北京邮电大学计算机学院（国家示范性软件学院）本科2019级

编译原理与技术

词法分析实验报告



姓名： 江姝潼

班级：2019211314

学号：2019211653

专业： 网络工程

目录

[一、实验概述 3](#_Toc84929448)

[1、实验题目与要求 3](#_Toc84929449)

[2、开发环境 3](#_Toc84929450)

[二、结构设计 3](#_Toc84929451)

[1、语言说明 3](#_Toc84929452)

[2、正规文法 4](#_Toc84929453)

[3、状态转换图 5](#_Toc84929454)

[三、实现方案 5](#_Toc84929455)

[1、处理方式 5](#_Toc84929456)

[2、输入输出 7](#_Toc84929457)

[3、数据结构 7](#_Toc84929458)

[4、函数说明 8](#_Toc84929459)

[四、实验运行结果及分析说明 11](#_Toc84929460)

[测试用例一： 11](#_Toc84929461)

[测试用例二： 12](#_Toc84929462)

[测试用例三： 14](#_Toc84929463)

[测试用例四： 15](#_Toc84929464)

[测试用例五： 18](#_Toc84929465)

[五、心得体会 22](#_Toc84929466)

[六、LEX词法运行 24](#_Toc84929467)

[1、基本原理 24](#_Toc84929468)

[2、结构分析 24](#_Toc84929469)

[3、输入输出 28](#_Toc84929470)

[4、心得体会 29](#_Toc84929471)

[附录一：运行源代码 29](#_Toc84929472)

[附录二：lex.i 代码文件 43](#_Toc84929473)

[附录二：lex.yy.c 代码文件 47](#_Toc84929474)

# 一、实验概述

## 1、实验题目与要求

题目：词法分析程序的设计与实现。

实验内容：设计并实现（语言的词法分析程序，要求实现如下功能。

（1）可以识别出用C语言编写的源程序中的每个单词符号,并以记号的形式输出每个单词符号。

（2）可以识别并跳过源程序中的注释。

（3）可以统计源程序中的语句行数、各类单词 的个数、以及字符总数，并输出统计结果。

（4）检查源程序中存在的词法错误,并报告错误所在的位置。

(5）对源程序中出现的错误进行适当的恢复，使词法分析可以继续进行，对源程序进行一次扫描,即可检查并报告源程序中存在的所有词法错误。

实现要求：分别用以下两种方法实现。

方法1：采用 C/C++作为实现语言,手工编写词法分析程序。

方法2：编写 LEX 源程序，利用 LEX编译程序自动生成词法分析程序。

## 2、开发环境

Windows 10 设备机，Visual Stdio Code 2019编程环境

# 二、结构设计

## 1、语言说明

该代码识别的语言有以下记号和单词：

1. 标识符：以字母开头的，后跟字母或数字或下划线的符号串
2. 关键字：该语言保留的字符合集，定义如下：

"if", "else", "while",

            "return", "for", "break",

                     "continue", "do", "int",

                     "char", "double", "void",

                     "short", "bool", "include",

                     "const", "long", "NULL",

                     "cin", "cout", "printf", "sizeof"

（3）常数（无符号数）：由整数部分、可选的小数部分组成

（4）符号：具体包括以下几类：

|  |  |
| --- | --- |
| 符号类型 | 包括符号 |
| 关系运算符 | >,>=,<,<=,=,==,!=,!,|,||,&,&& |
| 算数运算符 | +,++,+=,-,--,-=,\*,\*= |
| 标点符号 | ,,:,;,’,” |
| 其他符号 | #,空格 |

（5）注释，包括//,/\*和\*/

## 2、正规文法

下面给出标识符、无符号数和关系运算符的正则表达式和文法。

（1）标识符

正则表达式：

文法：

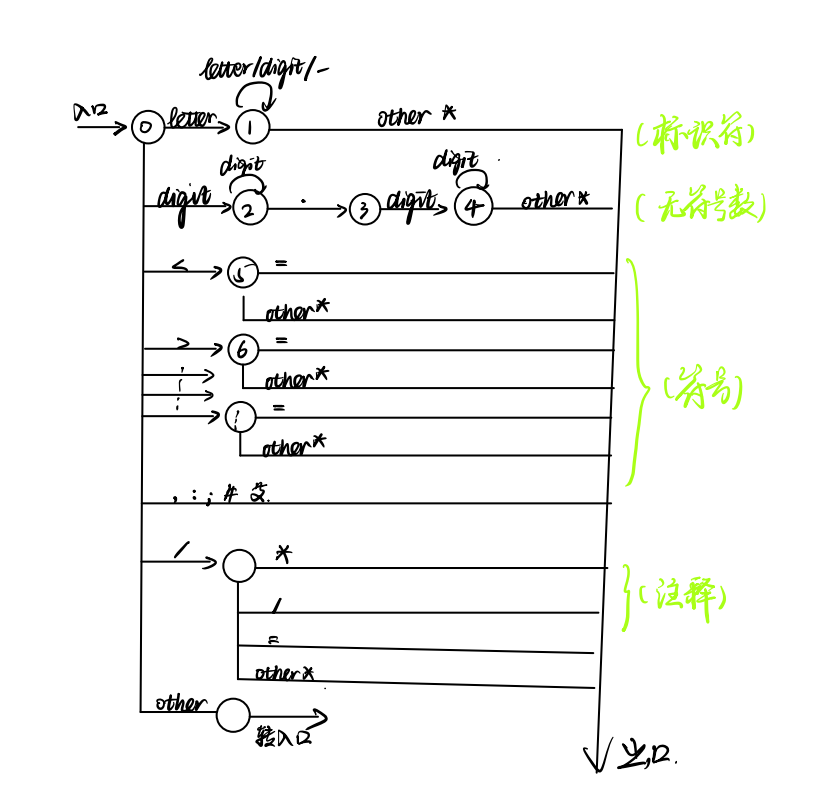
（2）常数

正则表达式：

文法：

（3）关系运算符的文法

## 3、状态转换图



其中，状态0是初始状态，若此时读入的字符是字母，则转换到状态1，进入标识符识别过程；如果读入的字符是数字，则转换到状态2，进人无符号数识别过程；若读入的字符是“/”，再读入下一个字符，如果读入的是“\*”或“/”，则进入注释处理状态。

# 三、实现方案

## 1、处理方式

有了上述状态转换图，只要把语义动作进一步添加到状态转换图中，使每一个状态都对应一小段程序，就可以构造出相应的词法分析程序，即根据所读进的非空字符转相应的程序段进行处理，下面为具体处理方式：

|  |  |  |
| --- | --- | --- |
| **符号类型** | | **处理方式** |
| 标识符/关键字 | | 在标识符状态，识别并组合出一个标识符之后，如查关键字表，以确定识别出的单词符号是关键字还是用户自定义标识符，并输出相应的记号。 |
| 无符号常数 | | 可识别出各种常数，包括整数和小数，并根据文法判断其是否为合法的常数 |
| 运算符 | 关系运算符 | 对<,>,!,|,&等符号，需要特别判断下一个字符，如在“<”状态，若读进的下一个字符是“=”，则输出关系运算符“<=”；否则输出关系运算符“<”。 |
| 算术运算符 | 对+,-,\*,/等符号，需要特别判断下一个字符，如在“+”状态，若读进的下一个字符是“+”，则输出算术运算符“++”,若读进的下一个字符是“=”，则输出算术运算符“+=”；否则输出算术运算符“+”。 |
| 界符和标点符号 | 只需输出其相应的记号即可 |
| 注释 | | 在“/”状态，若读进的下一个字符是“\*”或“/”，则进入注释处理状态，词法分析程序要做的工作是跳过注释，具体做法就是不断地读字符，直到遇到“\*/”为止，然后转开始状态，继续识别和分析下一个单词；若读进的下一个字符不是“\*”，则输出斜杠“/”。 |

