**Лексический анализатор**

**3.1 Введение в лексический анализ**

Лексический анализатор – это первый этап процесса компиляции, на котором символы, составляющие исходную программу, группируются в отдельные минимальные единицы текста, несущие смысловую нагрузку – лексемы.

На вход лексического анализатора (в дальнейшем будем называть его ЛА) поступает текст исходной программы, а выходная информация передается для дальнейшей обработки компилятором на этапе синтаксического анализа.

Все символы входной последовательности разделяются на символы, принадлежащие каким-либо лексемам, и символы, разделяющие лексемы (разделители). В некоторых случаях между лексемами может и не быть разделителей. С другой стороны, в некоторых языках лексемы могут содержать незначащие символы.

Традиционно принято организовывать процесс лексического анализа, рассматривая входную последовательность символов, как поток символов. При такой организации, процесс самостоятельно управляет выборкой отдельных символов из входного потока. Из-за специфики такой работы лексический анализатор иногда называют сканером.

Распознавание лексем в контексте грамматики обычно производится путём их идентификации (или классификации), согласно идентификаторам (или классам) токенов, определяемых грамматикой языка. При этом, любая последовательность символов входного потока (лексема), которая, согласно грамматике, не может быть идентифицирована как токен языка, обычно рассматривается как специальная токен-ошибка.

Каждый токен можно представить в виде структуры, содержащей идентификатор токена (или идентификатор класса токена) и, если нужно, последовательность символов *лексемы* выделенной из входного потока (строку, число и т. д.).

Пример. If a = b+c; end @

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Тип |
| Condition | If | Ключевое слово |
| Id | a | Идентификатор |
| Comparison | = | Сравнение |
| Id | b | Идентификатор |
| Plus | + | Арифметическая операция |
| Id | c | Идентификатор |
| Separator | ; | Разделитель |
| Condition | end | Ключевое слово |
| Error | @ | Ошибка |

Лексический анализатор необязательный этап компиляции, но желательный по следующим причинам:

1) Упрощается работа с текстом исходной программы на этапе синтаксического разбора и сокращается объем обрабатываемой информации, так как лексический анализатор структурирует поступающий на вход исходный текст программы и выкидывает всю незначащую информацию.

2) ЛА уменьшает длину программы, устраняя из ее исходного представления несущественные пробелы и комментарии.

3) Сканер отделяет сложный по конструкции синтаксический анализатор от работы непосредственно с текстом исходный программы, структура которого может варьироваться в зависимости от версии входного языка - при такой конструкции компилятора при переходе от одной версии языка к другой достаточно только перестроить относительно простой сканер.

4) Для выделения в тексте и разбора лексем возможно применять простую, эффективную и теоретически хорошо проработанную технику анализа, в то время как на этапе синтаксического анализа конструкций исходного языка используются достаточно сложные алгоритмы разбора.

В процессе выделения лексем ЛА может, как самостоятельно строить таблицы имен и констант, так и выдавать значения для каждой лексемы при очередном обращении к нему. В этом случае таблица имен строится совместно в нескольких фазах (например, в лексическом анализе делается попытка добавить новое имя, а синтаксический анализ разрешает или отвергает попытку и присваивает лексеме некоторые атрибуты, которые уточняются контекстным анализатором).

Функции, выполняемые лексическим анализатором, и состав лексем, которые он выделяет в тексте исходной программы, могут меняться в зависимости от версии компилятора. Тем не менее главной функцией лексического анализатора является распознавание слов (лексем) и преобразование их к виду, удобному для последующей обработки.

Но помимо этого лексический анализатор также может выполнять и другие задачи:

- введение номеров строк

- удаление комментариев

- определение ошибок (недопустимые символы, неправильная запись чисел, идентификаторов и др.).

Для того чтобы качественно написать распознаватель лексем, необходимо знать теоретические основы построения лексических анализаторов.

**3.2 Классы лексем**

В определении лексического анализатора мы встретились с понятием лексем. Рассмотрим это понятие подробнее.

Алфавит, используемый для записи языка программирования, ограничивается сравнительно небольшим множеством литер, обеспечиваемых устройствами ввода-вывода. Поэтому для записи конструкций, лежащих даже на нижнем уровне иерархии, используются не отдельные литеры, а цепочки литер, они то я называются лексемами. Символы, образующие лексему, теряют свою индивидуальность: смысловое значение, приписываемое этой строке семантикой, не вытекает из смысловых значений составляющих ее символов.

