Лабораторная работа №1. Разработка блока лексического анализа транслятора

Цель: изучение методов лексического анализа с их программной реализацией.

Задачи:

1. Изучение теоретического материала по организации лексического анализа языков программирования.
2. Составление формального описания программы лексического анализа.
3. Программная реализация по формальному описанию.

Ход работы:

1. Получить у преподавателя индивидуальный вариант задания, предусматривающий разработку программы лексического анализа простого языка программирования. См. приложение А.
2. Составить формальное описание программы лексического анализа с использованием математического аппарата регулярных выражений.
3. Произвести автоматическую генерацию исходного текста программы лексического анализа с использованием системы flex. Отладить полученную программу.
4. Написать отчет.

Ход защиты:

1. Продемонстрировать преподавателю корректную работу программы лексического анализа.
2. Пояснить работу изученных механизмов по полученному формальному описанию программы лексического анализа. При необходимости внести в него изменения в соответствии с требованиями преподавателя.

Во время защиты лабораторной работы необходимо иметь при себе исполняемый модуль программы, исходные тексты, файлы с тестовыми примерами, а также твердую копию отчета.

Содержание отчета:

1. Цель работы с постановкой задачи.
2. Полученная спецификация flex в соответствии с полученным заданием.
3. Тестовые примеры работы программы[1].

Краткие теоретические сведения

В процессе трансляции программы на фазе лексического анализа происходит формирование символов (лексем) языка из последовательностей знаков, составляющих текст программы. Со сформированными символами

[1] Не менее трех тестовых примеров

языка идет дальнейшая работа на последующих фазах процесса трансляции. Кроме того, на фазе лексического анализа обрабатываются пробелы, удаляются комментарии и другие символы, не имеющие смысла для следующих фаз.

Лексический анализатор – это часть транслятора, которая получает на вход исходный текст программы и выделяет в нем лексемы. Лексический анализатор работает с объектами, подобными идентификаторам или константам. Язык идентификаторов и констант является регулярным, следовательно, может задаваться регулярными выражениями, формально определяемыми так. Если A – алфавит, то к числу регулярных выражений относятся:

– пустая строка ε;

– любой элемент A.

При этом если P и Q – регулярные выражения, то регулярными являются и следующие выражения:

– PQ (или $P \bullet Q$) – Q следует за P (операция конкатенации);

– $P \mid Q$ – P или Q (операция «или»);

– P^* – нуль или более вхождений P (операция «итерация», или «звездочка Клини»).

Иногда используется P^+, что обозначает одно или более вхождений P, но данная операция («усеченная итерация») может быть выражена через три предыдущих.

При записи регулярных выражений могут использоваться круглые скобки, как и для обычных арифметических выражений. При отсутствии скобок операции выполняются слева направо с учетом приоритетов. Звездочка Клини имеет наивысший приоритет, ниже – конкатенация, а операция «или» имеет самый низкий приоритет. Например, идентификатор в большинстве языков программирования может быть представлен регулярным выражением

```
L(L|D)*
```

L обозначает букву, а D – цифру. Аналогично, можно определить, например, действительное число.

```
(+|-| )D*.DD*
```

или, эквивалентно,

```
(+|-|)D*.D+.
```

Простота и четкая формулировка целей фазы лексического анализа позволяют легко ее автоматизировать. В настоящее время известно несколько инструментальных средств автоматизации. Наиболее популярным среди них является Lex. Мы воспользуемся его бесплатным вариантом flex.

В целом flex использует форму записи, похожую на регулярные выражения, но с некоторыми отличиями. Чтобы определить идентификатор в обозначениях flex, сначала определяются нетерминалы *letter* и *digit*.

```
digit       [0-9]
letter      [a-z]
```

Это **определения** в flex. Не обязательно перечислять каждый знак из необходимого диапазона, достаточно указать границы. Далее определяется идентификатор.

```
identifikator       {letter}({letter}|{digit})*
```

Здесь у круглых скобок, операций «или» и «итерация» тот же смысл, что в регулярных выражениях. Фигурные скобки используются для обособления определенных выше величин. Если некое действие должно выполняться при каждом распознавании идентификатора, то оно выражается в виде **правила**:

```
{identifikator}  { printf("Распознан идентификатор\n"); }
```