若进入错误处理状态，表示词法分析程序从源程序中读入了一个不合法的字符。所谓不合法的字符是指该语言不包括以此字符开头的单词符号。词法分析程序发现不合法字符时，要做错误处理，其主要工作是显示或打印错误信息，并跳过这个字符，然后转开始状态继续识别和分析下一个单词符号。

## 2、输入输出

输入为一个待处理的完整的c/c++语言程序。

输出：在分离出一个单词后，对识别出的记号以二元式的形式加以输出，其形式为<记号，属性>，具体输出形式如下：

|  |  |  |
| --- | --- | --- |
| **符号类型** | | **输出形式** |
| 标识符 | | <变量名，Variable> |
| 关键字 | | <关键字，Keyword> |
| 无符号常数 | | <常数，Number> |
| 运算符 | 关系运算符和算术运算符 | <符号， operator> |
| 界符 | <界符，boundary> |
| 注释 | | //: single-lined annotation  /\*: long annotation |
| 报错 | | 常数错误：error! There is an illegal constant number!  变量错误：error! There is an illegal variable name!  单引号使用错误：There is no matching single colon.  双引号使用错误: There is no matching double colon. |

## 3、数据结构

（1）统计字符变量：

int NumOfLine = 0;     //行数

int NumOfKeyWord = 0;  //关键字数目

int NumOfVarible = 0;  //变量名数目

int NumOfNumber = 0;   //数字常量数目

int NumOfOperator = 0; //运算符数目

int NumOfBound = 0;    //界符数目

（2）保留字和运算符：采用了没有顺序且不能有重复元素的数据存储方式：set，查找时使用set的内置函数find()即可

set<string> keywords{  //保留字

                     "if", "else", "while",

                     "return", "for", "break",

                     "continue", "do", "int",

                     "char", "double", "void",

                     "short", "bool", "include",

                     "const", "long", "NULL",

                     "cin", "cout", "printf", "sizeof"};

set<char> symbols{

    '>','<','=','!','|','&',   //比较运算符

    '+','-','\*','/','%',       //算术运算符

    '[',']','(',')','{','}',   //界符

    ',',':',';','"','\'',' ','\n'   //其他符号

};

1. 符号表：采用了没有顺序且不能有重复键值的数据存储方式：unordered\_map，存储的是<变量名，所在行>形式的键值对，当遇到一个标识符需要查找是否在符号表里时，使用内置函数find()查找即可，若不在符号表中，则用emplace()函数插入map中。

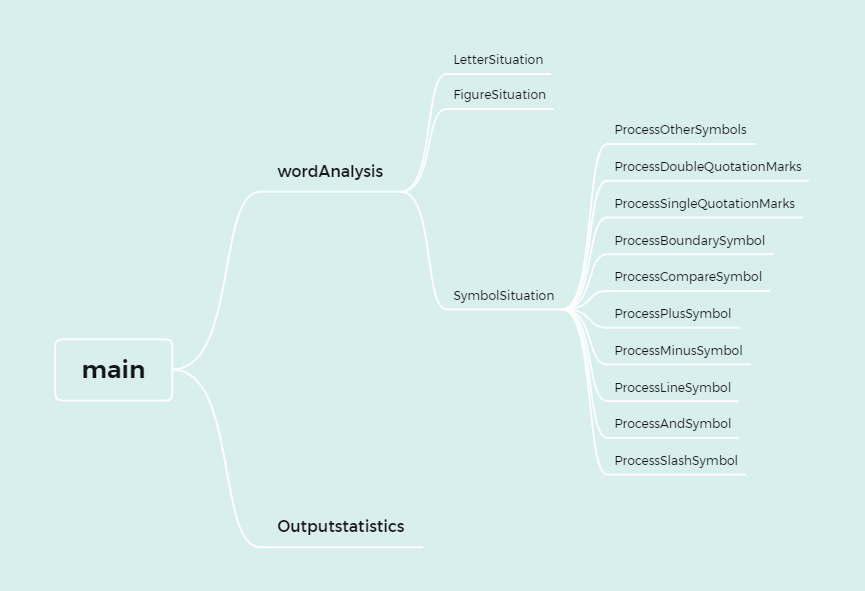
unordered\_map<string, int> SymbolTable;

## 4、函数说明

1、各函数功能表

|  |  |  |
| --- | --- | --- |
| **函数名** | **参数** | **功能** |
| bool isLetter(char ch); | 字符ch | 判断字符是否为字母 |
| bool isFigure(char ch); | 字符ch | 判断字符是否为字母 |
| void OutputStatistics(); | 无 | 输出符号表和统计的各字符数量 |
| void wordAnalysis(string s); | 字符串s | 对一行的内容进行词法分析 |
| void printPair(string sym, string attrtibute); | 符号sym，属性attribute | 输出<符号，属性二元组> |
| void LetterSituation(string s, int &pos, char &now); | 字符串s,当前所在的位置pos,当前字符now | 当读到的字符为字母时的处理 |
| void FigureSituation(string s, int &pos, char &now); | 字符串s,当前所在的位置pos,当前字符now | 当读到的字符为数字时的处理 |
| void SymbolSituation(string s, int &pos, char &now, int &state, int &flag); | 字符串s,当前所在的位置pos,当前字符now，当前状态state，标志变量flag | 读到的字符为其他字符时的处理 |
| void ProcessOtherSymbols(string s, int &pos, char &now, int &state); | 字符串s,当前所在的位置pos,当前字符now，当前状态state | 当字符为其他字符时的处理 |
| void ProcessSingleQuotationMarks(string s, int &pos, char &now, int &state); | 字符串s,当前所在的位置pos,当前字符now，当前状态state | 当字符为单引号时的处理 |
| void ProcessDoubleQuotationMarks(string s, int &pos, char &now, int &state); | 字符串s,当前所在的位置pos,当前字符now，当前状态state | 当字符为双引号时的处理 |
| void ProcessBoundarySymbol(string s, int &pos, char &now, int &state); | 字符串s,当前所在的位置pos,当前字符now，当前状态state | 当字符为界符时的处理 |
| void ProcessCompareSymbol(string s, int &pos, char &now, int &state); | 字符串s,当前所在的位置pos,当前字符now，当前状态state | 当字符为比较运算符时的处理 |
| void ProcessPlusSymbol(string s, int &pos, char &now, int &state); | 字符串s,当前所在的位置pos,当前字符now，当前状态state | 当字符为加号时的处理 |
| void ProcessMinusSymbol(string s, int &pos, char &now, int &state); | 字符串s,当前所在的位置pos,当前字符now，当前状态state | 当字符为减号时的处理 |
| void ProcessLineSymbol(string s, int &pos, char &now, int &state); | 字符串s,当前所在的位置pos,当前字符now，当前状态state | 当字符为“|”时的处理 |
| void ProcessAndSymbol(string s, int &pos, char &now, int &state); | 字符串s,当前所在的位置pos,当前字符now，当前状态state | 当字符为“&”时的处理 |
| void ProcessSlashSymbol(string s, int &pos, char &now, int &state, int &flag); | 字符串s,当前所在的位置pos,当前字符now，当前状态state，标志变量flag | 当字符为“/”时的处理 |

