МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

филиал федерального государственного бюджетного образовательного учреждения высшего образования «Национальный исследовательский университет

«Московский энергетический институт»

в г. Смоленске

Кафедра вычислительной техники

**Лабораторная работа № 1**

по курсу: «Трансляторы программных языков»

Тема: «Лексический анализ входного языка транслятора»

Группа: ПО-19

Преподаватель:

Смоленск, 2022 г.

Введение

Данная лабораторная работа предназначается для студентов, изучающих «Трансляторы программных языков». Лабораторная работа рассчитана на 4 аудиторных часа и 6 часов самостоятельной работы по составлению программы, изучение литературы и составление отчёта.

Объект исследования – трансляторы с алгоритмических языков программирования.

Процесс трансляции с алгоритмического языка можно условно разбить на три этапа: лексический анализ, грамматический разбор и генерацию машинного кода. В данной работе рассматривается задача построения *лексического анализатора* входного текста транслятора.

**Цель работы** состоит в составлении программы (сканера) производящей лексический анализ текста, соответствующего заданному алфавиту и грамматике алгоритмического языка.

Программа составляется на алгоритмических языках программирования высокого уровня (Паскаль, С++, С#, Python) по выбору студента.

# 1. Содержание работы.

Этап лексического анализа текста исходной программы выделяется в самостоятельный этап работы транслятора, как с методической целью, так и с целью сокращения времени компиляции программы. Последнее достигается за счёт того, что исходная программа в виде последовательности символов, преобразуется на этапе лексической обработки к некоторому стандартному виду, что облегчает дальнейший анализ.

Под *лексическим анализом* понимают процесс предварительной обработки исходной программы, на котором основные лексические единицы программы – **лексемы**: ключевые слова, идентификаторы, метки, константы приводятся к единому формату и заменяются условными кодами или ссылками на соответствующие таблицы, а комментарии исключаются из текста программы.

Результатом лексического анализа является список лексем-дескрипторов и таблицы. В таблицах хранятся значения выделенных в программе лексем.

**Дескриптор** – это пара вида: ( <тип лексемы> . < указатель>),

где <тип лексемы> – это, как правило, числовой код класса лексемы, который означает, что лексема принадлежит одному из конечного множества классов слов, выделенных в языке программирования;

<указатель> – это может быть либо начальный адрес области основной памяти, в которой хранится адрес этой лексемы, либо число, адресующее элемент таблицы, в которой хранится значение этой лексемы.

Количество классов лексем в языках программирования может быть различным. Наиболее распространёнными классами являются:

* идентификаторы;
* служебные (ключевые) слова;
* разделители (включают знаки операций);
* константы.

Могут вводиться и другие классы.

Это обусловлено в первую очередь той ролью, которую играют различные виды слов при написании исходной программы и переводе её в машинную программу. При этом наиболее предпочтительным является разбиение всего множества слов, допускаемых в языке программирования, на такие классы, которые бы не пересекались между собой.

В общем случае все выделяемые классы являются либо конечными (ключевые слова, разделители и др.) – классы фиксированных для данного языка программирования слов, либо бесконечными или очень большими (идентификаторы, константы, метки) – классы переменных для данного языка программирования слов.

С этих позиций коды лексем (дескрипторы) из конечных классов всегда одни и те же в различных программах для данного компилятора. Коды же лексем из бесконечных классов различны для разных программ и формируются всякий раз на этапе лексического анализа.

В ходе лексического анализа значения лексем из бесконечных классов помещаются в таблицы соответствующих классов. Конечность таблиц объясняет ограничения, существующие в языках программирования на длины и соответственно число используемых в программе идентификаторов и констант.

Числовые константы перед помещением их в таблицу могут переводиться из внешнего символьного во внутреннее машинное представление.

Содержимое таблиц, в особенности таблицы идентификаторов, в дальнейшем пополняется на этапе семантического анализа исходной программы и используется на этапе генерации объектной программы.

**В работе требуется** составить программу лексического анализатора (сканер) входного текста для транслятора:

* составляющую таблицы и проводящую кодирование идентификаторов, разделителей и констант.
* Производящую проверку правильности написания ключевых слов операторов, стандартных функций и использование служебных символов.
* Производящую отображение теста программы с комментариями и исключающую их из текста, подлежащего трансляции.
* Отображающую дескрипторный текст.

