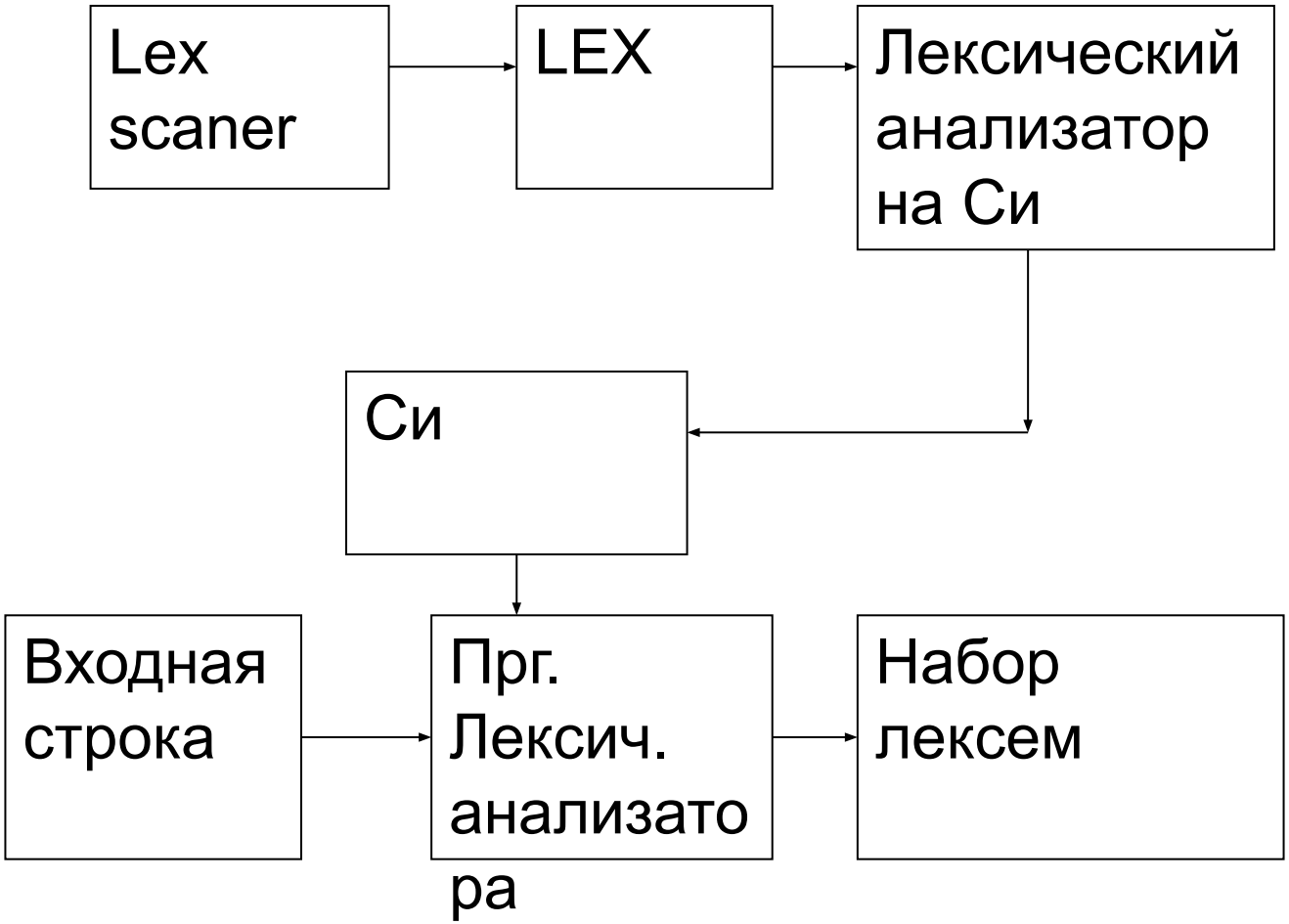
ГЛАВА 6

Генератор лексического анализатора

Архитектура



Определение

Описание
%% - разделение между секциями правила
%%
подпрограммы

Используемые классы символов:
[0-9] [0123456789] – любая цифра
[a-z A-Z] любая буква
[^0 - 7]- любой символ кроме восьмеричной цифры
\n – перевод строк
* - любой символ кроме перевода строки