2、函数调用关系图



# 四、实验运行结果及分析说明

根据程序执行过程，给出以下5个测试用例，测试了各方面的词法分析功能。

## 测试用例一：

用于测试各种注释，略过头文件，统计运算符

输入：

1. /\*this
2. Is
3. A
4. Long
5. Annotation\*/
6. //this is a short annotation
7. #include<stdio.h>
8. #include<stdlib.h>
10. **int** main(){
11. printf("hello world!");
12. **return** 0;
13. }

输出结果：

*Line 5: long annotation*

*Line 6: single-lined annotation*

*Line 10: <int, Keyword>*

*Line 10: <main, Variable>*

*Line 10: <(, boundary>*

*Line 10: <), boundary>*

*Line 10: <{, boundary>*

*Line 11: <printf, Keyword>*

*Line 11: <(, boundary>*

*Line 11: <hello world!, string>*

*Line 11: <), boundary>*

*Line 12: <return, Keyword>*

*Line 12: <0, Number>*

*Line 13: <}, boundary>*

*符号表:*

*At Line 10: main*

*行数：13*

*关键词出现的次数：3*

*变量出现的次数：1*

*数字常量出现的次数：1*

*运算符出现的次数：0*

*界符出现的次数：6*

输出结果分析：

该样例测试了各种注释的情况，当遇到由“/\*”符号代表的长注释则会一直往下遍历直到遍历到“\*/”代表的注释结束符为止，如果遇到双斜杠代表的短柱式则直接略过符号后的行内容，并对注释的地方输出。

除此之外，还对各种界符进行了处理，包括小括号中括号大括号等，以<符号，boundary>的形式输出，若遇保留字中的关键词则会以关键词的形式输出，遇到双引号中的内容则以字符串string的形式输出。

## 测试用例二：

用于测试常数、小数是否正确

输入：

1. #include<stdio.h>
2. **int** main(){
3. **int** a = 3.45t;
4. **int** b = 23d;
5. **double** ab3@;
6. **return** 0;
7. }

输出结果：

*Line 2: <int, Keyword>*

*Line 2: <main, Variable>*

*Line 2: <(, boundary>*

*Line 2: <), boundary>*

*Line 2: <{, boundary>*

*Line 3: <int, Keyword>*

*Line 3: <a, Variable>*

*Line 3: <=, operator>*

*Line 3: error! There is an illegal constant number!*

*Line 4: <int, Keyword>*

*Line 4: <b, Variable>*

*Line 4: <=, operator>*

*Line 4: error! There is an illegal constant number!*

*Line 5: <double, Keyword>*

*Line 5: error! There is an illegal variable name!*

*Line 6: <return, Keyword>*

*Line 6: <0, Number>*

*Line 7: <}, boundary>*

*符号表:*

*At Line 4: b*

*At Line 2: main*

*At Line 3: a*

*行数：7*

*关键词出现的次数：5*

*变量出现的次数：3*

*数字常量出现的次数：1*

*运算符出现的次数：2*

*界符出现的次数：4*

输出结果分析：

该样例对变量和常数出现的几种错误情况展开了判断并作适当的修复。

首先就无符号数来说，如果数字后面出现了，非常规字符则会报错，声明该常数这是不合法的；在变量声明或者是使用变量时检测到,在变量中字母开头后面还有除了数字、字母或下划线的其他的字符，则会报错声明这是一个不合法的变量名，并不把其加入到符号表中。

值得一提的是这个里面还进行了错误处理，那几个常数虽然不合法的，但是变量a和b还会依然加入到符号表中，但是当后面声明的变量是不合法的变量时，则不会放入到符号表中并报错。

## 测试用例三：

用于测试单、双引号使用是否正确

输入：

1. #include<stdio.h>
2. **int** main(){
3. **int** a = 3;
4. **char** ch = 'k;
5. printf("**this** is an output\n);
6. **return** 0;
7. }

输出：

*Line 2: <int, Keyword>*

*Line 2: <main, Variable>*

*Line 2: <(, boundary>*

*Line 2: <), boundary>*

*Line 2: <{, boundary>*

*Line 3: <int, Keyword>*

*Line 3: <a, Variable>*

*Line 3: <=, operator>*

*Line 3: <3, Number>*

*Line 4: <char, Keyword>*

*Line 4: <ch, Variable>*

*Line 4: <=, operator>*

*Line 4: error! There is no matching single colon.*

*Line 5: <printf, Keyword>*

*Line 5: <(, boundary>*

*Line 5: error! There is no matching double colon.*

*Line 6: <return, Keyword>*

*Line 6: <0, Number>*

*Line 7: <}, boundary>*

*符号表:*

*At Line 4: ch*

*At Line 2: main*

*At Line 3: a*

*行数：7*

*关键词出现的次数：5*

*变量出现的次数：3*

*数字常量出现的次数：2*

*运算符出现的次数：2*

*界符出现的次数：5*

输出结果分析：

在第3个样例中，就单双引号使用不正确的情况进行了错误判断，在使用单引号时，如果单引号后面两个字符的位置没有发现另外一个配对的单引号，在会报错；如遇到双引号则一直往后遍历，直到遍历到下一个双引号的位置。如果找不到下一个双引号，则会报错没有配对的双引号，且第一个双引号后的内容会被忽略并不加入到符号表中。

## 测试用例四：

判断是否为素数的完整代码测试：

输入：

1. #include <stdio.h>
2. #include <math.h>
4. **int** main()
5. {
6. **int** m, flag, i;
7. scanf("%d", &m);
8. flag = 1;
9. **for**(i = 2;i < m;i++)
10. {
11. **if**(m%i == 0)
12. {
13. flag = 0;
14. **break**;
15. }
16. }
17. **if**(flag)
18. {
19. printf("%d is primers.\n", m);
20. }
21. **else**
22. {
23. printf("%d is not primers.\n", m);
24. }
25. **return** 0;
26. }

