Лабораторная работа № 2

ЛЕКСИЧЕСКИЙ АНАЛИЗ (КЛАССИФИКАЦИЯ ЛЕКСЕМ)

Цель работы: Ознакомление с назначением и принципами работы лексических анализаторов, получение практических навыков построения сканера на примере заданного входного языка.

Задача: Классификация и сохранение информации об обнаруженной лексеме.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Лексический анализ — процесс аналитического разбора входной последовательности символов на распознанные группы — *лексемы*, с целью получения на выходе идентифицированных последовательностей, называемых «*токенами*» (*token*). В простых случаях понятия «*лексема*» и «*токен*» идентичны, но более сложные анализаторы дополнительно выполняют классификацию лексем по различным типам («*идентификатор*», «*оператор*», «*разделитель*» и т.п.). Лексический анализ используется в компиляторах и интерпретаторах как предварительный этап перед синтаксическим или семантическим анализом.

Во время лексического анализа исходный текст разбивается на лексемы. При этом идентификаторы и константы, имеющие произвольную длину, заменяются символами фиксированной длины (*токенами*). Это позволяет облегчить работу последующих фаз компилятора (синтаксического и семантического анализа, оптимизации и др.).

Сканер работает с тремя группами: служебные слова и разделители языка, символические имена (идентификаторы) и литералы (числовые или символьные константы). Таблица служебных слов – это предопределенный фиксированный список, который может содержать и служебные слова и разделители.

На этапе классификации лексем самый долгий процесс – поиск в таблицах. Простейший способ организации таблиц состоит в том, чтобы

добавлять элементы в порядке их поступления. В этом случае таблица идентификаторов будет представлять собой неупорядоченный массив. Поиск искомого элемента в этом массиве представляет собой последовательное сравнение искомого элемента с очередным элементом таблицы, пока элемент не будет найден или пока не будут просмотрены все элементы. Поиск нужного элемента в неупорядоченном массиве длинны *N* будет выполняться в среднем за *N/2* сравнений.

Гораздо лучших результатов можно добиться если элементы таблицы упорядочены, а в качестве метода выбран бинарный поиск. Элемент в упорядоченном массиве длины N будет найден в среднем за $\log_2 N$ сравнений. Но за эффективность поиска приходится расплачиваться увеличением времени на извлечение элемента и пополнения таблицы на последующих этапах.

Лучших результатов можно достичь, если применить методы, связанные с использованием хэш-функций и хэш-адресации.

Результатом работы сканера является последовательность кодов лексем. Эту последовательность обычно называют таблицей стандартных символов, так как в ней хранятся стандартизованные представления лексем. Информация в этой таблице расположена в том же порядке, что и в исходной программе.

тип токена	примеры лексем	описание
<id>	**varA**	*идентификатор*
<num>	**153**	*число*
<for>	**for**	*оператор цикла*

При описании лексического анализа, важно понимать и разделять три связанных понятия.

Токен

Структура, состоящая из имени и набора необязательных произвольных атрибутов. Имя токена – абстрактный символ, представляющий тип лексической единицы, например, <ключевое слово>, <название переменной>, и т.п.

Лексема

представляет собой последовательность символов исходной программы, которая идентифицируется лексическим анализатором как экземпляр токена.

Шаблон

описание вида, который может принимать лексема токена. Лексический анализатор принимает решение о принадлежности лексемы токену на основе шаблона.

Лексический токен (или просто токен) – это строка с присвоенным и, таким образом, идентифицированным значением. Он структурирован как пара, состоящая из *имени токена* и необязательного *значения токена*. Имя токена - это категория лексической единицы. Примеры токенов:

идентификатор: имена, выбираемые программистом;

ключевое слово: зарезервированные слова языка программирования;

разделитель: символы пунктуации и парные разделители;

литерал : числовые, логические, текстовые, ссылочные символьные последовательности;

комментарий: строка, блок.

На вход лексического анализатора поступает текст исходной программы, а выходная информация передаётся для дальнейшей обработки синтаксическому анализатору. Каждой выделенной из текста лексеме сканер ставит в соответствие токен вида:

‹*имя_токена, значение_атрибута*›

<*имя_токена*>, представляет собой абстрактный символ, использующийся во время синтаксического анализа, а второй компонент, значение атрибута, указывает на запись в таблице идентификаторов, соответствующую данному токену.

