Глава 2. Лексический анализ

2.1. Задачи лексического анализа

Лексический анализ (сканирование) – первая фаза компиляции. Реализуется частью компилятора, которая называется *лексическим анализатором* (или *сканером*). Его основная задача состоит в предварительной обработке исходного текста программы, которая заключается в группировании символов входного потока в лексические единицы (*лексемы*). Для каждой лексемы сканер формирует выходной *токен* вида <*код_токена*, *атрибут*> для последующих фаз компиляции. *Код_токена* идентифицирует класс лексемы (*лексический класс*) и определяет работу синтаксического анализатора (рассматривается как терминал). Для удобства *код_токена* будем представлять абстрактным именем (или специальным обозначением), выделенным жирным шрифтом, и ссылаться на токен по его имени (обозначению). *Атрибут* токена обеспечивает доступ к дополнительной информации о лексеме, если лексическому классу соответствует множество лексем, и определяет трансляцию токена (семантический анализ и генерация промежуточного кода).

Часто фазы лексического и синтаксического анализа объединяют в один проход. В этом случае лексический анализатор является подпрограммой синтаксического анализатора (рис. 2.1). Когда синтаксическому анализатору требуется очередной токен, он вызывает лексический анализатор, который формирует очередной токен и возвращает управление синтаксическому анализатору. В результате исходная программа полностью преобразуется в последовательность токенов.

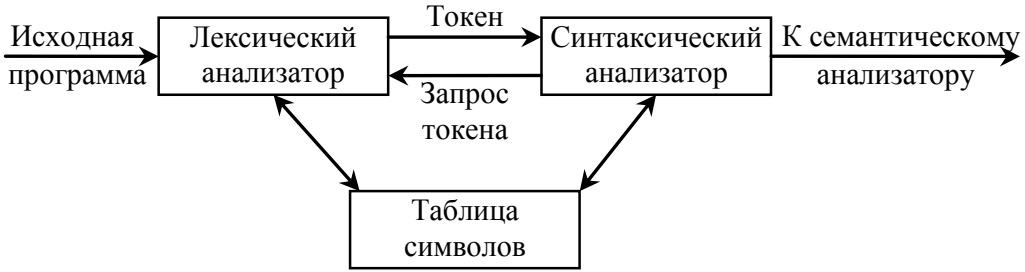

Рис. 2.1. Взаимодействие лексического и синтаксического анализаторов

Лексический анализатор выполняет также и другие функции. В частности, он удаляет из текста исходной программы комментарии и не несущие смысловой нагрузки пробелы, символы табуляции и символы перевода строки. Еще одна его задача – согласование сообщений об ошибках компиляции и текста исходной программы (указать каким-либо образом позицию ошибки и ее характер в тексте программы). Кроме того, лексический анализатор должен строить различные таблицы, необходимые как для собственно лексического анализа, так и для последующих фаз компиляции.

2.2. Лексические классы

Множество лексем разбивается на непересекающиеся подмножества лексических классов, неразличимых с точки зрения синтаксического анализа. Каждый лексический класс описывается соответствующими правилами (*шаблон* токена). В большинстве языков программирования токенами являются ключевые слова, идентификаторы, константы, символы операций, символы пунктуации (скобки, запятые и т. д.). Токену может соответствовать единственная лексема (по одному токену для каждого ключевого слова, символа операции и символа пунктуации), конечное или бесконечное множество лексем (идентификаторы, константы, сгруппированные в один токен наборы операций). Для формального описания шаблонов токенов используются регулярные грамматики или регулярные выражения.

Многие языки используют определенные заранее лексемы в качестве указания конкретных конструкций языка или специальных символов пунктуации (**begin**, **end**, **while**, **do** и т. д.). Такие лексемы называются *ключевыми словами* и обычно удовлетворяют правилам образования идентификаторов. Поэтому необходим специальный механизм, позволяющий отличить ключевые слова от других идентификаторов. Для упрощения решения этой проблемы во многих языках ключевые слова *зарезервированы*, т. е. они не могут использоваться в качестве идентификаторов. Тогда лексема является идентификатором только в том случае, если она не является ключевым словом.

Атрибутом токенов, которым может соответствовать бесконечное множество лексем (идентификаторы, константы), является указатель на соответствующую запись в таблице символов, в которой хранится информация о токене. Если шаблону токена соответствует конечное множество лексем (например, символы операций сравнения), то атрибутом может быть не указатель на соответствующую запись в специальной таблице операций сравнения, а соответствующий код операции. В этом случае отпадает необходимость явно хранить эту таблицу операций. Если шаблону токена соответствует единственная лексема, то атрибут имеет пустое значение (будем обозначать символом 0).