输出结果：

*Line 4: <int, Keyword>*

*Line 4: <main, Variable>*

*Line 4: <(, boundary>*

*Line 4: <), boundary>*

*Line 5: <{, boundary>*

*Line 6: <int, Keyword>*

*Line 6: <m, Variable>*

*Line 6: <flag, Variable>*

*Line 6: <i, Variable>*

*Line 7: <scanf, Keyword>*

*Line 7: <(, boundary>*

*Line 7: <%d, string>*

*Line 7: <&, operator>*

*Line 7: <m, Variable>*

*Line 7: <), boundary>*

*Line 8: <flag, Variable>*

*Line 8: <=, operator>*

*Line 8: <1, Number>*

*Line 9: <for, Keyword>*

*Line 9: <(, boundary>*

*Line 9: <i, Variable>*

*Line 9: <=, operator>*

*Line 9: <2, Number>*

*Line 9: <i, Variable>*

*Line 9: <<, operator>*

*Line 9: <m, Variable>*

*Line 9: <i, Variable>*

*Line 9: <++, operator>*

*Line 9: <), boundary>*

*Line 10: <{, boundary>*

*Line 11: <if, Keyword>*

*Line 11: <(, boundary>*

*Line 11: <m, Variable>*

*Line 11: <%, operator>*

*Line 11: <i, Variable>*

*Line 11: <==, operator>*

*Line 11: <0, Number>*

*Line 11: <), boundary>*

*Line 12: <{, boundary>*

*Line 13: <flag, Variable>*

*Line 13: <=, operator>*

*Line 13: <0, Number>*

*Line 14: <break, Keyword>*

*Line 15: <}, boundary>*

*Line 16: <}, boundary>*

*Line 17: <if, Keyword>*

*Line 17: <(, boundary>*

*Line 17: <flag, Variable>*

*Line 17: <), boundary>*

*Line 18: <{, boundary>*

*Line 19: <printf, Keyword>*

*Line 19: <(, boundary>*

*Line 19: <%d is primers.\n, string>*

*Line 19: <m, Variable>*

*Line 19: <), boundary>*

*Line 20: <}, boundary>*

*Line 21: <else, Keyword>*

*Line 22: <{, boundary>*

*Line 23: <printf, Keyword>*

*Line 23: <(, boundary>*

*Line 23: <%d is not primers.\n, string>*

*Line 23: <m, Variable>*

*Line 23: <), boundary>*

*Line 24: <}, boundary>*

*Line 25: <return, Keyword>*

*Line 25: <0, Number>*

*Line 26: <}, boundary>*

*符号表:*

*At Line 6: flag*

*At Line 4: main*

*At Line 6: i*

*At Line 6: m*

*行数：26*

*关键词出现的次数：11*

*变量出现的次数：16*

*数字常量出现的次数：5*

*运算符出现的次数：8*

*界符出现的次数：24*

输出结果分析：

后两个样例就能实现完整功能的C语言代码开展了测试，具体词法分析的过程如下：

在分析到“#”代表的头文件时则会略过这一行，然后依行地往下读，如果读到的是字母开头的则判断是否为变量或关键字，如果是关键字则输出，若为变量则判断是否加入了符号表中，如果没有则加入，反之则略过，并加统计数量之中。

如果读到的为数字开头，继续向后读，可以合法读入一个“.”号和若干个数字，如果读到除了符号外的其他字符，则报错。

如果遇到括号小括号大括号等分界符，则对应地在分界符中的计数增加，如果遇到加减等符号，则在对应算术运算符统计中数量加加。遇到如空格，逗号，分号等其他字符，则会略过。

## 测试用例五：

完整冒泡排序代码测试：

输入：

1. # include <stdio.h>
2. **int** main()
3. {
4. **int** a[] = {900, 2, 3, -58, 34, 76, 32, 43, 56, -70, 35, -234, 532, 543, 2500};
5. **int** n;  //存放数组a中元素的个数
6. **int** i;  //比较的轮数
7. **int** j;  //每轮比较的次数
8. **int** buf;  //交换数据时用于存放中间数据
9. n = **sizeof**(a) / **sizeof**(a[0]);  /\*a[0]是int型, 占4字节, 所以总的字节数除以4等于元素的个数\*/
10. **for** (i=0; i<n-1; ++i)  //比较n-1轮
11. {
12. **for** (j=0; j<n-1-i; ++j)  //每轮比较n-1-i次,
13. {
14. **if** (a[j] < a[j+1])
15. {
16. buf = a[j];
17. a[j] = a[j+1];
18. a[j+1] = buf;
19. }
20. }
21. }
22. **for** (i=0; i<n; ++i)
23. {
24. printf("%d\x20", a[i]);
25. }
26. printf("\n");
27. **return** 0;
28. }