Токенизация - это процесс классификации разделов строки входных символов. Полученные токены затем передаются на следующий этап компиляции, то есть в синтаксический анализатор.

Можно представить эту информацию иначе, в виде набора множеств разбитых по классам лексем. Тогда каждой входной лексеме сопоставляется пара (*тип_лексемы*, *указатель_на_информацию_о_ней*) или (*номер_класса*, *номер_в_классе*). Это позволяет избежать дополнительного поиска по таблицам на последующих этапах трансляции.

Когда класс лексемы представляет более одной возможной лексемы, лексер часто сохраняет достаточно информации для воспроизведения исходной лексемы, чтобы ее можно было использовать в семантическом анализе. Синтаксический анализатор обычно получает эту информацию из лексического анализатора и сохраняет ее в абстрактном синтаксическом дереве. Это необходимо во избежание потери информации в случае чисел и идентификаторов. Такой информацией может являться номер строки и столбца, в котором в исходном коде встретилась лексема. Также дополняется информацией о типе идентификатора, присутствие в блоке объявления или факт использования в операторах.

Лексический анализатор обычно ничего не делает с комбинациями лексем, задача, оставленная парсеру . Например, типичный лексический анализатор распознает круглые скобки как токены, но ничего не делает, чтобы гарантировать, что каждому "(" соответствует ")".

Вид представления информации после выполнения лексического анализа целиком зависит от конструкции компилятора. Но в общем виде её можно представить как таблицу лексем, которая в каждой строчке должна содержать информацию о виде лексемы, её типе и, возможно, значении. Обычно такая таблица имеет два столбца: первый – строка лексемы, второй – указатель на информацию о лексеме, может быть включён и третий столбец – тип лексем. Не следует путать таблицу лексем и таблицу идентификаторов – это две принципиально разные таблицы. Таблица лексем содержит весь текст исходной программы, обработанный лексическим анализатором на первом этапе. В неё входят все возможные лексемы, при этом, любая лексема может в ней встречаться любое число раз. Таблицы идентификаторов и литералов

содержат только следующие типы лексем: идентификаторы и константы. В неё не попадают ключевые слова входного языка, знаки операций и разделители. Каждая лексема в таблице идентификаторов может встречаться только один раз.

Для формирования типа лексемы можно использовать тип ENUM:

```
public enum TokenType {
   INT, BOOL, LITERAL, NUMBER, IDENTIFIER, BEGIN, END,
   IF, ELSE, TRUE, FALSE, LPAR, RPAR, PLUS,
   MINUS, EQUAL, SEMICOLON
}
```

Для описания токена можно создать отдельный класс, хранящий его тип и значение:

```
class Token {
   public TokenType Type;
   public string Value;
   public Token(TokenType type) {
        Type = type;
    }
    public override string ToString() {
        return string.Format("{0}, {1}", Type, Value);
    }
}
```

Заранее известные типы лексем можно объединить в массивы. Например, все разделители:

```
static TokenType[] Delimiters = new TokenType[]
   { TokenType.SEMICOLON, TokenType.PLUS, TokenType.MINUS,
     TokenType.EQUAL, TokenType.RPAR, TokenType.LPAR
};
```

Для определения, что текущая лексема является разделителем, создать метод, определяющий вхождение в список:

```
public static bool IsDelimiter(Token token) {
    return Delimiters.Contains(token.Type);
```

```
}
```

Аналогично, собрать в единый список все ключевые слова языка:

```
static Dictionary<string, TokenType> SpecialWords =
                        new Dictionary<string, TokenType>() {
            { "int", TokenType.INT },
            { "bool", TokenType.BOOL },
            { "if", TokenType.IF },
            { "else", TokenType.ELSE },
            { "{", TokenType.BEGIN },
            { "}", TokenType.END }
};
```

А проверку текущей лексемы на соответствие ключевому слову выполнять методом:

```
public static bool IsSpecialWord(string word) {
    if (string.IsNullOrEmpty(word)) {
        return false;
    }
    return (SpecialWords.ContainsKey(word));
}
```

Аналогично, собрать в единый список все символы-разделители:

```
static Dictionary<char, TokenType> SpecialSymbols =
                    new Dictionary<char, TokenType>() {
        { ';', TokenType.SEMICOLON },
        { '(', TokenType.LPAR },
        { ')', TokenType.RPAR },
        { '+', TokenType.PLUS },
        { '-', TokenType.MINUS },
        { '=', TokenType.EQUAL },
};
```

И проверять методом:

```
public static bool IsSpecialSymbol(char ch) {
    return SpecialSymbols.ContainsKey(ch);
```

```
}
```

Далее, в теле метода лексического анализа, необходимо взять выделенную в ходе предыдущей лабораторной работы лексему, и проверить на соответствию шаблону токена. Сформировать экземпляр класса Token и добавить в список токенов.