# 2. Задание по работе.

2.1. Получить вариант задания.

2.2. В соответствии с выданным вариантом выполнить следующее:

2.2.1. Составить техническое задание (ТЗ) на разработку программы сканера, производящей лексический анализ произвольных текстов в пределах установленного алфавита.

2.2.2. Согласовать ТЗ с преподавателем.

2.2.3. Разработать программу-сканер на языках высокого уровня или в интегрированных средах по собственному усмотрению.

2.2.4. Провести тестирование программы, особенно для всех случаев выдачи пользователю сообщений об ошибках.

2.2.5. Составить отчёт по работе и приложить к нему ТЗ.

# 3. Варианты заданий.

## Операции языка (первая цифра варианта) представлены в таблицах 5.1 – 5.4.

Таблица 5.1 - Операции группы «отношение»

|  |  |
| --- | --- |
| Номер | Синтаксис группы операций (в порядке следования: неравно, равно, меньше, меньше или равно, больше, больше или равно) |
| 1 | <операции\_группы\_отношения>:: = < > | = | < | <= | > | >= |

Таблица 5.2 - Операции группы «сложение»

|  |  |
| --- | --- |
| Номер | Синтаксис группы операций (в порядке следования: сложение, вычитание, дизъюнкция) |
| 1 | <операции\_группы\_сложения>:: = + | - | *or* |

Таблица 5.3 - Операции группы «умножение»

|  |  |
| --- | --- |
| Номер | Синтаксис группы операций(в порядке следования: умножение, деление, конъюнкция) |
| 1 | <операции\_группы\_умножения>::= \* | / | *and* |

Таблица 5.4 - Унарная операция

|  |  |
| --- | --- |
| Номер | Синтаксис операции |
| 1 | <унарная\_операция>::= *not* |

Выражения языказадаются правилами:

<выражение>::= <операнд>{<операции\_группы\_отношения> <операнд>}

<операнд>::= <слагаемое> {<операции\_группы\_сложения> <слагаемое>}

<слагаемое>::= <множитель> {<операции\_группы\_умножения> <множитель>}

<множитель>::= <идентификатор> | <число> | <логическая\_константа> |

<унарная\_операция> <множитель> | (<выражение>)

<число>::= <целое> | <действительное>

<логическая\_константа>::= *true* | *false*

## Правила, определяющие идентификатор, букву и цифру:

<идентификатор>::= <буква> {<буква> | <цифра>}

<буква>::= *A* | *B* | *C* | *D* | *E* | *F* | *G* | *H* | *I* | *J* | *K* | *L* | *M* | *N* | *O* | *P* | *Q* | *R* | *S* | *T* |

*U* | *V* | *W* | *X* | *Y* | *Z* | *a* | *b* | *c* | *d* | *e* | *f* | *g* | *h* | *i* | *j* | *k* | *l* | *m* | *n* | *o* | *p*

*q* | *r* | *s* | *t* | *u* | *v* | *w* | *x* | *y* | *z*

<цифра>::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Правила, определяющие целые числа:

<целое>::= <двоичное> | <восьмеричное> | <десятичное> |

<шестнадцатеричное>

<двоичное>::= {/ 0 | 1 /} (*B* | *b*)

<восьмеричное>::= {/ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 /} (*O* | *o*)

<десятичное>::= {/ <цифра> /} [*D* | *d*]

<шестнадцатеричное>::= <цифра> {<цифра> | *A* | *B* | *C* | *D* | *E* | *F* | *a* | *b* |

*c* | *d* | *e* | *f*} (*H* | *h*)

Правила, описывающие действительные числа:

<действительное>::= <числовая\_строка> <порядок> |

[<числовая\_строка>] . <числовая\_строка> [порядок]

<числовая\_строка>::= {/ <цифра> /}

<порядок>::= ( E | e )[+ | -] <числовая\_строка>

Правила, определяющие структуру программы (вторая цифра варианта), представлены в таблице 5.5.

Таблица 5.5 – Структура программы

|  |  |
| --- | --- |
| Номер | Структура программы |
| 3 | <программа> = {/ (<описание> | <оператор>) переход строки /} *end* |

Правила, определяющиераздел описания переменных (третья цифра варианта) показаны в таблице 5.6.