输出结果：

*Line 2: <int, Keyword>*

*Line 2: <main, Variable>*

*Line 2: <(, boundary>*

*Line 2: <), boundary>*

*Line 3: <{, boundary>*

*Line 4: <int, Keyword>*

*Line 4: <a, Variable>*

*Line 4: <[, boundary>*

*Line 4: <], boundary>*

*Line 4: <=, operator>*

*Line 4: <{, boundary>*

*Line 4: <900, Number>*

*Line 4: <2, Number>*

*Line 4: <3, Number>*

*Line 4: <-, operator>*

*Line 4: <58, Number>*

*Line 4: <34, Number>*

*Line 4: <76, Number>*

*Line 4: <32, Number>*

*Line 4: <43, Number>*

*Line 4: <56, Number>*

*Line 4: <-, operator>*

*Line 4: <70, Number>*

*Line 4: <35, Number>*

*Line 4: <-, operator>*

*Line 4: <234, Number>*

*Line 4: <532, Number>*

*Line 4: <543, Number>*

*Line 4: <2500, Number>*

*Line 4: <}, boundary>*

*Line 5: <int, Keyword>*

*Line 5: <n, Variable>*

*Line 5: single-lined annotation*

*Line 6: <int, Keyword>*

*Line 6: <i, Variable>*

*Line 6: single-lined annotation*

*Line 7: <int, Keyword>*

*Line 7: <j, Variable>*

*Line 7: single-lined annotation*

*Line 8: <int, Keyword>*

*Line 8: <buf, Variable>*

*Line 8: single-lined annotation*

*Line 9: <n, Variable>*

*Line 9: <=, operator>*

*Line 9: <sizeof, Keyword>*

*Line 9: <(, boundary>*

*Line 9: <a, Variable>*

*Line 9: <), boundary>*

*Line 9: </, operator>*

*Line 9: <sizeof, Keyword>*

*Line 9: <(, boundary>*

*Line 9: <a, Variable>*

*Line 9: <[, boundary>*

*Line 9: <0, Number>*

*Line 9: <], boundary>*

*Line 9: <), boundary>*

*Line 9: long annotation*

*Line 10: <for, Keyword>*

*Line 10: <(, boundary>*

*Line 10: <i, Variable>*

*Line 10: <=, operator>*

*Line 10: <0, Number>*

*Line 10: <i, Variable>*

*Line 10: <<, operator>*

*Line 10: <n, Variable>*

*Line 10: <-, operator>*

*Line 10: <1, Number>*

*Line 10: <++, operator>*

*Line 10: <i, Variable>*

*Line 10: <), boundary>*

*Line 10: single-lined annotation*

*Line 11: <{, boundary>*

*Line 12: <for, Keyword>*

*Line 12: <(, boundary>*

*Line 12: <j, Variable>*

*Line 12: <=, operator>*

*Line 12: <0, Number>*

*Line 12: <j, Variable>*

*Line 12: <<, operator>*

*Line 12: <n, Variable>*

*Line 12: <-, operator>*

*Line 12: <1, Number>*

*Line 12: <-, operator>*

*Line 12: <i, Variable>*

*Line 12: <++, operator>*

*Line 12: <j, Variable>*

*Line 12: <), boundary>*

*Line 12: single-lined annotation*

*Line 13: <{, boundary>*

*Line 14: <if, Keyword>*

*Line 14: <(, boundary>*

*Line 14: <a, Variable>*

*Line 14: <[, boundary>*

*Line 14: <j, Variable>*

*Line 14: <], boundary>*

*Line 14: <<, operator>*

*Line 14: <a, Variable>*

*Line 14: <[, boundary>*

*Line 14: <j, Variable>*

*Line 14: <+, operator>*

*Line 14: <1, Number>*

*Line 14: <], boundary>*

*Line 14: <), boundary>*

*Line 15: <{, boundary>*

*Line 16: <buf, Variable>*

*Line 16: <=, operator>*

*Line 16: <a, Variable>*

*Line 16: <[, boundary>*

*Line 16: <j, Variable>*

*Line 16: <], boundary>*

*Line 17: <a, Variable>*

*Line 17: <[, boundary>*

*Line 17: <j, Variable>*

*Line 17: <], boundary>*

*Line 17: <=, operator>*

*Line 17: <a, Variable>*

*Line 17: <[, boundary>*

*Line 17: <j, Variable>*

*Line 17: <+, operator>*

*Line 17: <1, Number>*

*Line 17: <], boundary>*

*Line 18: <a, Variable>*

*Line 18: <[, boundary>*

*Line 18: <j, Variable>*

*Line 18: <+, operator>*

*Line 18: <1, Number>*

*Line 18: <], boundary>*

*Line 18: <=, operator>*

*Line 18: <buf, Variable>*

*Line 19: <}, boundary>*

*Line 20: <}, boundary>*

*Line 21: <}, boundary>*

*Line 22: <for, Keyword>*

*Line 22: <(, boundary>*

*Line 22: <i, Variable>*

*Line 22: <=, operator>*

*Line 22: <0, Number>*

*Line 22: <i, Variable>*

*Line 22: <<, operator>*

*Line 22: <n, Variable>*

*Line 22: <++, operator>*

*Line 22: <i, Variable>*

*Line 22: <), boundary>*

*Line 23: <{, boundary>*

*Line 24: <printf, Keyword>*

*Line 24: <(, boundary>*

*Line 24: <%d\x20, string>*

*Line 24: <a, Variable>*

*Line 24: <[, boundary>*

*Line 24: <i, Variable>*

*Line 24: <], boundary>*

*Line 24: <), boundary>*

*Line 25: <}, boundary>*

*Line 26: <printf, Keyword>*

*Line 26: <(, boundary>*

*Line 26: <\n, string>*

*Line 26: <), boundary>*

*Line 27: <return, Keyword>*

*Line 27: <0, Number>*

*Line 28: <}, boundary>*

*符号表:*

*At Line 8: buf*

*At Line 7: j*

*At Line 6: i*

*At Line 5: n*

*At Line 2: main*

*At Line 4: a*

*行数：28*

*关键词出现的次数：15*

*变量出现的次数：38*

*数字常量出现的次数：25*

*运算符出现的次数：25*

*界符出现的次数：48*

输出结果分析：

第五段代码是完整的冒泡排序代码，和前面不同的是，有代码的行中加入了注释，在读取的时候，词法分析程序会以行的方式读取，一行中在注释前的内容都会正常读取，读到了注释号则会略过注释号后的内容，并记录下该行有注释并在输出。

# 五、心得体会

**课程内容：**

经过这次词法分析程序的编写实验，加深了我对词法分析的认识，它既可以作为独立的一遍来为语法分析程序提供基础，也可以作为语法分析的协同程序。在程序的编译运行过程中，词法分析的存在避免了中间文件，省去了取送符号的工作，有利于提高编译程序的效率。本实验是将词法分析独立分离出来的，拥有诸多好处，如可以简化设计，容易识别并去除空格、注释，使语法分析程序致力于语法分析，结构清晰，易于实现；可以改进编译程序的效率，利用专门的读字符和处理记号的技术构造更有效的词法分析程序；还可以加强编译程序的可移植性，在词法分析程序中处理特殊的或非标准的符号即可等。在本次实验过程中，对这些好处有了更加实际的体会。

**代码规范性：**

本次实验就代码能力上加深了我对代码规范性的认知，之前的程序设计老师曾提过，优秀的代码应该是高内聚低耦合的。我一开始的代码是将所有的词法分析处理放在一个work函数中，该函数有超过400行的内容，显得非常的冗长和杂乱。于是代码完成后我对结果展开了优化，我根据各种情况将其分块，把具有一个完整功能的代码块单独提取出来成为一个函数，例如分析符号的情况转化成了分析“+”，“-”，“\”号等各个函数的内容。在上文中的函数关系图可以看出，各个函数之间的关系是非常明确，在增加了代码的可读性同时使代码的结构更加的明晰，让代码的结构更加清晰。

**目前不足和改进：**

在完成这个实验的同时，我也认识到我的代码就一个完整的c语言词法分析程序来说还是有非常多的不足的，有很多各种情况没有办法进行很好的处理。比如说，该代码无比无法区分出函数和变量，在识别main函数时，它会把main函数当成一个变量信息处理，在输入测试的所有代码程序中都没有出现子函数的情况，若出现了该种情况，该程序没法识别出函数是否正确声明，且没有把它当作函数的功能来进行处理，而是将其识别成了一个变量，就子函数的情况没法将其正常处理。

同时还对一些其他的特殊字符的多种情况没有考虑完全，若出现了“\*=\*”的符号，程序则不会报错，而是判断完前两个符号后，把第3个符号单独提取出来当作一个符号进行分析。在写代码的时候考虑过相关情况，但需要特判，且需要判断的情况过多，难以一一去处理。

# 六、LEX语言实现

## 1、基本原理

Lex的基本工作原理为：由正规式生成NFA，将NFA变换成DFA，DFA经化简后，模拟生成词法分析器。

其运行的主要步骤为：由翻译器将Lex源程序翻译成一个名为lex.yy.c的C语言源文件，用C语言编译程序对lex.yy.c进程单独编译，可生成目标文件lex.yy.o，或直接生成可执行程序a.exe。然后运行词法分析程序a.exe,将输入的字符串转换成相应的记号序列。

## 2、结构分析

Lex源程序一般分为定义段、词法规则段和辅助函数段三部分，三部分之间以符号%%分隔，接下来就各部分的结构对我编写的lex代码展开分析。

1、定义段

定义段可以分为两部分：第一部分以符号%{和%}包裹，里面写的一些定义和声明，包括头文件，宏定义，全局变量等。在代码中根据状态申明如下：

%{

#include <stdio.h>

#include <string.h>

/\* 比较运算符 \*/

#define LT                  1

#define LE                  2

#define GT                  3

#define GE                  4

#define EQ                  5

#define PL                  6

#define PP                  7

#define PE                  8

#define MS                  8

#define MM                  9

#define ME                  10

/\* 保留字 \*/

#define IF                  11

#define ELSE                12

#define RETURN              13

#define BREAK               14

#define WHILE               15

#define DO                  16

#define ID                  17

#define NUMBER              18

#define RELOP               19

#define DELIMITER           20

/\* 其他情况 \*/

#define ERRORCHAR           21

int yylval;

char symbolTable[1024][1024];  //表示符号表

int loc = 0;                   //表示符号表的大小

%}

第二部分是正规定义和状态定义。正规定义是为了简化后面的词法规则而给部分正规式定义了名字。每条正规定义也都要顶着行首写。下面正规定义分别定义了letter，digit和id所表示的正规式：

delim       [ \t \n]

ws          {delim}+

letter      [A-Za-z\_]

digit       [0-9]

id          {letter}({letter}|{digit})\*

number      {digit}+(\.{digit}+)?

2、词法规则段

在词法规则段列出词法分析器需要匹配的正规式，以及匹配该正规式后需要进行的相关动作，并采取最长匹配原则、优先匹配排在前面的正规式的规则进行匹配

{ws}              {;/\* 此时词法分析器没有动作，也不返回，而是继续分析。 \*/}

while             {return (WHILE);}

do                {return (DO);}

if                {return (IF);}

else              {return (ELSE);}

return            {return (RETURN);}

break             {return (BREAK);}

{id}              {yylval = installID (); return (ID);}

{number}          {return (NUMBER);}

"<"               {yylval = LT; return (RELOP);}

"<="              {yylval = LE; return (RELOP);}

"="               {yylval = EQ; return (RELOP);}

">"               {yylval = GT; return (RELOP);}

">="              {yylval = GE; return (RELOP);}

"+"               {yylval = PL; return (RELOP);}

"++"              {yylval = PP; return (RELOP);}

"+="              {yylval = PE; return (RELOP);}

"-"               {yylval = MS; return (RELOP);}

"--"              {yylval = MM; return (RELOP);}

"-="              {yylval = ME; return (RELOP);}

"{"               {return (DELIMITER);}

"}"               {return (DELIMITER);}

"("               {return (DELIMITER);}

")"               {return (DELIMITER);}

";"               {return (DELIMITER);}

.                 {yylval = ERRORCHAR; return ERRORCHAR;}

 /\*.匹配除换行之外的任何字符，一般可作为最后一条翻译规则。\*/

3、辅助函数段

辅助函数段用C语言语法来写，辅助函数一般是在词法规则段中用到的函数。这一部分一般会被直接拷贝到lex.yy.c中，这里写了插入符号表函数installID()，在该函数中判断当前读入ID是否在符号表中，若不在则加入其中。

int installID () {

    for(int i = 0; i < loc; i ++){

        if(strcmp(symbolTable[i], yytext) == 0) //如果符号表中已经存在

        return ID;