Выделение лексических классов в языках программирования обусловлено в первую очередь эффективностью выполнения последующих фаз компиляции. Например, набор всех шести операций сравнения можно сгруппировать в один токен (чаще всего так и делается), а можно каждой операции сопоставить свой токен. В первом случае токен рассматривается как единственный терминал при синтаксическом анализе для любой операции сравнения, а атрибут дает информацию о семантике операции сравнения для трансляции. Во втором случае имеется шесть токенов (шесть терминалов для синтаксического анализа) и сам же токен несет информацию о семантике операции.

Рассмотрим фрагмент исходной программы:

for *i* := 1 **to** 20 **do** *MyArr*[*i*] := 0;

Пусть в процессе формирования таблицы символов информация об идентификаторах *i* и *MyArr* оказалась в записях с номерами 3 и 7 соответственно, а о константах 1, 20 и 0 – в записях с номерами 2, 10 и 11 соответственно. Тогда сканер сформирует следующую последовательность токенов:

<**for**, 0>, <**id**, 3>, <**ass**, 0>, <**num**, 2>, <**to**, 0>,

<**num**, 10>, <**do**, 0>, <**id**, 7>, <**[**, 0>, <**id**, 3>, <**]**, 0>,

<**ass**, 0>, <**num**, 11>, <**;**, 0>.

Здесь имена токенов **for**, **to**, **do** обозначают соответствующие ключевые слова, **ass** – операцию присваивания, **id** – идентификатор, **num** – числовую константу, остальные токены обозначены соответствующими абстрактными символами **[,], ;**. Токены **for**, **to**, **do**, **ass**, **[,]**, **;** имеют пустые значения атрибутов, поскольку для каждого из них в качестве шаблона определена единственная лексема. Атрибутами токенов **id** и **num** являются указатели на соответствующие записи в таблице символов, так как им может соответствовать бесконечное множество лексем.

2.3. Таблица символов

Таблица символов представляет собой структуру данных, которая используется компилятором для хранения информации о конструкциях исходной программы. Структура данных должна обеспечивать компилятору возможность быстрого поиска нужной записи, а также возможность быстрого сохранения данных в записи и получения их из нее. Некоторые компиляторы формируют единую хеш-таблицу, что обеспечивает, по сути, константное время доступа к нужной записи.

В ряде случаев таблицу символов удобно реализовать с помощью нескольких отдельных таблиц, например, таблица ключевых слов, таблица идентификаторов, таблица констант. Очевидно, что в этом случае таблица ключевых слов является статической и ее содержимое не изменяется в процессе компиляции (носит константный характер). Таблицы идентификаторов и констант являются динамическими, для них нужна структура данных, обеспечивающая наибольшую эффективность работы (вплоть до организации хеш-таблиц). Числовые константы перед помещением их в таблицу могут переводиться из внешнего символьного представления во внутреннее машинное представление.

В процессе лексического анализа формируются начальные элементы таблицы символов для хранения информации об объектах. По мере выполнения других фаз компиляции таблица дополняется новыми данными. В общем случае информация, хранимая в таблице символов, зависит от семантики входного языка и вида объекта. Например, для имени переменной может храниться ее лексема, тип (вещественный, целый и т.д.), точность, длина, адрес памяти, число измерений и значения граничных пар (для массивов); для имени функции – количество и типы формальных параметров, тип возвращаемого результата, адрес вызова кода функции и т.п.

Если лексема распознается как идентификатор, то осуществляется ее поиск в таблице символов, если поиск безуспешный, лексема добавляется в таблицу. Следует отметить, что поиск и добавление идентификаторов в таблицу символов осуществляется сканером. Другие фазы компиляции имеют прямой доступ к нужной записи через атрибут соответствующего токена.

Во многих языках программирования есть предопределенные идентификаторы (имена стандартных типов, процедур, функций), которые не являются ключевыми словами. Такие идентификаторы должны быть занесены в таблицу символов заранее.

Многие языки программирования имеют структуру вложенных блоков и процедур, когда один и тот же идентификатор может быть объявлен и использован поразному в различных блоках и процедурах. В этом случае важным становится понятие области видимости объявлений. *Область видимости* объявления представляет собой часть программы, в которой может применяться данное объявление.

Можно реализовать области видимости путем использования отдельной таблицы символов для каждой области видимости, т.е. программный блок с объявлениями будет иметь собственную таблицу символов с данными для каждого объявления в блоке. При выходе из блока соответствующая таблица символов может быть удалена (если она не требуется для последующих фаз компиляции).

Другой подход заключается в применении одной таблицы символов для всех блоков. В этом случае данные об идентификаторе дополняются номером блока, т.е. один и тот же идентификатор с различными номерами блоков будут иметь отдельную запись в таблице символов и рассматриваться как разные идентификаторы.

Блочную структуру программы можно распознать только при выполнении фазы синтаксического анализа. Поэтому синтаксический анализатор при запросе следующего токена должен предоставить лексическому анализатору номер блока.