Общий вид входа, ожидаемого flex:

```
определения
%%
правила
%%
функции пользователя
```

Из этих частей, разделенных соответствующими находящимися в крайнем левом положении двойными символами, вторая является обязательной. Таким образом, полным входом для лексического анализатора, распознающего идентификаторы, будет следующий.

```
digit       [0-9]
letter      [a-z]
identifikator       {letter}({letter}|{digit})*
%%
{identifikator}  { printf("Распознан идентификатор\n"); }
%%
```

Далее приведены наиболее часто используемые шаблоны flex, которые используются для расширенных регулярных выражений.

```
a             отдельный знак
\a            a, если он знак из системы обозначений Flex
.             любой знак, кроме символа новой строки
[abc]         набор знаков a или b или c
[j-o]         набор знаков из диапазона от j до o
[^j-o]        набор без знаков из диапазона от j до o
b?            нуль или одно вхождение b
b+            одно или более вхождений b
b*            нуль или более вхождений b
a|b           a или b
b{q,p}        от q до p вхождений b
b{q,}         q или более вхождений b
b{q}          точно q вхождений b
{expr}        выражение, определенное expr
(b)           b, скобки для указания приоритета
ab\xy         ab, которое находится перед xy
^b            b в начале строки
b$            b в конце строки
\X            Если X это ‘a’, ‘b’, ‘f’, ‘n’, ‘r’, ‘t’ или ‘v’,
```

```
              то это интерпретируется как '\x'
              в языке C. Иначе – обычный символ 'X',
              экранированный символом обратной косой черты.
<<EOF>>       конец входного потока
[:alnum:]     синоним для алфавитно-цифрового символа; об этом и
              аналогичных символах см. в руководстве пользователя.
```

Функция *yylex*() – это лексический анализатор, который можно вызывать из программы на языках C/C++. Вызов указывается в разделе **пользовательских функций**:

```
%%
// отказываемся от использования функции noyywrap()
%option noyywrap
...
%%
int main()
{
    return yylex();
...
}
```

Как правило, *yylex*() вызывается функцией *main*(), если лексический анализатор самодостаточен, либо функций *yyparse*(), представляющей синтаксический анализатор, построенный в программах yacc/bison.

Если входная спецификация flex находится в файле с именем *mylexer.l*, то исходный текст лексического анализатора можно сгенерировать следующей командной строкой:

```
$flex mylexer.l
```

Здесь $ – это подсказка операционной системы для ввода команд.

На выходе будет получен файл с именем, похожим на *yy.lex.c* (в зависимости от используемой версии flex возможны разные варианты), содержащий помимо прочего кода требуемую функцию *yylex*(). Далее этот файл нужно откомпилировать и при отсутствии ошибок получить исполняемый файл лексического анализатора. Справка по запуску flex из командной строки:

```
$flex -h
```

или

```
$flex --help
```

Для полноты картины попробуем с использованием flex построить лексический анализатор языка со следующими конструкциями. Его элементами являются десятичные целочисленные константы, имена переменных (последовательность букв латинского алфавита, десятичных цифр и символов подчеркивания, всегда начинающаяся с буквы), строковые литералы (последовательность символов, заключенная в кавычки), арифметические