    }

    strcpy(symbolTable[loc], yytext);

    //printf("at loc %d, symbol is %s", loc, symbolTable[loc]);

    loc ++;

    return ID;

}

int yywrap (){

  return 1;

}

void writeout(int c){

  switch(c){

    case ERRORCHAR: fprintf(yyout, "(ERRORCHAR, \"%s\")\n", yytext);break;

    case RELOP: fprintf(yyout, "(RELOP, \"%s\") \n", yytext);break;

    case DELIMITER: fprintf(yyout, "(DELIMITER, \"%s\") \n", yytext);break;

    case WHILE: fprintf(yyout, "(WHILE, \"%s\")\n", yytext);break;

    case DO: fprintf(yyout, "(DO, \"%s\") \n", yytext);break;

    case IF: fprintf(yyout, "(IF, \"%s\") \n", yytext);break;

    case ELSE: fprintf(yyout, "(ELSE, \"%s\") \n", yytext);break;

    case RETURN: fprintf(yyout, "(RETURN, \"%s\") \n", yytext);break;

    case BREAK: fprintf(yyout, "(BREAK, \"%s\") \n", yytext);break;

    case NUMBER: fprintf(yyout, "(NUM, \"%s\")\n", yytext);break;

    case ID: fprintf(yyout, "(ID, \"%s\")\n", yytext);break;

    default:break;

  }

  return;

}

int main (int argc, char \*\* argv){

    int c,j=0;

    if (argc>=2){

      if ((yyin = fopen(argv[1], "r")) == NULL){

        printf("Can't open file %s\n", argv[1]);

        return 1;

      }

      if (argc>=3){

        yyout=fopen(argv[2], "w");

      }

    }

    while (c = yylex()){

        writeout(c);

        j++;

    }

    if(argc>=2){

      fclose(yyin);

      if (argc>=3) fclose(yyout);

    }

    printf("The symbolTable is listed as follows:\n");

    for(int i = 0; i < loc; i ++){

        printf("the %d symbol is: %s\n", i, symbolTable[i]);

    }

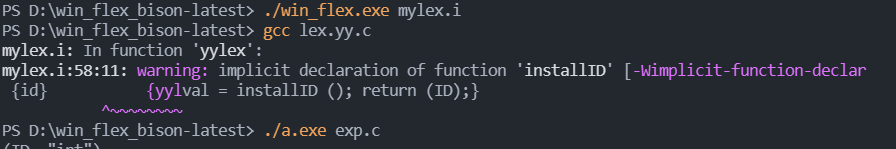
    return 0;

}

值得注意的是，这里还写了一个yywrap的辅助函数，该函数是当词法分析器遇到输入文件结尾时调用，若yywrap返回0，则继续扫描；返回1，则词法分析器返回报告文件已结束的0。故此情况需要返回1。

## 3、输入输出

编译运行mylex.i过程如下：



输入的input.c文件如下：

int main()

{

    int i = 0;

    do{

       i ++;

    }while(i < 10);

    return 0;

}

运行输出结果如图：

*(ID, "int")*

*(ID, "main")*

*(DELIMITER, "(")*

*(DELIMITER, ")")*

*(DELIMITER, "{")*

*(ID, "int")*

*(ID, "i")*

*(RELOP, "=")*

*(NUM, "0")*

*(DELIMITER, ";")*

*(DO, "do")*

*(DELIMITER, "{")*

*(ID, "i")*

*(RELOP, "++")*

*(DELIMITER, ";")*

*(DELIMITER, "}")*

*(WHILE, "while")*

*(DELIMITER, "(")*

*(ID, "i")*

*(RELOP, "<")*

*(NUM, "10")*

*(DELIMITER, ")")*

*(DELIMITER, ";")*

*(RETURN, "return")*

*(NUM, "0")*

*(DELIMITER, ";")*

*(DELIMITER, "}")*

*The symbolTable is listed as follows:*

*the 0 symbol is: int*

*the 1 symbol is: main*

*the 2 symbol is: i*

## 4、心得体会

此本次实验我接触了利用LEX自动生成词法分析程序。这对我来说是一门崭新的编程语言，在查阅资料和学习的过程中极大地锻炼了我的自学能力，同时该语言和c语言及自动机联系紧密，让我对自动机、词法分析的相关知识有了更深一步的理解和运用。

本次实验用两种语言的实现，也让我对比了LEX自动生成和自己手动实现词法分析的区别，通过lex.i生成的lex.yy.c文件虽然冗长复杂，不便于从人的角度阅读，但是各个状态较为独立，过程清晰，便于计算机的运行，同时容错性强。而我自己手动编写的c++语言程序，为了简化程序对一些处理细节作了合并和优化，较lex.yy.c文件简化了很多，独立分块的代码模式也便于他人阅读，但就处理起来可能效率略差。从这个探索的过程中我体会到了前人词法分析方面智慧，吸引我不断进取地学习。LEX源程序和生成的lex.yy.c文件在报告结尾附。

# 附录一：运行源代码

/\*

@project: wordAnalysis

@IDE:    Visual Studio

@Author: Jiang Shutong

@Date:   Oct 10th

\*/

#include <iostream>

#include <cstdio>

#include <cstdlib>

#include <string>

#include <cstring>

#include <set>

#include <unordered\_map>

using namespace std;

int NumOfLine = 0;     //行数

int NumOfKeyWord = 0;  //关键字数目

int NumOfVarible = 0;  //变量名数目

int NumOfNumber = 0;   //数字常量数目

int NumOfOperator = 0; //运算符数目

int NumOfBound = 0;    //界符数目

int flag = 0;

set<string> keywords{//保留字

                     "if", "else", "while",

                     "return", "for", "break",

                     "continue", "do", "int",

                     "char", "double", "void",

                     "short", "bool", "include",

                     "const", "long", "NULL", "scanf",

                     "cin", "cout", "printf", "sizeof"};

set<char> symbols{

    '>', '<', '=', '!', '|', '&',       //比较运算符

    '+', '-', '\*', '/', '%',            //算术运算符

    '[', ']', '(', ')', '{', '}',       //界符

    ',', ':', ';', '"', '\'', ' ', '\n' //其他符号

};

unordered\_map<string, int> SymbolTable;

bool isLetter(char ch);                        //判断是否为字母

bool isFigure(char ch);                        //判断是否为数字

void OutputStatistics();                       //输出各符号的统计结果

void wordAnalysis(string s);                   //词法分析

void printPair(string sym, string attrtibute); //以二元式的形式输出

void LetterSituation(string s, int &pos, char &now);          //读到的字符为字母时的处理

void FigureSituation(string s, int &pos, char &now);          //读到的字符为数字时的处理

void SymbolSituation(string s, int &pos, char &now, int &state, int &flag);  //读到的字符为其他字符时的处理

void ProcessOtherSymbols(string s, int &pos, char &now, int &state);         //当字符为其他字符时的处理

void ProcessSingleQuotationMarks(string s, int &pos, char &now, int &state); //当字符为单引号时的处理

void ProcessDoubleQuotationMarks(string s, int &pos, char &now, int &state); //当字符为双引号时的处理

void ProcessBoundarySymbol(string s, int &pos, char &now, int &state);       //当字符为界符时的处理

void ProcessCompareSymbol(string s, int &pos, char &now, int &state);        //当字符为比较运算符时的处理

void ProcessPlusSymbol(string s, int &pos, char &now, int &state);           //当字符为加号时的处理

void ProcessMinusSymbol(string s, int &pos, char &now, int &state);          //当字符为减号时的处理

void ProcessLineSymbol(string s, int &pos, char &now, int &state);           //当字符为|时的处理

void ProcessAndSymbol(string s, int &pos, char &now, int &state);            //当字符为&时的处理

void ProcessSlashSymbol(string s, int &pos, char &now, int &state, int &flag);//当字符为/时的处理

int main()

