**Лабораторная работа №2**

**Лексический анализ**

 ***Цель работы*** – закрепление теоретических знаний по методам построения лексических анализаторов (сканеров), способам построения распознавателей для регулярных языков; приобретение умений и практических навыков синтеза распознающих конечных автоматов и их программной реализации, выбора структур данных для представления таблиц сканера, разработки лексического анализатора.

**Основные сведения**

 Лексический анализ (сканирование) является первой фазой компиляции. 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. Когда синтаксическому анализатору требуется очередной токен, он вызывает лексический анализатор, который формирует очередной токен и возвращает управление синтаксическому анализатору.

 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 одной задачей является 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.

 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.

 Атрибутом для токенов, которым может соответствовать бесконечное множество лексем (идентификаторы, константы), является указатель на соответствующую запись в таблице, 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, которое будем обозначать символом 0.

 Выделение токенов в языках программирования обусловлено в первую очередь эффективностью выполнения последующих фаз компиляции. Например, набор всех шести операций сравнения учебного языка 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.

 В процессе лексического анализа формируются начальные элементы таблицы символов для хранения информации об идентификаторах (и, возможно, также для констант). 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 элемента. Например, для имени переменной может храниться ее 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.

 В ряде случаев таблицу символов удобно реализовать с помощью двух таблиц: таблицы идентификаторов и таблицы констант. Числовые константы перед помещением их в таблицу могут переводиться из внешнего символьного представления во внутреннее машинное представление.

 Таким образом, таблица символов является динамической и необходимо выбрать структуры данных и алгоритмы, обеспечивающие наибольшую эффективность работы. Для работы с таблицей символов следует использовать хеширование [6].

 Во многих языках программирования имеют место предопределенные идентификаторы (имена стандартных типов, процедур, функций), которые не являются ключевыми словами. Такие идентификаторы должны быть занесены в таблицу символов заранее. В языке «У» предопределенными идентификаторами являются имена типов integer, float, Boolean, char.

 Многие языки программирования имеют структуру вложенных блоков и процедур, когда один и тот же идентификатор может быть объявлен и использован по-разному 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. В данной работе все идентификаторы следует рассматривать как относящиеся к самому внешнему блоку, поскольку лексический анализатор не в состоянии распознать блочную структуру.

 Множество лексем, соответствующих некоторому токену, можно рассматривать как формальный язык этого токена (язык идентификаторов, язык констант и т. д.). Языки токенов настолько просты, что в качестве формального описания шаблонов токенов достаточно использовать регулярные грамматики или регулярные выражения.

 Существует полное соответствие между регулярными грамматиками и конечными автоматами. Процедуры построения конечного автомата-распознавателя по регулярной грамматике или регулярному выражению достаточно подробно рассмотрены в соответствующей литературе, например, в [1; 2; 9]. В общем случае в результате построения получается недетерминированный конечный автомат, который всегда можно преобразовать в детерминированный автомат.

 При синтезе автоматов необходимо учитывать следующую особенность. Для распознавания токена лексическому анализатору может потребоваться чтение дополнительных символов входного потока, следующих за текущим символом (*опережающее чтение*). Например, встретив символ '**:**', необходимо прочитать следующий за ним символ. Если он окажется символом '**=**', то получена лексема '**:=**' токена «оператор присваивания». В противном случае символ '**:**' является лексемой токена «двоеточие», при этом прочитан один лишний символ входного потока, который следует вернуть во входной поток (этот символ может являться началом другой лексемы). Действия, связанные с определением токена и возврата лишних прочитанных символов во входной поток, легко можно реализовать в конечных состояниях соответствующих автоматов-распознавателей.

 Для облегчения реализации опережающего чтения и возврата символов во входной поток рекомендуется использовать входной буфер, из которого лексический анализатор может выполнять чтение и в который может возвращать прочитанные символы путем простого перемещения указателя. Использование входного буфера повышает также эффективность анализатора, так как считывание блока символов обычно существенно более эффективно, чем посимвольное считывание.

 Вместо построения отдельных автоматов-распознавателей для каждого ключевого слова можно использовать специальную таблицу ключевых слов. Ключевыми словами являются определенные заранее строки символов, которые удовлетворяют правилам образования идентификаторов. Чтобы отличить ключевые слова от идентификаторов, в большинстве языков программирования их делают *зарезервированными*, т. е. ключевые слова нельзя использовать в качестве идентификаторов. Если лексический анализатор распознает лексему как идентификатор, то он простым просмотром таблицы ключевых слов может определить, является эта лексема ключевым словом или нет. Таким образом, таблица ключевых слов является статической и необходимо выбрать структуру данных для ее представления и алгоритмы поиска, обеспечивающие наибольшую эффективность работы (хеширование или, по крайней мере, бинарный поиск [6]).

 После синтеза всех автоматов-распознавателей для всех токенов, а также автомата для пропуска комментариев и не несущих смысловой нагрузки пробелов, символов табуляции и символов перевода строки наиболее предпочтительным подходом является объединение всех автоматов в один детерминированный конечный автомат, который выбирает наибольшую лексему, соответствующую некоторому шаблону. Тогда программная реализация лексического анализатора сводится к программной реализации этого автомата с соответствующим формированием токенов и реализацией возврата символов во входной поток при опережающем чтении в конечных состояниях автомата.

 Более подробные рекомендации по проектированию лексического анализатора рассмотрены в [10].

**Подготовка к работе**

 1. Изучить описание работы и другие теоретические сведения, касающиеся тематики выполняемой работы.

 2. Выделить конструкции и элементы учебного языка (лексемы), обрабатываемые на этапе лексического анализа, и определять для них соответствующие регулярные грамматики и/или регулярные выражения.

 3. Выделить множество лексических классов (токенов). Для каждого токена определить формальный шаблон в виде регулярной грамматики или регулярного выражения.

 4. Выбрать формат представления токенов и их атрибутов на выходе лексического анализатора.

 5. Для каждого лексического класса синтезировать распознающий детерминированный конечный автомат по соответствующей регулярной грамматике (регулярному выражению).

 6. Объединить все полученные автоматы в один автомат, реализующий лексический анализ.

 7. Выбрать или разработать структуры данных и алгоритмы обработки таблиц, создаваемых лексическим анализатором.

 8. Разработать структуру лексического анализатора, определить его функции и алгоритмы и программно его реализовать.

**Выполнение работы**

 1. Выполнить отладку всех программ, разработанных на этапе подготовки.

 2. Исследовать работу лексического анализатора, внося в исходный текст транслируемой программы различные ошибки и рассматривая реакцию на них анализатора.

 3. Продемонстрировать работу лексического анализатора преподавателю.

**Контрольные вопросы**

1. Что такое лексема, лексический класс, токен?
2. Основные задачи лексического анализа.
3. Регулярные грамматики и конечные автоматы.
4. Регулярные выражения и конечные автоматы.
5. Преобразование недетерминированного конечного автомата в детерминированный конечный автомат.
6. Статические таблицы, используемые в сканере.
7. Динамические таблицы, используемые в сканере.
8. Способы организации таблицы символов.
9. Что такое поиск с хешированием?
10. Входные и выходные данные сканера.