операторы (+, -, *, /, %) и операторы сравнения (>, <, >=, <=, <>, =), ключевые слова *auto*, *break*, *case*, *char*, *const*, *continue*, *default*, *do*, *double*, *else*, *extern*, *float*, *for*, *goto*, *if*, *int*, *return*, *switch*, *void*, *while*, а также знаки пунктуации (запятая, точка, точка с запятой, круглые скобки, фигурные скобки). Пробельные символы значимыми не считаются. Лексический анализатор получает на вход исходный текст на описанном языке и выводит на экран таблицу лексем с двумя колонками — класс лексемы и ее текстовое представление, а также обеспечивает обработку лексических ошибок — отсутствие знака в алфавите входного языка и незакрытый строковый литерал.

```
%{
#if defined _WIN32
#include <io.h>              // Для isatty
#elif defined _WIN64
#include <io.h>              // Для isatty
#endif
#ifdef MSVC
#define  isatty _isatty      // В VC isatty назван _isatty
#endif

#include <stdio.h>
#include <stdlib.h>

%}

%option nounistd

stringch      [^"\n]
%%

        /* Пропускаются все пробельные символы.
         *  Аналогично будет выглядеть код для комментариев.
         */
[ \t\n]         { }

        /* Целочисленные константы. */
0|[1-9][0-9]*    { printf("%-20s%-20s\n", "CONSTANT", yytext); }

        /* Ключевые слова. */

auto|break|case|char|const|continue
                              { printf("%-20s%-20s\n", "KEYWORD", yytext); }
default|do|double|else|extern
                              { printf("%-20s%-20s\n", "KEYWORD", yytext); }
float|for|goto|if|int|return
                              { printf("%-20s%-20s\n", "KEYWORD", yytext); }
switch|void|while
                              { printf("%-20s%-20s\n", "KEYWORD", yytext); }

        /* Знаки пунктуации. */

";"|","|"."|"("|")"|"{"|"}"         { printf("%-20s%-20s\n"
                                            , "PUNCTUATOR", yytext
                                           );
                                    }

        /* Знаки операций. */

"+"|"-"|"*"|"/"|"%"
                             { printf("%-20s%-20s\n", "OPERATOR", yytext); }
">="|"<="|"<"|">"|"="|"<>"
```

```
                            { printf("%-20s%-20s\n", "OPERATOR", yytext); }

        /* Идентификаторы. Это правило должно идти после, но ни в коем
         * случае до шаблона, представляющего то, что может быть
         * идентификатором, например, ключевые слова.
         */
[a-zA-Z][a-zA-Z0-9_]* { printf("%-20s%-20s\n", "IDENTIFIER", yytext); }

        /* Строковые литералы. */
        /* Сначала определяется ошибочный литерал, в котором отсутствует
         * закрывающая кавычка. По аналогичной схеме можно решить проблему
         * незакрытого комментария.
         */

\"{stringch}*$  {
                printf("Error at line %d: Unfinished string literal - %s.\n"
                       , yylineno, &yytext[1]
                           );
           }

        /* Строковый литерал без ошибки с незакрытой кавычкой. */

\"{stringch}*\" {
                  char string[256] = "";
                  int len = strlen(yytext)-2;
                  strncpy(string, &yytext[1], len);
                  (len > 0) ? printf("%-20s%-20s\n", "STRING", string)
                            : printf("EMPTY STRING CONSTANT\n")
                            ;
                }

        /* Все остальное. */
        /* Элемент, не принадлежащий алфавиту языка, приводит к ошибке. */
.       {
            printf("Error at line %d: %s - not in alphabet.\n"
                   , yylineno, yytext
                  );
        }
%%
int yywrap () { return 1; }

int main(int argc, char* argv[])
{
    FILE* infile;    /* Структура для входного файла. */

    /* Проверка на правильность запуска лексического анализатора
     * из командной строки.
     */
    if(argc < 2)
    {
        printf("Too few parameters in command line.\n");
        return 1;
    }
    else if (argc > 2)
    {
        printf("Too much parameters in command line.\n");
        return 1;
    }

    /* Первый параметр командной строки – имя входного файла */
    infile = fopen(argv[1], "r");
    if (NULL == infile)
    {
        printf("Cannot open file %s.\n");
```

```
        return 1;
    }
    yyin = infile;
    yylex();

    fclose(infile);

    return 0;
}
```

Использованы следующие источники:

1. Костельцев, А.В. Построение интерпретаторов и компиляторов / А.В. Костельцев. – СПб.: Наука и Техника, 2001. – 224 с.
2. Levine, J. lex & yacc, Second Edition / J. Levine, T. Mason, D. Brown. – Sebastopol, CA: O'Reilly Media, 1992. – 384 p.
3. Levine, J. flex & bison / J. Levine. – Sebastopol, CA: O'Reilly Media, 2009. – 292 p.
4. http://epaperpress.com/lexandyacc/index.html
5. Компиляция. 1: лексер - http://habrahabr.ru/blogs/programming/99162/

Общая постановка задачи и варианты заданий

Общие требования:

1) Разработать программу, осуществляющую лексический анализ простого языка программирования.