{

    char buffer[1024];

    string buf;

    FILE \*fptr;

    //如果打开文件失败

    if ((fptr = fopen("Input.txt", "r")) == NULL)

    {

        printf("Error! opening file");

    }

    //对每一行分别处理

    while (fgets(buffer, sizeof(buffer), fptr) != NULL)

    {

        NumOfLine++;

        buf = buffer;

        wordAnalysis(buffer);

    }

    OutputStatistics();

    return 0;

}

bool isLetter(char ch)

{

    if (ch >= 'a' && ch <= 'z')

        return true;

    else

        return false;

}

bool isFigure(char ch)

{

    if (ch >= '0' && ch <= '9')

        return true;

    else

        return false;

}

void wordAnalysis(string s)

{

    int state = 0;

    int pos = 0;

    int posVar;

    char now = ' ';

    char var[200]; //用于暂存标识符或关键字

    string variable;

    if (flag)

        state = 3;

    while (now != '\0')

    {

        //printf("now is %d or '%c', state is %d\n", now, now, state);

        switch (state)

        {

        case 0:

            now = s[pos];

            if (isLetter(now))

                state = 1;

            else if (isFigure(now))

                state = 2;

            else

                state = 3;

            break;

        case 1: //如果是字母开头，有标识符和关键字两种情况

            LetterSituation(s, pos, now);

            state = 0;

            break;

        case 2: //如果开头是数字

            FigureSituation(s, pos, now);

            state = 0;

            break;

        case 3:

            SymbolSituation(s, pos, now, state, flag);

            break;

        default:

            break;

        }

    }

}

void LetterSituation(string s, int &pos, char &now)

{

    int posVar;

    char var[200]; //用于暂存标识符或关键字

    string variable;

    memset(var, 0, sizeof(var));

    posVar = 0;

    while (isLetter(now) || isFigure(now) || now == '\_')

    {

        var[posVar++] = now;

        now = s[++pos];

    }

    if (symbols.find(s[pos]) == symbols.end()) //错误处理1：如果变量申明不合法即报错

    {

        cout << "Line " << NumOfLine << ": error! There is an illegal variable name!" << endl;

        now = '\0'; //有错误的这一行直接略过

    }

    else //如果变量申明合法

    {

        var[posVar] = '\0';

        variable = var;

        //如果是关键词，输出

        if (keywords.find(variable) != keywords.end())

        {

            printPair(var, "Keyword");

            NumOfKeyWord++;

        }

        //如果是变量，先输出，判断是否在符号表里，在则不做处理

        //如果不在，加进符号表里

        else

        {

            printPair(var, "Variable");

            NumOfVarible++;

            if (!SymbolTable.count(variable))

            {

                SymbolTable.emplace(variable, NumOfLine);

            }

        }

    }

}

void FigureSituation(string s, int &pos, char &now)

{

    int posVar;

    char var[200]; //用于暂存标识符或关键字

    string variable;

    memset(var, 0, sizeof(var));

    posVar = 0;

    while (isFigure(now))

    {

        var[posVar++] = now;

        now = s[++pos];

    }

    if (now == '.') //如果该数字含有小数点

    {

        var[posVar++] = now;

        now = s[++pos];

        while (isFigure(now))

        {

            var[posVar++] = now;

            now = s[++pos];

        }

    }

    if (symbols.find(s[pos]) == symbols.end()) //错误处理2：如果常数申明不合法即报错

    {

        cout << "Line " << NumOfLine << ": error! There is an illegal constant number!" << endl;

        now = '\0'; //有错误的这一行直接略过

    }

    else //如果变量申明合法

    {

        var[posVar] = '\0';

        variable = var;

        printPair(var, "Number");

        NumOfNumber++;

    }

}

void ProcessOtherSymbols(string s, int &pos, char &now, int &state)

{

    pos++;

    state = 0;

    now = s[pos];

}

void ProcessSingleQuotationMarks(string s, int &pos, char &now, int &state)

{

    pos += 2; //位置移到单引号后面两个位置

    now = s[pos];

    if (now != '\'') //错误处理4：若一行中没有找到对应的单引号，则报错

    {

        cout << "Line " << NumOfLine << ": error! There is no matching single colon." << endl;

        state = 0;

    }

}

void ProcessDoubleQuotationMarks(string s, int &pos, char &now, int &state)

{

    int posVar;

    char var[200]; //用于暂存标识符或关键字

    string variable;

    memset(var, 0, sizeof(var));

    posVar = 0;

    now = s[++pos];

    while (now != '"' && now != '\0')

    {

        var[posVar++] = now;

        now = s[++pos];

    }

    variable = var;

    if (now == '"')

    {

        printPair(variable, "string");

        pos++;

        state = 0;

        now = s[pos];

    }

    else //错误处理3：若一行中没有找到对应的双引号，则报错

    {

        cout << "Line " << NumOfLine << ": error! There is no matching double colon." << endl;

        state = 0;

    }

}

void ProcessBoundarySymbol(string s, int &pos, char &now, int &state)

{

    string variable;

    NumOfBound++;

    variable = now;

    printPair(variable, "boundary");

    pos++;

    state = 0;

    now = s[pos];

}

void ProcessCompareSymbol(string s, int &pos, char &now, int &state)

{

    string variable;

    //对于以上符号有两种情况，单独符号和在后面加等号的情况（如 >=, ==, !=）

    //对于后面加等号的情况需要另判断

    if (s[pos + 1] != '=')

    {

        variable = now;

        printPair(variable, "operator");

    }

    else

    {

        variable = now;

        variable += "=";

        printPair(variable, "operator");

        pos++;

    }

    NumOfOperator++;

    pos++;

    state = 0;

    now = s[pos];

}

void ProcessPlusSymbol(string s, int &pos, char &now, int &state)

{

    string variable;

    if (s[pos + 1] == '=') //"+=" 的情况

    {

        variable = now;

        variable += "=";

        printPair(variable, "operator");

        pos++;

    }

    else if (s[pos + 1] == '+') //"++" 的情况

    {

        variable = now;

        variable += "+";

        printPair(variable, "operator");

        pos++;

    }

    else

    {

        variable = now;

        printPair(variable, "operator");

    }

    NumOfOperator++;

    pos++;

    state = 0;

    now = s[pos];

}

void ProcessMinusSymbol(string s, int &pos, char &now, int &state)

{

    string variable;

    if (s[pos + 1] == '=')

    { //"-=" 的情况

        variable = now;

        variable += "=";

        printPair(variable, "operator");

        pos++;

    }

    else if (s[pos + 1] == '-') //"--" 的情况

    {

        variable = now;

        variable += "-";

        printPair(variable, "operator");

        pos++;

    }

    else

    {

        variable = now;

        printPair(variable, "operator");

    }

    NumOfOperator++;

    pos++;

    state = 0;

    now = s[pos];

}

void ProcessLineSymbol(string s, int &pos, char &now, int &state)

{

    string variable;

    if (s[pos + 1] == '|')

    { //"||" 的情况

        variable = now;

        variable += "|";

        printPair(variable, "operator");

        pos++;