a “a” \a
\. – получится .
\\ – получится \

{p}* - повторение предыдущего шаблона 0 или более раз
{p}+ - повторение предыдущего шаблона 1 или более раз
{p}? – повторение фрагмента 0 или 1 раз
<p><p> - конкатенация
<p>{m,n} – повторение шаблона от m до n раз
<p>{m} – повторение шаблона m раз
^<p> - фрагмент должен быть в начале строки
<p>$ - фрагмент должен быть в конце строки
<p> / <p> - альтернатива
<p> | <p> - выбор первого элемента
(p) – группировка

Секции

Секция описания
ИМЯ ВЫРАЖЕНИЕ
Если потом это имя встречается, то оно заменяется на это выражение.

Код на Си
%{…}% - всё что находится в этих скобках, копируют в С-факты которые будут сгенерированы.

Секция правил
Шаблон {действие}

Пример:

```
%{
      int I
%}
L [a-z A-Z]
D [0-9]
FLOAT {d}*\ . {d}+
SIGNED(\+|\-)? {FLOAT}
```

Замечания

- Если несколько правил дают код одинаковой длины , то выполняется подстановки соответствующие первому правилу.
- Если один из блоков является короче другого, то сопоставляются самое длинное из подходящих цепочек

  [0-9]

  (\+|\-)?[0-9]+

  (\+|\-)?[0-9]+”.”[0-9]
- Если не один из шаблонов не подходит, то входной символ копируется в поток YYOUT, если это не желательно то надо добавить следующие действия

  {/ */}

  \n;

Секция подпрограмм

- Все копируются в генерируемый Си файл. В ней описывается >=2 функций.
- Int YYWRAP() – определяет что делать при достижении автоматом конца файла, ненулевое значение приводит к завершению разбора, нулевое – к продолжению.
- Интерпретатор таблиц конечного автомата YYLEX
  INT YYLEX() – запускается автомат.
- Автомат прекращает работу если в одном из действий выполняется оператор RETURN или достигнут конец файла и значение YYWRAP() отлично от нуля, а результат YYLEX будет равно 0.

ПРИМЕР

```
NODELIM [^  \ t \ n]
%{
      int l,w,c;
      l = w = c = 0; /* инициализация */
}%
%%
{NODELIM}+ {w++; c + = yyleng; /*слово*/}
\n    {l++;   /*строка*/}
.     {c++;  /*остальные символы */}
%%

main() {return (yylex()); }
int yywrap()
{printf (“line % d word % d chars % d \n”, l ,w,c)};
return(1);
}
```

Полезные функции

- char yytex – буфер в котором накапливается выделяемая процедура.
- int yyleng – длина цепочки, которая находится в буфере.
- FILE * yyin – из него читается информация
- FILE * yyout – в него записывается информация

Функции обработки символов

- int input() – читает информацию из yyin.
- input (int) – помещает символ во входной поток.
- output (int) – помещает символ в выходной поток.
- yymore – следующее значение заносится в буфер yytext
- yyless(n) – она возвращает последние n распознанных символов цепочки во входной поток.
- ECHO – выводит распознанную цепочку в выходной поток.
- REJECT – немедленный переход к следующему правилу без изменения YYTEXT.

Еще пример. Калькулятор

```
%{ #include <stdlib.h>
    #include "yycalc.h"

   extern int yylval;
  void yyerror(char *);
%}

L      [a-z]
D      [0-9]
D8     [0-7]
INT    {D}+
INT8 0{D8}+

%%
{L}   {
         yylval = *yytext - 'a';
         return VARIABLE;
       }
```

```
{INT8} {
         yylval = strtol(yytext,(char**) NULL ,8 );
         return INTEGER;
       }

{INT}  {
         yylval = atoi(yytext);
         return INTEGER;
       }

[-+()=/*\n]       { return *yytext;}

[ \t]  ;



.       yyerror("Unknown character\n");

%%

int yywrap(void) {  return 1; }
```