2) На вход лексического анализатора подается внешний файл с текстом программы на языке, описанном в конкретном варианте задания.

3) На выходе лексический анализатор должен выдавать таблицу всех лексем с важнейшими характеристиками: тип/класс лексемы; ее литеральное написание; порядковый номер строки, содержащей лексему.

4) Используется интерфейс командной строки, т.е. анализатор в общем случае запускается так:

```
user$: lexer -parameters input.lng
```

здесь *user$* – подсказка командной строки; *lexer* – имя исполняемого модуля программы лексического анализа; *-parameters* – 0 или более параметров командной строки; *input.lng* – пример имени внешнего файла с программой на анализируемом языке.

5) При выводе информации об ошибке обязательно указание на номер строки в исходном файле.

6) Диагностируются только лексические ошибки.

Вариант 1. Входной язык содержит арифметические выражения, разделенные символом точки с запятой (';'). Арифметические выражения состоят из идентификаторов, десятичных чисел с плавающей запятой[1] в обычном и нормализованном экспоненциальном форматах, а также целочисленных констант. Кроме того, элементами арифметических выражений являются знаки присваивания ('='), знаки операций ('+', '-', '*', '/', '%') и круглые скобки.

[1] http://ru.wikipedia.org/wiki/Плавающая_запятая

Вариант 2. Входной язык содержит логические выражения, разделенные символом точки с запятой (';'). Логические выражения состоят из идентификаторов, нечувствительных к регистру констант *true* и *false*, знаков присваивания ('='), знаков операций *or*, *xor*, *and*, *not* и круглых скобок.

Вариант 3. Входной язык содержит операторы условий if-без-else и if-else, разделенные символом точки с запятой (';'). Операторы условий состоят из идентификаторов, десятичных чисел с плавающей запятой[1] в обычном и нормализованном экспоненциальном форматах, знаков присваивания ('='), знаков операций сравнения (<, >, ==, !=, >=, <=) и круглых скобок.

Вариант 4. Входной язык содержит операторы цикла с параметром, разделенные символом точки с запятой (';'). Операторы цикла состоят из идентификаторов, десятичных чисел с плавающей запятой[3] в обычном и нормализованном экспоненциальном форматах, знаков присваивания ('='), знаков операций сравнения (<, >, ==, !=, >=, <=) и круглых скобок.

Вариант 5. Входной язык содержит арифметические выражения, разделенные символом точки с запятой (';'). Арифметические выражения состоят из идентификаторов, римских чисел[2] (корректные последовательности заглавных и строчных литер X, V и I). Кроме того, элементами арифметических выражений являются знаки присваивания ('='), знаки операций ('+', '-', '*', '/') и круглые скобки.

Вариант 6. Входной язык содержит логические выражения, разделенные символом точки с запятой (';'). Логические выражения состоят из идентификаторов, логических констант 0 и 1, знаков присваивания ('='), знаков операций *or*, *xor*, *and*, *not* и круглых скобок.

Вариант 7. Входной язык содержит операторы условий if-без-else и if-else, разделенные символом точки с запятой (';'). Операторы условий состоят из идентификаторов, римских чисел[4] (корректные последовательности заглавных и строчных литер X, V и I), знаков присваивания ('='), знаков операций сравнения (<, >, ==, !=, >=, <=) и круглых скобок.

Вариант 8. Входной язык содержит операторы цикла с параметром, разделенные символом точки с запятой (';'). Операторы цикла состоят из идентификаторов, римских чисел[4] (корректные последовательности заглавных и строчных литер X, V и I), знаков присваивания ('='), знаков операций сравнения (<, >, =, !=, >=, <=) и круглых скобок.

Вариант 9. Входной язык содержит арифметические выражения, разделенные символом точки с запятой (';'). Арифметические выражения состоят из

[1] http://ru.wikipedia.org/wiki/Плавающая_запятая
[2] http://ru.wikipedia.org/wiki/Римские_числа

идентификаторов, целочисленных констант в шестнадцатеричной системе. Кроме того, элементами арифметических выражений являются знаки присваивания ('='), знаки операций ('+', '-', '*', '/') и круглые скобки.