Текст программы можно рассматривать как цепочку в алфавите лексем. Такая цепочка получила название лексической свертки программы.

Обычно все лексемы делятся на классы. Примерами таких классов являются:

* Идентификаторы
* Число
* Операции
* Разделители (однолитерные, двулитерные)
* Знаки
* Ключевые слова
* Шумовые слова

Рассмотрим эти классы подробнее.

**Идентификаторы.** Идентификаторами обычно называют лексемы, состоящие из букв и цифр и начинающиеся с буквы. В разных языках допускаются следующие вариации этого (синтаксического) правила: определяется ограничение на длину цепочки (например, в Фортране длина идентификатора не превышает шести), разрешается включать в идентификаторы специальные литеры, такие как «.» или «—», или запрещается использовать в качестве идентификаторов определенные слова, называемые ключевыми или зарезервированными (например, VAR, DIV или FOR в. языке Паскаль).

Например:

int a, b,c ;

В данном случае идентификаторами являются a, b, c, в то время как int зарезервировано и идентификатором быть не может

**Число или цифра.** Цифры так же являются отдельным классом лексем. Число – последовательность цифр.

**Символы (знаки) операций.** Для обозначения операций в большинстве языков используются отдельные специальные литеры (например, «+» или «—») или их цепочки (например, \*\* или :==).

**Ключевые слова**. В каждом языке программирования имеется набор лексем, используемых в качестве фиксированных частей в синтаксисе понятий. В качестве таких лексем используются либо отдельные литеры (например, (, [, begin или for в Алголе 60), идентификаторы специального вида (например, IF или DO в Фортране) или ключевые слова (например, IF или ТО в Паскале).

Например:

int max;

if (a < b)

{

max = b;

}

else

{

max = a;

}

Ключевыми словами являются if, else, int.

**Шумовые или необязательные слова**. Большинство языков предусматривает возможность включать различного рода необязательные слова, которые игнорируются исполняющей машиной. Иногда они определяются явно (например, в Паскале комментариями являются последовательности символов, содержащие специальные ограничители) или неявно, через контекст (например, в Алголе 60 любая последовательность, лежащая между символами end и ; и не содержащая этих символов, рассматривается в качестве комментария). Во многих языках используются комбинированные правила задания шумовых слов (например, в языке Фортран пробелы существенны только в строках).

**Разделители**. В некоторых языках требуется размещение шумовых слов между определенными видами лексем (например, в Паскале между идентификатором и ключевым словом должен находиться хотя бы один символ конца строки, комментарий или пробел).

Все эти лексемы описываются регулярными выражениями.

**3.6 Общая схема работы лексического анализатора**

Итак, какова же схема работы лексического анализатора?

* Сначала выделяем отдельную лексему
  + Если это идентификатор, то делается проверка на принадлежность его множеству ключевых слов. Если да, то выдается признак соответствующего ключевого слова, если нет – выдается лексема идентификатора. Сам идентификатор сохраняется в таблице имен, если его еще там нет. В лексему помещается ссылка на таблицу имен.
  + Если выделенная лексема принадлежит какому-либо из других классов лексем (число, строка и т.д.), то выдается признак лексема соответствующего класса, а значение лексемы сохраняется в качестве признака или в таблице имен.

Лексический анализатор может быть как самостоятельной фазой трансляции, так и подпрограммой, работающей по принципу «дай лексему». В первом случае (а) выходом анализатора является файл лексем, во втором (б) лексема выдается при каждом обращении к анализатору (при этом, как правило, признак класса лексемы возвращается как результат функции «лексический анализатор», а значение лексемы передается через глобальную переменную). С точки зрения обработки значений лексем, анализатор может либо просто выдавать значение каждой лексемы, и в этом случае построение таблиц объектов (идентификаторов, строк, чисел и т.д.) переносится на более поздние фазы, либо он может самостоятельно строить таблицы объектов. В этом случае в качестве значения лексемы выдается указатель на вход в соответствующую таблицу.

|  |
| --- |
| Тип, Значение  Таблица  Синт. анализатор  Лекс. анализатор  Значение  Тип лексемы  а  б |
| Рис. 3.10 |

Второй случай может быть предпочтительней, т.к. тогда синтаксический и лексический анализатор могут работать более слажено. Скажем синтаксический анализатор сообщает лексическому о возможности появления нового идентификатора или нет. Что улучшает обработку ошибок и позволяет проставить некоторые атрибуты лексем до контекстного анализа.