ГЛАВА 6

ГЕНЕРАТОР СИНТАКСИЧЕСКОГО АНАЛИЗАТОРА

Yet Another Compiler Compiler

Генерирует восходящий распознаватель слева-направо

Спецификация
Секция определения
%%
Секция правил
%%
Секция подпрограмм

Правила задаются в правилах близких
A: BCD
| G – альтернатива
; - конец правила
Все имена в правилах по умолчанию нетерминалы

%left UMIN – унарный минус
%right ASS – право ассоциативно
%nonasoc – все нетерминалы неассоциативны (операции не могут стоять рядом)

```
%token id
%left ‘+’ ‘-’
%left ‘*’ ‘/’
%%
e: e ’+’ e | e ’-’ e | e ’*’ e | e ’/’ e | id
%%
```

```
%%
e:   e ’+’ t
   | e ’-‘ e
   | e ’*’ t
   | t
   | t ’*‘ f
   | t ’/’ f
   | t
 f : id
    | ’(‘ e ’)’
%%
```

Разрешение конфликтов

- Если приоритеты альтернативных действий определены и различны, то выполняется действие с большим приоритетом.
- Если приоритеты альтернативных действий определены и одинаковы, то в случае левой ассоциативности производится свёртка, а в случае правой – сдвиг. Если они не ассоциативны возбуждается ошибочная ситуация.
- Если приоритеты хотя бы одной из действий не специфицированы, то в случае конфликтной свёртки-сдвига выполняется сдвиг, а в случае конфликта свёртка-свёртка выполняется свёртка по правилу определённому выше по тексту в конкретной ситуации.
- Также можно указать приоритет свёртки указав в

Семантика

Семантические действия – это код на Си заключённый в фигурные скобки

```
Statmt:    if ‘(‘expr’)’ statmt {ifc++;}
        | WHILE’(’expr’)’ statmt  {wc++;}
        | ass {assc++;}
        | /* */
        IF ‘(‘expr { action 1 ; } ’)’  ;
```

Семантический стек

В семантический стек попадают значения при свертке правил, они заносятся в псевдопеременные $1, $2 … $n, результат свертки – в $$.

```
expr: expr ’+’ expr
{$$= $1 + $3 ; }
```

yylval – здесь поддерживаются постоянные значения

Все символы получали номер от 1 до 256

Пример. Калькулятор

```
%{
#include <stdio.h>
void yyerror(char*);
int yylex(void);
int sym[26];
%}

%token INTEGER VARIABLE
%left '+' '-'
%left '*' '/'
%left UMINUS

%%
program: /*EMPTY*/
        |program statement '\n'
        |program error '\n'
        { yyerrok;}
        ;
```

```
statement:
        expression      { printf("%d\n", $1);}
        | VARIABLE '=' expression {sym[$1] = $3; }
        ;

expression:
        INTEGER
        | VARIABLE      {$$ = sym[$1];}
        | expression '+' expression {$$ = $1 + $3;}
        | expression '-' expression {$$ = $1 - $3;}
        | expression '*' expression {$$ = $1 * $3;}
        | expression '/' expression {$$ = $1 / $3;}
        | '-' expression %prec UMINUS {$$ = -$2;}
        | '(' expression ')' {$$ = $2;}
        ;
%%
void yyerror(char *s) {  fprintf(stderr, "%s\n", s); }
int main(void){  yyparse();  return 0; }
```

Сборка
LEX и YACC вместе

```
CC = gcc
CFLAGS = -Wall -O2

O_TARGET = calc

objs = yycalc.o llcalc.o

all_target      : $(objs)
        $(CC) -o$(O_TARGET)
```

```
$(objs)
yycalc.o  : yycalc.c

llcalc.o    : llcalc.c

llcalc.c    : calc.l
        win_flex -o llcalc.c calc.l

yycalc.c  : calc.y
        win_bison -d -o yycalc.c calc.y

clean :
        rm *.h *.c *.o $(O_TARGET)
```