Вариант 10. Входной язык содержит выражения, разделенные символом точки с запятой (';'). Выражения состоят из идентификаторов, целочисленных констант в шестнадцатеричной системе, знаков присваивания ('='), знаков побитовых операций *or*, *xor*, *and*, *not* и круглых скобок.

Вариант 11. Входной язык содержит операторы условий if-без-else и if-else, разделенные символом точки с запятой (';'). Операторы условий состоят из идентификаторов, целочисленных констант в шестнадцатеричной системе, знаков присваивания ('='), знаков операций сравнения (<, >, ==, !=, >=, <=) и круглых скобок.

Вариант 12. Входной язык содержит операторы цикла с параметром, разделенные символом точки с запятой (';'). Операторы цикла состоят из идентификаторов, целочисленных констант в шестнадцатеричной системе, знаков присваивания ('='), знаков операций сравнения (<, >, =, !=, >=, <=) и круглых скобок.

Вариант 13. Входной язык содержит арифметические выражения, разделенные символом точки с запятой (';'). Арифметические выражения состоят из идентификаторов, комплексных чисел[1] (действительная и мнимая части отделяются нечувствительными к регистру символами I или J, и представляют собой два целых и/или десятичных числа с плавающей точкой в обычном формате), знаков присваивания ('='), знаков операций ('+', '-', '*', '/') и круглых скобок.

Вариант 14. Входной язык содержит логические выражения, разделенные символом точки с запятой (';'). Логические выражения состоят из идентификаторов, нечувствительных к регистру логических констант *T* и *NIL*, знаков присваивания ('='), знаков логических операций *or*, *xor*, *and*, *not* и круглых скобок.

Вариант 15. Входной язык содержит операторы условий if-без-else и if-else, разделенные символом точки с запятой (';'). Операторы условий состоят из идентификаторов, строковых констант (заключенная в двойные кавычки последовательность любых символов, за исключением двойных кавычек), знаков присваивания ('='), знаков операций сравнения (<, >, ==, !=, >=, <=) и круглых скобок.

[1] http://ru.wikipedia.org/wiki/Комплексное_число

Вариант 16. Входной язык содержит операторы цикла с параметром, разделенные символом точки с запятой (';'). Операторы цикла состоят из идентификаторов, целочисленных констант в восьмеричной системе, знаков присваивания ('='), знаков операций сравнения (<, >, =, !=, >=, <=) и круглых скобок.

Вариант 17. Входной язык содержит выражения, разделенные символом точки с запятой (';'). Выражения состоят из идентификаторов, битовых строк (начинающихся с обязательной пары знаков 0b или 0B последовательностей из 0 и 1), знаков присваивания ('='), знаков операций *or*, *xor*, *and*, *not* и круглых скобок.

Вариант 18. Входной язык содержит операторы условий if-без-else и if-else, разделенные символом точки с запятой (';'). Операторы условий состоят из идентификаторов, дат двух любых (на усмотрение разработчика) форматов, знаков присваивания ('='), знаков операций сравнения (<, >, ==, !=, >=, <=) и круглых скобок.

Вариант 19. Входной язык содержит операторы условий if-без-else и if-else, разделенные символом точки с запятой (';'). Операторы условий состоят из идентификаторов, временных величин любого (на усмотрение разработчика) формата, знаков присваивания ('='), знаков операций сравнения (<, >, ==, !=, >=, <=) и круглых скобок.

Вариант 20. Входной язык содержит операторы цикла типа с параметром, разделенные символом точки с запятой (';'). Операторы цикла состоят из идентификаторов, символьных констант в стиле языка программирования C[1], заключенных в апострофы, знаков присваивания ('='), знаков операций сравнения (<, >, =, !=, >=, <=) и круглых скобок.

Вариант 21. Входной язык содержит арифметические выражения, разделенные символом точки с запятой (';'). Арифметические выражения состоят из идентификаторов, десятичных чисел с плавающей запятой в обычном формате[2]. Кроме того, элементами арифметических выражений являются знаки присваивания (':='), знаки операций ('+', '-', '*', '/', '%', модуля) и круглые скобки.