**3.7 Программирование лексических анализаторов**

Обобщим вышесказанное.

На вход лексическому анализатору подается

* Текст исходной программы;
* Список зарезервированных слов языка;
* Список служебных символов и символов разделителей.

На выходе лексический анализатор выдает:

* Последовательность классов лексем. (Ограничителей, ключевых слов, идентификаторов, литералов и т.д.)
* Таблица имен, содержащая уточняющую информацию для идентификаторов и возможно литералов.

Существует много способов построения лексического анализатора. Классическим является анализатор на основе автоматной модели и графа переходов. Достоинством такого подхода, является простота программы, но высокая сложность таблицы переходов.

Как уже отмечалось ранее, лексический анализатор (ЛА) может быть оформлен как подпрограмма. При обращении к ЛА, вырабатываются как минимум два результата: тип выбранной лексемы и ее значение (если оно есть).

Если ЛА сам формирует таблицы объектов, он выдает тип лексемы и указатель на соответствующий вход в таблице объектов. Если же ЛА не работает с таблицами объектов, он выдает тип лексемы, а ее значение передается, например, через некоторую глобальную переменную. Помимо значения лексемы, эта глобальная переменная может содержать некоторую дополнительную информацию: номер текущей строки, номер символа в строке и т.д. Эта информация может использоваться в различных целях, например, для диагностики.

В основе ЛА лежит диаграмма переходов соответствующего конечного автомата. Отдельная проблема здесь - анализ ключевых слов. Как правило, ключевые слова - это выделенные идентификаторы. Поэтому возможны два основных способа распознавания ключевых слов: либо очередная лексема сначала диагностируется на совпадение с каким-либо ключевым словом и в случае неуспеха делается попытка выделить лексему из какого-либо класса, либо, наоборот, после выборки лексемы идентификатора происходит обращение к таблице ключевых слов на предмет сравнения.

Рассмотрим несколько подробнее вопросы программирования ЛА. Основная операция лексического анализатора, на которую уходит большая часть времени его работы - это сканирование очередного символа и проверка на принадлежность его некоторому диапазону. Например, основной цикл при выборке числа в простейшем случае может выглядеть следующим образом:

While (Insym<='9' &&Insym>='0')

{...}

Программу можно значительно улучшить следующим образом. Пусть LETTER, DIGIT, BLANK - элементы перечислимого типа. Введем массив map, входами которого будут символы, значениями - типы символов. Инициализируем массив map следующим образом:

map['a']=LETTER;

.......

map['z']=LETTER;

map['0']=DIGIT;

........

map['9']=DIGIT;

map[' ']=BLANK;

........

Тогда приведенный выше цикл примет следующую форму:

while(map[Insym]==DIGIT)

{...}

Выделение ключевых слов может осуществляться после выделения идентификаторов. ЛА работает быстрее, если ключевые слова выделяются непосредственно.

Для этого строится конечный автомат, описывающий множество ключевых слов. Рассмотрим пример программирования этого конечного автомата на языке Си:

|  |
| --- |
| i  Ключевое слово ***if***  Идентификатор  Не буква и не цифра  f  Буква или цифра  Не ***f*** и не ***n***  n  t  Не ***t***  Ключевое слово ***int***  Буква или цифра  Не буква и не цифра |
| Рис 3.11 |

Здесь cp - указатель текущего символа. В массиве map классы символов кодируются битами.

Поскольку ЛА анализирует каждый символ входного потока, его скорость существенно зависит от скорости выборки очередного символа входного потока. В свою очередь, эта скорость во многом определяется схемой буферизации. Рассмотрим возможные эффективные схемы буферизации.

