**Лабораторная работа № 2  
Проектирование лексического анализатора**

**Цель работы:** изучение основных понятий теории регулярных грамматик, ознакомление с назначением и принципами работы лексических анализаторов (сканеров), получение практических навыков построения сканера на примере заданного простейшего входного языка.

Для выполнения лабораторной работы требуется написать программу, которая выполняет лексический анализ входного текста в соответствии с заданием и порождает таблицу лексем с указанием их типов и значений. Текст на входном языке задается в виде символьного (текстового) файла. Программа должна выдавать сообщения о наличие во входном тексте ошибок, которые могут быть обнаружены на этапе лексического анализа.

Длину идентификаторов и строковых констант считать ограниченной 32 символами. Программа должна допускать наличие комментариев неограниченной длины во входном файле. Форму организации комментариев выбрать самостоятельно.

**Лексический анализатор** — это компонент компилятора или интерпретатора, который отвечает за разбор исходного текста программы на элементы, называемые лексемами. Лексема — это последовательность символов, которая имеет смысл в контексте языка программирования. В процессе лексического анализа исходный текст разбивается на такие лексемы, как ключевые слова, идентификаторы, литералы (константы), операторы и другие значимые компоненты языка.

Основные задачи, которые выполняет лексический анализатор:

1. **Разделение на токены**: лексический анализатор читает исходный текст программы и делит его на отдельные компоненты, называемые токенами. Например, в выражении a = 5 + b токенами будут a, =, 5, +, и b.
2. **Игнорирование пробелов и комментариев**: лексический анализатор обычно игнорирует пробелы, табуляции и комментарии, которые не имеют значения для синтаксического анализа.
3. **Проверка синтаксиса токенов**: лексический анализатор проверяет, что токены соответствуют правилам лексического анализа, определенным для языка программирования. Например, он проверяет, что идентификаторы состоят только из допустимых символов.
4. **Выдача ошибок**: если лексический анализатор обнаруживает ошибки, такие как недопустимые символы или неверно оформленные идентификаторы, он генерирует сообщения об ошибках.
5. **Передача токенов в синтаксический анализатор**: после анализа текста лексический анализатор передает токены следующему компоненту компилятора или интерпретатора — синтаксическому анализатору, который будет использовать их для построения синтаксического дерева или другой структуры данных, отражающей структуру программы.

Пример работы лексического анализатора:

int main() {

int a = 5;

printf("Hello, world!");

return 0;

}

После лексического анализа:

* int → ключевое слово (токен типа "Ключевое слово");
* main → идентификатор;
* ( → открывающая скобка;
* ) → закрывающая скобка;
* { → открывающая фигурная скобка;
* int → ключевое слово;
* a → идентификатор;
* = → оператор присваивания;
* 5 → целое число (литерал);
* ; → точка с запятой;
* printf → идентификатор;
* ( → открывающая скобка;
* "Hello, world!" → строковая константа;
* ) → закрывающая скобка;
* ; → точка с запятой;
* return → ключевое слово;
* 0 → целое число (литерал);
* } → закрывающая фигурная скобка.

В результате лексический анализатор преобразует исходный текст в последовательность токенов, которые затем обрабатываются синтаксическим анализатором.

**Порядок выполнения работы**

Написать программу для решения задачи согласно варианту.

Вариант лабораторной работы определяется преподавателем с помощью генератора случайных чисел.