Таблица 5.6 - Синтаксис команд описания данных

|  |  |
| --- | --- |
| Номер | Синтаксис команд описания данных |
| 2 | <описание>::= *dim* <идентификатор> {, <идентификатор> } <тип> |

## Правила, определяющие типы данных (четвертая цифра варианта) представлены в таблице 5.7.

Таблица 5.7- Описание типов данных

|  |  |
| --- | --- |
| Номер | Описание типов  (в порядке следования: целый, действительный, логический) |
| 1 | <тип>::= % | ! | $ |

## Правило, определяющее оператор программы (пятая цифра варианта).

<оператор>::= <составной> | <присваивания> | <условный> |

<фиксированного\_цикла> | <условного\_цикла> | <ввода> | <вывода>

Составной операторописан в таблице 5.8.

Таблица 5.8 - Синтаксис составного оператора

|  |  |
| --- | --- |
| Номер | Синтаксис оператора |
| 1 | <составной>::= <оператор> { : <оператор> } |

Оператор присваиванияописан в таблице 5.9.

Таблица 5.9 - Синтаксис оператора присваивания

|  |  |
| --- | --- |
| Номер | Оператор присваивания |
| 1 | <присваивание>::= <идентификатор> *ass* <выражение> |

Оператор условного перехода задан в таблице 5.10.

Таблица 5.10 - Синтаксис оператора условного перехода

|  |  |
| --- | --- |
| Номер | Оператор условного перехода |
| 1 | <условный>::= *if* <выражение> *then* <оператор> [ *else* <оператор>] |

Оператор цикла с фиксированным числом повторений описан в таблице 5.11.

Таблица 5.11 - Синтаксис оператора цикла с фиксированным числом повторений

|  |  |
| --- | --- |
| Номер | Синтаксис оператора |
| 1 | <фиксированного\_цикла>::= *for* <присваивания> *to* <выражение> *do* <оператор> |

Условный оператор циклазадан в таблице 5.12.

Таблица 5.12 - Синтаксис условного оператора цикла

|  |  |
| --- | --- |
| Номер | Синтаксис оператора |
| 1 | <условного\_цикла>::= *while* <выражение> *do* <оператор> |

Оператор вводаописан в таблице 5.13.

Таблица 5.13 - Синтаксис оператора ввода

|  |  |
| --- | --- |
| Номер | Синтаксис оператора |
| 1 | <ввода>::= *read* (<идентификатор> {, <идентификатор> }) |

Оператор выводапредставлен в таблице 5.14.

Таблица 5.14 - Синтаксис оператора вывода

|  |  |
| --- | --- |
| Номер | Синтаксис оператора |
| 1 | <вывода>::= *write* (<выражение> {, <выражение> }) |

Многострочные комментарии в программе(шестая цифра варианта) определены в таблице 5.15. Индивидуальные номера вариантов представлены в таблице 5.16.

Таблица 5.15 – Синтаксис многострочных комментариев

|  |  |  |
| --- | --- | --- |
| Номер | Признак начала комментария | Признак конца комментария |
| 1 | \\* | \*\ |

Текст программы должен допускать использование комментариев.

# 4. Методические указания.

Рассмотрим основные идеи, которые лежат в основе построения

лексического анализатора, и проблемы, возникающие при его разработке.

Первоначально в тексте входной программы сканер выделяет последовательность символов, которая по его предположению должна быть словом в программе, т.е. лексемой. Может выделяться не вся последовательность, а только один символ, который считается началом лексемы. Это сделать просто, если слова в программе отделяются друг от друга специальными разделителями,

например, пробелами или запрещено использование служебных слов в качестве переменных, либо классы лексем распознаются по вхождению первых символов лексемы.

Затем, проводится идентификация лексемы. Она заключается в сборке лексемы из символов, начиная с выделенного на предыдущем этапе, и проверки правильности записи лексемы данного класса.

Идентификация лексемы из конечного класса выполняется путём сравнения её с эталонным значением. Основная проблема здесь - минимизация времени поиска эталона. В общем случае может понадобиться полный перебор слов данного класса, особенно, если выделенное слово содержит ошибку. Уменьшить время поиска можно, используя различные методы ускоренного поиска: упорядоченный список, линейный список, метод расстановки и др.

