5.2. СКАНЕР

Сканер как составная часть языкового процессора стоит на первой фазе обработки исходного текста. Функциональным назначением сканера является *лексическая свертка* исходного текста программы в символы, идентифицируемые в дальнейшем как ключевые слова, числовые константы, встроенные функции и так далее. Кроме того, на этапе лексического анализа происходит так называемая фильтрация незначащей части текста.

Незначащей частью текста будем называть вспомогательные символы, которые не являются носителями смыслового содержания текста. Такими символами являются символы табуляции ‘*t*’, символы перевода на новую строку ‘*n*’, пробелы ‘ ’, комментарии и другие. Эти символы используются только при редактировании исходного текста, когда необходимо соблюдать определенное форматирование текста и повысить читабельность.

На рис 5.5 показана схема потоков при фильтрации исходного текста.

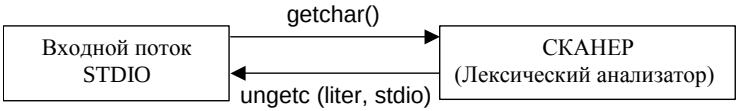

*Рис. 5.5.*Схема потоков сканера при фильтрации

Реализация процедуры фильтрации (очистки от *мусора* [13]) осуществляется с помощью стандартных функций getchar( ) и ungetc( ) из встроенной стандартной библиотеки <stdio.h>. Ниже (рис. 5.6) приведена функция clear( ), которая очищает текст от символов табуляции, переводов строки и пробелов и возвращает отфильтрованный текст во входной поток.

```
#include  <stdio.h>
int  clear ( )
{
int liter;
while (1) /* цикл по всему входному потоку символов*/
{
liter = getchar( );
if ( liter = =    ' ' | |  liter = =   ' \n ' ||  liter = = ' \t ' )
then ungetc (liter, stdio);
}
}
```

Рис. 5.6. Фильтрация

Часто в процессорах функцию сканера заменяет синтаксический анализатор, при этом нет необходимости в специальном просмотре исходного текста, который осуществляется на этапе синтаксического анализа. В отличие от синтаксического анализатора сканер определяет лишь принадлежность символов алфавиту языка и не устанавливает принадлежность языковых конструкций к грамматике.

Проиллюстрируем разработку сканера, используя грамматику арифметических выражений G[<АВ>]

<АВ> → Т | <АВ> + Т | <АВ> – Т
Т → О | О∗Т | О/Т
О → (<АВ>) | <Идентификатор> | <ЦБЗ>
<Идентификатор>→ Б{Б|Ц}
<Целое без знака>→Ц{Ц},

где Б – [*A, B, C,..., Z*]; Ц – [0,1,…,9]; <ЦБЗ> – <Целое без знака>. Функцией сканера на этом примере является внутреннее представление символов арифметического выражения, под которым будем понимать символический (условный) код. Этим кодом характеризуются идентификаторы, числовые константы и другие объекты языка. Для рассматриваемого примера арифметических выражений примем следующие соглашения относительно лексем:

Символ	<ЦБЗ>	<идент-р>	+	–	/	*	(	)
Условный код	1	2	3	4	5	6	7	8

На рис. 5.7 приведена диаграмма состояний для грамматики **G**[<АВ>]. На диаграмме представлена посимвольная декомпозиция арифметических выражений с генерацией соответствующего символического кода символов <идентификатор>, <целое без знака> и литер «+», «–» и др. Непомеченные дуги на диаграмме соответствуют состоянию ERROR (отсутствие данного символа в словаре грамматики) либо выходу из обработки очередного символа и переходу на start для обработки следующего терминального символа.

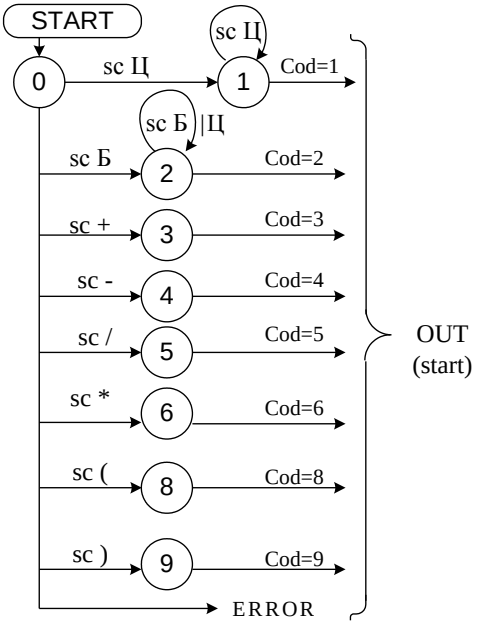

Рис. 5.7. Диаграмма состояний сканера

Пример

Пусть строка с арифметическим выражением имеет вид:

A1*A2 #.

Перемещение по узлам графа для данного примера:

0 – 2 – 2 – OUT – START – 6 – OUT – START – 2 – 2 – OUT

СЕМАНТИКА СКАНЕРА

На семантической диаграмме «навешаны» семантические атрибуты: SC – сканирование очередного символа, Cod – адресная генерация кода соответствующего символа.

Реализацию сканера по диаграмме состояний или по графу на рис. 5.6 приведем на языке *C*. Отметим, что данная реализация универсальна для графов, в которых переходы соответствуют терминальному символу из узла в узел либо по петле. Поэтому программа для графа <числовая константа> (см. рис. 5.3) мало чем будет отличаться от приведенной ниже.