+)?(0-9)+(f|l|F|L)?
```

Текст программы на Flex

```
%option yylineno
%{
#include <stdio.h>

int ch;
%}

digit[0-9]
letter[a-zA-Z]
delim[();]
oper[<>=]
ws[ \t\n]

%%
```

Текст программы на Flex (продолжение)

```
for { printf("KEYWORD (%d, %d): %s\n", yylineno, ch, yytext);
      ch += yyleng;
    }
do { printf("KEYWORD (%d, %d): %s\n", yylineno, ch, yytext);
     ch += yyleng;
   }
("_"|{letter})("_"|{letter}|{digit})* {
  printf("IDENTIFIER (%d, %d): %s\n", yylineno, ch, yytext);
  ch += yyleng;
}
[-+]?({digit}*\.{digit}+|{digit}+\.|{digit}+)[eE][-+]?{digit}+[flFL]? {
  printf("NUMBER (%d, %d): %s\n", yylineno, ch, yytext);
  ch += yyleng;
}
```

Текст программы на Flex (продолжение)

```
{oper} { printf("OPERATION (%d, %d): %s\n", yylineno, ch, yytext);
         ch += yyleng;
       }
":=" { printf("OPERATION (%d, %d): %s\n", yylineno, ch, yytext);
       ch += yyleng;
     }
{delim} { printf("DELIMITER (%d, %d): %s\n", yylineno, ch, yytext);
          ch += yyleng;
        }
{ws}+ {ch += yyleng; }
. { printf("Unknown character (%d, %d): %s\n", yylineno, ch, yytext);
    ch += yyleng;
  }


%%
```

Текст программы на Flex (окончание)

```
int main(int argc, char **argv)
{
    if(argc < 2)
    {
        printf("\nNot enough arguments. Please specify filename.\n");
        return -1;
    }
    if((yyin = fopen(argv[1], "r")) == NULL)
    {
        printf("\nCannot open file %s.\n", argv[1]);
        return -1;
    }
    ch = 1;
    yylineno = 1;
    yylex();
    fclose(yyin);
    return 0;
}
```

```
for(abc1:=.;abc1<11.0E+1;abc1:=abc1>.1)do abc1:=abc1=34E5;

$ ./scanner prog
KEYWORD (1, 1): for
DELIMITER (1, 4): (
IDENTIFIER (1, 5): abc1
OPERATION (1, 9): :=
Unknown character (1, 11): .
DELIMITER (1, 12): ;
IDENTIFIER (1, 13): abc1
OPERATION (1, 17): <
NUMBER (1, 18): 11.0E+1
DELIMITER (1, 25): ;
IDENTIFIER (1, 26): abc1
OPERATION (1, 30): :=
IDENTIFIER (1, 32): abc1
OPERATION (1, 36): >
NUMBER (1, 37): .1
DELIMITER (1, 39): )
KEYWORD (1, 40): do
IDENTIFIER (1, 43): abc1
OPERATION (1, 47): :=
IDENTIFIER (1, 49): abc1
OPERATION (1, 53): =
NUMBER (1, 54): 34E5
DELIMITER (1, 58): ;
```

Применение состояний в Flex

```
...
%x COMM

%%


...

"/*" { ch += yyleng; BEGIN COMM; }
<COMM>"/*" { ch += yyleng;
             printf("\"/*\" within comment" (%d, %d): %s\n", yylineno, ch, yytext);
             BEGIN INITIAL;
           }
<COMM>.|\n { ch += yyleng; continue; }
<COMM>"*/" { ch += yyleng; BEGIN INITIAL; }
<COMM><<EOF>> { ch += yyleng;
                printf("Unterminated comment (%d, %d): %s\n", yylineno, ch, yytext);
                BEGIN INITIAL;
              }
...
```

```
/* this is a comment !!! */
for(abc1:=.;abc1<11.0E+1;abc1:=abc1>.1)do abc1:=abc1=34E5;

$ ./scanner prog
KEYWORD (2, 29): for
DELIMITER (2, 32): (
IDENTIFIER (2, 33): abc1
OPERATION (2, 37): :=
Unknown character (2, 39): .
DELIMITER (2, 40): ;
IDENTIFIER (2, 41): abc1
OPERATION (2, 45): <
NUMBER (2, 46): 11.0E+1
DELIMITER (2, 53): ;
IDENTIFIER (2, 54): abc1
OPERATION (2, 58): :=
IDENTIFIER (2, 60): abc1
OPERATION (2, 64): >
NUMBER (2, 65): .1
DELIMITER (2, 67): )
KEYWORD (2, 68): do

...
```

Многострочный комментарий

```
/* this is the 1st line of a comment
   this is the 2nd line of a comment
   this is the last line of a comment
*/

for();
```

```
$ ./scanner prog
KEYWORD (6, 118): for
DELIMITER (6, 121): (
DELIMITER (6, 122): )
DELIMITER (6, 123): ;
```

Многострочный комментарий

```
/* this is the 1st line of a comment
   this is the 2nd line of a comment
   this is the last line of a comment


for();


$ ./scanner prog
Unterminated comment (7, 124):
```