Вариант 22. Входной язык содержит логические выражения, разделенные символом точки с запятой (';'). Логические выражения состоят из идентификаторов, нечувствительных к регистру логических констант *t* (истина) и *f* (ложь), знаков присваивания (':='), знаков операций *or*, *xor*, *and*, *not*, стрелка Пирса (пара знаков -> или ключевое слово *peirce*) и круглых скобок.

[1] http://citforum.ru/programming/c/h11.shtml#112
[2] http://ru.wikipedia.org/wiki/Плавающая_запятая

Вариант 23. Входной язык содержит операторы условий if-без-else и if-else, разделенные символом точки с запятой (';'). Операторы условий состоят из идентификаторов, десятичных чисел с плавающей запятой[1] в нормализованном экспоненциальном формате, знаков присваивания (':='), знаков операций сравнения (<, >, =, <>, >=, <=, =>, =<) и круглых скобок.

Вариант 24. Входной язык содержит операторы цикла с параметром, разделенные символом точки с запятой (';'). Операторы цикла состоят из идентификаторов, десятичных чисел с плавающей запятой[8] в обычном и нормализованном экспоненциальном форматах, знаков присваивания (':='), знаков операций сравнения (<, >, =, <>, >=, <=, =>, =<) и круглых скобок.

Вариант 25. Входной язык содержит арифметические выражения, разделенные символом точки с запятой (';'). Арифметические выражения состоят из идентификаторов, чисел из римских цифр[2] (корректные последовательности заглавных и строчных литер X, V и I). Кроме того, элементами арифметических выражений являются знаки присваивания (':='), знаки операций ('+', '-', '*', '/', '%') и круглые скобки.

Вариант 26. Входной язык содержит логические выражения, разделенные символом точки с запятой (';'). Логические выражения состоят из идентификаторов, логических констант 0 и 1, знаков присваивания (':='), знаков операций *or*, *xor*, *and*, *not*, штрих Шеффера (знак вертикальной черты или ключевое слово *sheffer*) и круглых скобок.

Вариант 27. Входной язык содержит операторы условий if-без-else и if-else, разделенные символом точки с запятой (';'). Операторы условий состоят из идентификаторов, римских чисел[9] (корректные последовательности заглавных и строчных литер X, V и I), знаков присваивания (':='), знаков операций сравнения (<, >, =, <>, !=, >=, <=, =>, =<) и круглых скобок.

Вариант 28. Входной язык содержит операторы цикла с параметром, разделенные символом точки с запятой (';'). Операторы цикла состоят из идентификаторов, римских чисел[9] (корректные последовательности заглавных и строчных литер X, V и I), знаков присваивания (':='), знаков операций сравнения (<, >, =, <>, >=, <=, =>, =<) и круглых скобок.

Вариант 29. Входной язык содержит арифметические выражения, разделенные символом точки с запятой (';'). Арифметические выражения состоят из идентификаторов, целочисленных констант в шестнадцатеричной системе. Кроме того, элементами арифметических выражений являются знаки присваивания (':='), знаки операций ('+', '-', '*', '/', '%', '^') и круглые скобки.

---

[1] http://ru.wikipedia.org/wiki/Плавающая_запятая
[2] http://ru.wikipedia.org/wiki/Римские_числа

Вариант 30. Входной язык содержит выражения, разделенные символом точки с запятой (';'). Выражения состоят из идентификаторов, целочисленных констант в шестнадцатеричной системе, знаков присваивания (':='), знаков побитовых операций *or*, *xor*, *and*, *not*, <<, >> (сдви[1]г влево и вправо) и круглых скобок.

Вариант 31. Входной язык содержит операторы условий if-без-else и if-else, разделенные символом точки с запятой (';'). Операторы условий состоят из идентификаторов, целочисленных констант в шестнадцатеричной системе, знаков присваивания (':='), знаков операций сравнения (<, >, =, ==, <>, !=, >=, <=) и круглых скобок.

Вариант 32. Входной язык содержит операторы цикла с параметром, разделенные символом точки с запятой (';'). Операторы цикла состоят из идентификаторов, целочисленных констант в шестнадцатеричной системе, знаков присваивания (':='), знаков операций сравнения (<, >, =, ==, <>, !=, >=, <=) и круглых скобок.

[1] http://ru.wikipedia.org/wiki/Битовый_сдвиг