Для идентификации из очень больших классов используются специальные методы сборки лексем с одновременной проверкой правильности написания. В этих методах применяется формальный математический аппарат − теория регулярных языков и конечных распознавателей.

При успешной идентификации значение лексемы из бесконечного класса помещается в таблицу идентификации лексем данного класса. При этом осуществляют проверку: не хранится ли уже там значение данной лексемы, т.е. необходимо проводить просмотр элементов таблицы. Таблица при этом должна допускать расширение. Опять же для уменьшения времени доступа к элементам таблицы она должна быть специальным образом организована, при этом должны использоваться специальные методы ускоренного поиска элементов.

После проведения успешной идентификации лексемы формируется её образ - дескриптор, он помещается в выходные данные лексического анализатора. В случае неуспешной идентификации формируется сообщение об ошибках в написании слов программы.

В ходе лексического анализа могут выполняться и другие виды лексического контроля, в частности, проверяться парность скобок и других парных символов.

Результаты работы сканера передаются в последствии на вход синтаксического анализатора. Имеется две возможности их связи:

раздельная связь и нераздельная связь.

При раздельной связи выходные данные сканера формируются полностью и затем передаются синтаксическому анализатору. При нераздельной связи, когда синтаксическому анализатору требуется очередной образ лексемы, он вызывает лексический анализатор, который генерирует дескриптор и возвращает управление синтаксическому анализатору.

Второй вариант характерен для однопроходных трансляторов.

Таким образом, процесс лексического анализа достаточно прост, но может занимать значительное время трансляции.

Рассмотрим конкретный пример. Пусть нам дана программа на некотором алгоритмическом языке:

PROGRAM PRIMER;

VAR X,Y,Z : REAL;

BEGIN

X:=5;

Y:=6;

Z:=X+Y;

END;

Применим следующие коды для типов лексем:

К1- ключевое слово;

К2- разделитель;

К3- идентификатор;

К4- константа.

Лексический анализ можно производить, если нам задан алфавит,

список ключевых слов языка и служебных символов. Пусть всё это

имеется. Тогда внутренние таблицы сканера примут следующий вид.

Таблица 4. Ключевые слова.

|  |  |
| --- | --- |
| № | Ключевое слово |
| 1 | PROGRAM |
| 2 | BEGIN |
| 3 | END |
| 4 | FOR |
| 5 | REAL |
| 6 | VAR |

Таблица 5. Разделители.

|  |  |
| --- | --- |
| № | Разделители |
| 1 | ; |
| 2 | , |
| 3 | + |
| 4 | - |
| 5 | / |
| 6 | \* |
| 7 | : |
| 8 | = |
| 9 | . |

Результат работы сканера таблица идентификаторов и таблица констант

Таблица 6. Идентификаторы.

|  |  |
| --- | --- |
| № | Идентификаторы |
| 1 | PRIMER |
| 2 | X |
| 3 | Y |
| 4 | Z |

Таблица 7. Константы.

|  |  |
| --- | --- |
| № | Знач. констант |
| 1 | 5 |
| 2 | 6 |

## На основании составленных таблиц можно записать входной текст через введённые дескрипторы (дескрипторный текст):

( К1, 1) (К3, 1) (K2, 1)

( K1, 6) (K3, 2) (K2, 2) (k3, 3) ( K2, 2) (K3, 4) ( K2, 7) (K1, 5) (K2, 1)

( K1, 2)

( K3, 2) (K2, 7) (K2, 8) (K4, 1) (K2, 1)

( K3, 3) (K2, 7) (K2, 8) (K4, 2) (K2, 1)

( K3, 4) (K2, 7) (K2, 8) (K3, 2) (K2, 3) (K3, 3) (K2, 1)

( K1, 3) (K2, 9).

# 6. Содержание отчёта.

1. Титульный лист.

2. Вариант задания.

3. Полный список выбранных ключевых слов и стандартных функций.

4. Внутренние таблицы сканера.

5. Техническое задание на разработку сканера (по ЕСПД).

6. Отладочные примеры работы сканера с выходными таблицами и дескрипторным текстом.

# 7. Контрольные вопросы.

1. Дайте определение грамматики.

2. Назовите этапы трансляции программы.

3. Что такое лексема?

4. В чём состоят задачи лексического анализа?

5. Дайте определение метаязыка.

6. Исходные данные для сканера.

7. Результаты работы сканера.