Например, для получения токена с разделителями:

```
if (IsSpecialSymbol(currentLexem)) {
   token = new Token(SpecialSymbols[currentLexem]);
   token.Value = currentLexem;
}
```

В данном случае в `currentLexem` хранится текущая лексема. Ее строковое значение сохраняется в параметре `Value`.

Или для числовых литератов

```
if (IsDigit(currentLexem)) {
   token = new Token(TokenType.NUMBER);
   token.Value = currentLexem;
}
```

После классификации типов лексем и получения токенов их необходимо сохранять в список `List<Token>`.

Итогом работы лексического анализатора должен быть список лексем, в котором каждой сопоставлен ее конкретный тип. Далее эта информация будет использована синтаксическим анализатором для проверки принадлежности грамматике.

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Дополнить лексический анализатор, подготовленный на первой лабораторной работе, выполнив классификацию выделенных лексем.

1. Продумать архитектуру системы.
2. Продумать организацию структур данных.
3. Дополнить программу лексического анализа классификатором лексем в виде токенов.

4. Выводить на форму лексемы с указанием типа каждой из них.

5. Составить тестовые наборы данных и проверить на них работу программы.

Содержание отчета

- Титульный лист
- Текст задания
- Программный код реализации задания
- Скриншоты работы программы

Вопросы для самоконтроля

1. Описать задачу классификации лексем
2. Основные типы и назначение лексем
3. Назначение таблицы стандартных символов

Список литературы

1. Шульга, Т. Э. Теория автоматов и формальных языков : учебное пособие / Т. Э. Шульга. — Саратов : Саратовский государственный технический университет имени Ю.А. Гагарина, ЭБС АСВ, 2015. — 104 с. — ISBN 987-5-7433-2968-7. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: https://www.iprbookshop.ru/76519.html (дата обращения: 15.04.2021). — Режим доступа: для авторизир. пользователей. - DOI: https://doi.org/10.23682/76519 - https://www.iprbookshop.ru/76519.html

2. Алымова, Е. В. Конечные автоматы и формальные языки : учебник / Е. В. Алымова, В. М. Деундяк, А. М. Пеленицын. — Ростов-на-Дону, Таганрог : Издательство Южного федерального университета, 2018. — 292 с. — ISBN 978-5-9275-2397-9. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: https://www.iprbookshop.ru/87427.html (дата обращения: 15.04.2021). — Режим доступа: для авторизир. пользователей - https://www.iprbookshop.ru/87427.html

3. Пентус, А. Е. Математическая теория формальных языков : учебное пособие / А. Е. Пентус, М. Р. Пентус. — 3-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. — 218 с. — ISBN 978-5-4497-0662-1. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: https://www.iprbookshop.ru/97548.html (дата обращения: 15.04.2021). — Режим доступа: для авторизир. пользователей - https://www.iprbookshop.ru/97548.html

4. Миронов, С. В. Формальные языки и грамматики : учебное пособие для студентов факультета компьютерных наук и информационных технологий / С. В. Миронов. — Саратов : Издательство Саратовского университета, 2019. — 80 с. — ISBN 978-5-292-04613-4. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: https://www.iprbookshop.ru/99047.html (дата обращения: 15.04.2021). — Режим доступа: для авторизир. пользователей - https://www.iprbookshop.ru/99047.html

5. Малявко, А. А. Формальные языки и компиляторы : учебник / А. А. Малявко. — Новосибирск : Новосибирский государственный технический университет, 2014. — 431 с. — ISBN 978-5-7782-2318-9. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: https://www.iprbookshop.ru/47725.html (дата обращения: 15.04.2021). — Режим доступа: для авторизир. пользователей - https://www.iprbookshop.ru/47725.html