**Варианты заданий**

1. Входной язык содержит арифметические выражения, разделенные символом ;(точка с запятой). Арифметические выражения состоят из идентификаторов, десятичных чисел с плавающей точкой (в обычной и логарифмической форме), знаков операций и скобок.
2. Входной язык содержит арифметические выражения, разделенные символом ;(точка с запятой). Арифметические выражения состоят из идентификаторов, римских чисел, знаков операций и скобок. (Римскими считать числа, записанные большими буквами X, V и I).
3. Входной язык содержит упрощенные операторы цикла типа  
   while <логическое выражение> do <оператор присваивания>; Логическое выражение может содержать идентификаторы, знаки операций сравнения, целые десятичные числа без знака, скобки и логические операции and и or. Оператор присваивания должен состоять из идентификатора, знака присваивания и целой десятичной константы без знака.
4. Входной язык содержит упрощенные условные операторы типа  
   if <логическое выражение> then <оператор присваивания> else <оператор присваивания>; (часть else в операторе может отсутствовать). Логическое выражение может содержать идентификаторы, знаки операций сравнения, целые десятичные числа без знака, скобки и логические операции and и not. Оператор присваивания должен состоять из двух идентификаторов, разделенных знаком присваивания.
5. Входной язык содержит выражения над строковыми константами, разделенные символом ;(точка с запятой). Выражения состоят из идентификаторов, строковых констант, заключенных в двойные кавычки, одиночных символов, заключенных в одинарные кавычки и знаков операции конкатенации +.
6. Входной язык содержит последовательность вызовов процедур, разделенных символом ;(точка с запятой). Вызов процедуры должен состоять из имени процедуры и списка параметров. В качестве параметров могут выступать идентификаторы, целые десятичные числа без знака, шестнадцатеричные числа, десятичные числа с плавающей точкой.
7. Входной язык содержит последовательность вызовов процедур, разделенных символом ;(точка с запятой). Вызов процедуры должен состоять из имени процедуры и списка параметров. В качестве параметров могут выступать идентификаторы, строковые константы, заключенные в двойные кавычки и одиночные символы, заключенные в одинарные кавычки.
8. Входной язык содержит последовательность вызовов процедур, разделенных символом ;(точка с запятой). Вызов процедуры должен состоять из имени процедуры и списка параметров. В качестве параметров могут выступать идентификаторы и римские цифры со знаком. (Римскими считать числа, записанные большими буквами X, V и I)
9. Входной язык содержит последовательность описаний массивов в соответствии со спецификацией языка Паскаль, разделенных символом ;(точка с запятой). Считать, что массивы могут содержать только элементы скалярных типов integer, real, byte, word и char.
10. См. вариант №9, но размер массива должен указываться с помощью римских чисел.
11. Входной язык содержит последовательность описаний записей (record) в соответствии со спецификацией языка Паскаль, разделенных символом ;(точка с запятой). Считать, что записи могут содержать только поля скалярных типов integer, real, byte, word, char и строки string с возможным указанием длины строки в квадратных скобках.
12. Входной язык содержит последовательность описаний констант в соответствии со спецификацией языка Паскаль. Константы могут быть целыми десятичными числами со знаком, целыми шестнадцатеричными числами, целыми десятичными числами с плавающей точкой, строками или символами.
13. Входной язык содержит последовательность команд ассемблера в форме:  
    <метка>: <команда> <операнд1>,<операнд2> (метка, а также один или оба операнда могут отсутствовать). В качестве операндов могут выступать регистры процессора 80х86, идентификаторы, целые десятичные числа или целые шестнадцатеричные числа. (Предусмотреть наличие не менее 6 допустимых команд).
14. Входной язык содержит выражения, разделенные символом; (точка с запятой). Выражения состоят из идентификаторов, бинарных операций (сложение, вычитание, умножение, деление) и функций (например, sqrt, log, exp). Функции могут принимать идентификаторы или числовые константы в качестве аргументов. Также поддерживаются скобки для задания приоритетов операций.
15. Входной язык содержит последовательность логических выражений, разделенных символом; (точка с запятой). Логические выражения могут включать идентификаторы, булевы значения (true, false), знаки операций сравнения (равно, больше, меньше) и логические операции (and, or, not). Кроме того, присутствуют упрощенные условные операторы if <логическое выражение> then <оператор>, где оператор может быть присваиванием или вызовом процедуры.
16. Входной язык содержит описание массивов и записей в виде текстовых строк, разделенных символом; (точка с запятой). Массивы и записи могут включать поля различных скалярных типов: целых чисел, чисел с плавающей точкой, символов и строк. В описании массивов используются только целые числа для указания размеров, в описании записей — различные скалярные типы и строки.
17. Входной язык содержит многоуровневые арифметические выражения, разделенные символом; (точка с запятой). Выражения могут включать идентификаторы, числовые константы, скобки для задания приоритетов, а также функции (например, sin, cos). Операции могут включать сложение, вычитание, умножение и деление.
18. Входной язык содержит цикл for и условные операторы if. Цикл имеет вид for <идентификатор> := <начало> to <конец> do <оператор>, а условный оператор — if <логическое выражение> then <оператор> else <оператор>. В качестве операторов могут быть использованы присваивания или вызовы процедур.
19. Входной язык содержит описание типов данных и констант, разделенных символом; (точка с запятой). Типы данных могут включать простые (целые, вещественные, строковые) и составные (массивы, записи). Константы могут быть целыми числами, строками и символами. Описание типов должно включать указание имен типов и их соответствующих значений.
20. Входной язык содержит описание и вызовы процедур и функций, разделенные символом; (точка с запятой). Описание процедуры или функции включает имя, список параметров и тело. Параметры могут быть идентификаторами, строковыми или числовыми константами. Также предусмотрены вызовы процедур и функций с аргументами, которые могут быть идентификаторами или константами.