Будем использовать буфер, состоящий из двух одинаковых частей длины N (а), где N - размер блока обмена (например, 1024, 2048 и т.д.).

|  |
| --- |
| а  N  N  N  N  Начало  Начало  Продвижение  Продвижение  б  **eof** |
| Рис 3.12 |

Чтобы не читать каждый символ отдельно, в каждую из половин буфера поочередно одной командой чтения считывается N символов. Если на входе осталось меньше N символов, в буфер помещается специальный символ (eof). На буфер указывают два указателя: продвижение и начало. Между указателями размещается текущая лексема. Вначале они оба указывают на первый символ выделяемой лексемы. Один из них, продвижение, продвигается вперед, пока не будет выделена лексема, и устанавливается на ее конец. После обработки лексемы оба указателя устанавливаются на символ, следующий за лексемой. Если указатель продвижение переходит середину буфера, правая половина заполняется новыми N символами. Если указатель продвижение переходит правую границу буфера, левая половина заполняется N символами и указатель продвижение устанавливается на начало буфера.

При каждом продвижении указателя необходимо проверять, не достигли ли мы границы одной из половин буфера. Для всех символов, кроме лежащих в конце половин буфера, требуются две проверки. Число проверок можно свести к одной, если в конце каждой половины поместить дополнительный «сторожевой» символ, в качестве которого логично взять eof (б).

В этом случае почти для всех символов делается единственная проверка на совпадение с eof и только в случае совпадения нужно дополнительно проверить, достигли ли мы середины или правого конца.

**3.8 Лексический анализ различных языков программирования**

Некоторые языки обладают особенностями, существенно затрудняющими лексический анализ. Такие языки, как Фортран и Кобол, требуют размещения конструкций языка в фиксированных позициях входной строки. Такое размещение лексем могло быть очень важным при выяснении корректности программы. Например, при переносе строки в Коболе необходимо поставить специальный символ в 6-й колонке, иначе следующая строка будет разобрана неправильно. Основной тенденцией современных языков программирования является свободное размещение текста программы.

От одного языка к другому варьируются правила использования символов языка, в частности, пробелов. В некоторых языках, таких как Алгол 68 и Фортран, пробелы являются значащими только в строковых литералах.

Рассмотрим популярный пример, иллюстрирующий потенциальную сложность распознавания лексем в Фортране. В операторе DO 5 I = 1.25 мы не можем определить, что DO не является ключевым словом до тех пор, пока не встретим десятичную точку. С другой стороны, в операторе DO 5 I = 1,25 мы имеем семь лексем: ключевое слово DO, метку 5, идентификатор I, оператор =, константу 1, запятую и константу 25. Причем, до тех пор пока мы не встретим запятую, мы не можем быть уверены в том, что DO – это ключевое слово. Чтобы как-то разрешить эту ситуацию, Fortran 77 позволяет использовать необязательную запятую между меткой и индексом DO оператора. Использование такой запятой позвляет сделать DO оператор понятнее и более читабельным.

В большинстве современных языков программирования ключевые слова являются зарезервированными, т.е. их смысл предопределен и не может быть изменен пользователем. Если ключевые слова не являются зарезервированными, то лексический анализ должен уметь различать ключевые слова и определенные пользователем идентификаторы. Естественно, что это сильно затрудняет лексический анализ; например, в PL/I вполне легален следующий оператор:

IF THEN THEN THEN = ELSE; ELSE ELSE = THEN;

При разборе такого оператора необходимо постоянно переключаться с режима "THEN, ELSE как ключевые слова" на трактовку "THEN, ELSE как идентификаторы", и обратно.

**Контрольные вопросы**

* 1. Какова главная функция лексического анализатора?
  2. Перечислите классы лексем.
  3. В чем отличие регулярного выражения от регулярного множества?
  4. Какие виды разборов Вы знаете?
  5. Как строится диаграмма состояний для праволинейного (леволинейного) регулярного выражения?

**Задания для самоконтроля**

1. Составьте таблицу разбора лексического анализатора для оператора for.
2. Составьте регулярное выражение для грамматики

S→AB

A→X | Y

X→x | xX

Y→ y | yY

B→b | bB

1. Проведите разбор цепочки cabc. Какой разбор Вы выберете?

S→Bc

B→Ab

A→Сa

С→с

1. Постройте диаграмму состояний для грамматики

S→C

B→B1 | 0 | D0

C → B1 | C1

D → D0 | 0