    }

    else

    {

        variable = now;

        printPair(variable, "operator");

    }

    NumOfOperator++;

    pos++;

    state = 0;

    now = s[pos];

}

void ProcessAndSymbol(string s, int &pos, char &now, int &state)

{

    string variable;

    if (s[pos + 1] == '&')

    { //"&&" 的情况

        variable = now;

        variable += "&";

        printPair(variable, "operator");

        pos++;

    }

    else

    {

        variable = now;

        printPair(variable, "operator");

    }

    NumOfOperator++;

    pos++;

    state = 0;

    now = s[pos];

}

void ProcessSlashSymbol(string s, int &pos, char &now, int &state, int &flag)

{

    string variable;

    if (flag)

        goto annotation;

    //对于后面加等号的情况需要另判断，表示除法

    if (s[pos + 1] == '=')

    {

        variable = now;

        variable += "=";

        printPair(variable, "operator");

        NumOfOperator++;

        pos += 2;

    }

    else if (s[pos + 1] == '/')

    { //表示“//”的注释

        cout << "Line " << NumOfLine << ": single-lined annotation" << endl;

        now = '\0';

    }

    else if (s[pos + 1] == '\*')

    { //表示“/\*”的注释

        flag = 1;

        pos += 2;

    annotation:

        while (!(s[pos] == '\*' && s[pos + 1] == '/') && s[pos] != '\0')

        {

            pos++;

        }

        if (s[pos] != '\0') //如果成功找到\*/

        {

            cout << "Line " << NumOfLine << ": long annotation" << endl;

            flag = 0;

            pos += 2;

        }

    }

    else

    { //如果只表示除

        NumOfOperator++;

        variable = now;

        printPair(variable, "operator");

        pos++;

    }

}

void SymbolSituation(string s, int &pos, char &now, int &state, int &flag)

{

    int posVar;

    char var[200]; //用于暂存标识符或关键字

    string variable;

    memset(var, 0, sizeof(var));

    posVar = 0;

    if (flag)

        goto annotation;

    switch (now)

    {

    //下面处理其他符号，遇到这些符号则直接略过

    case ' ':

    case ',':

    case ':':

    case ';':

    case '\n':

    case '?':

    {

        ProcessOtherSymbols(s, pos, now, state);

        break;

    }

    //双引号和单引号都需要单独处理，直到找到其后面的匹配为止

    case '"':

    {

        ProcessDoubleQuotationMarks(s, pos, now, state);

        break;

    }

    case '\'':

    {

        ProcessSingleQuotationMarks(s, pos, now, state);

        break;

    }

    case '#': //有#为宏定义，改行直接跳过

    {

        now = '\0';

        break;

    }

    //下面是对界符的处理

    case '[':

    case ']':

    case '(':

    case ')':

    case '{':

    case '}':

    {

        ProcessBoundarySymbol(s, pos, now, state);

        break;

    }

    //下面是对运算符的处理

    case '>':

    case '<':

    case '=':

    case '!':

    case '\*':

    case '%':

    {

        ProcessCompareSymbol(s, pos, now, state);

        break;

    }

    case '+':

    {

        ProcessPlusSymbol(s, pos, now, state);

        break;

    }

    case '-':

    {

        ProcessMinusSymbol(s, pos, now, state);

        break;

    }

    case '|':

    {

        ProcessLineSymbol(s, pos, now, state);

        break;

    }

    case '&':

    {

        ProcessAndSymbol(s, pos, now, state);

        break;

    }

    case '/': //对斜杠需要特判，判断后面是否有注释

    {

    annotation:

        ProcessSlashSymbol(s, pos, now, state, flag);

        state = 0;

        break;

    }

    default: //对于其他不可识别符号

        cout << "unidentified symbol : " << now << endl;

        pos++;

        state = 0;

        break;

    }

}

void OutputStatistics()

{

    if (SymbolTable.empty())

    {

        cout << "符号表为空" << endl;

    }

    else

    {

        cout << "符号表:" << endl;

        for (auto iter = SymbolTable.begin(); iter != SymbolTable.end(); ++iter)

        {

            cout << "At Line " << iter->second << ": " << iter->first << endl;

        }

    }

    cout << "行数：" << NumOfLine << endl;

    cout << "关键词出现的次数：" << NumOfKeyWord << endl;

    cout << "变量出现的次数：" << NumOfVarible << endl;

    cout << "数字常量出现的次数：" << NumOfNumber << endl;

    cout << "运算符出现的次数：" << NumOfOperator << endl;

    cout << "界符出现的次数：" << NumOfBound << endl;

}

void printPair(string sym, string attrtibute)

{

    cout << "Line " << NumOfLine << ": <" << sym << ", " << attrtibute << ">" << endl;

}

# 附录二：lex.i 代码文件

/\*

\* @project: wordAnalysis(lex version)

\* @IDE:    Visual Studio

\* @Author: Jiang Shutong

\* @Date:   Oct 10th

\*/

%{

#include <stdio.h>

#include <string.h>

/\* 比较运算符 \*/

#define LT                  1

#define LE                  2

#define GT                  3

#define GE                  4

#define EQ                  5

#define PL                  6

#define PP                  7

#define PE                  8

#define MS                  8

#define MM                  9

#define ME                  10

/\* 保留字 \*/

#define IF                  11

#define ELSE                12

#define RETURN              13

#define BREAK               14

#define WHILE               15

#define DO                  16

#define ID                  17

#define NUMBER              18

#define RELOP               19

#define DELIMITER           20

/\* 其他情况 \*/

#define ERRORCHAR           21

int yylval;

char symbolTable[1024][1024];  //表示符号表

int loc = 0;                   //表示符号表的大小

%}

delim       [ \t \n]

ws          {delim}+

letter      [A-Za-z\_]

digit       [0-9]

id          {letter}({letter}|{digit})\*

number      {digit}+(\.{digit}+)?

%%

{ws}              {;/\* 此时词法分析器没有动作，也不返回，而是继续分析。 \*/}

while             {return (WHILE);}

do                {return (DO);}

if                {return (IF);}

else              {return (ELSE);}

return            {return (RETURN);}

break             {return (BREAK);}

{id}              {yylval = installID (); return (ID);}

{number}          {return (NUMBER);}

"<"               {yylval = LT; return (RELOP);}

"<="              {yylval = LE; return (RELOP);}

"="               {yylval = EQ; return (RELOP);}

">"               {yylval = GT; return (RELOP);}

">="              {yylval = GE; return (RELOP);}

"+"               {yylval = PL; return (RELOP);}

"++"              {yylval = PP; return (RELOP);}

"+="              {yylval = PE; return (RELOP);}

"-"               {yylval = MS; return (RELOP);}

"--"              {yylval = MM; return (RELOP);}

"-="              {yylval = ME; return (RELOP);}

"{"               {return (DELIMITER);}

"}"               {return (DELIMITER);}

"("               {return (DELIMITER);}

")"               {return (DELIMITER);}

";"               {return (DELIMITER);}

.                 {yylval = ERRORCHAR; return ERRORCHAR;}

 /\*.匹配除换行之外的任何字符，一般可作为最后一条翻译规则。\*/

%%

int installID () {

    for(int i = 0; i < loc; i ++){

        if(strcmp(symbolTable[i], yytext) == 0) //如果符号表中已经存在

        return ID;

    }

    strcpy(symbolTable[loc], yytext);

    loc ++;

    return ID;

}

int yywrap (){

  return 1;

}

void writeout(int c){

  switch(c){

    case ERRORCHAR: fprintf(yyout, "(ERRORCHAR, \"%s\")\n", yytext);break;

    case RELOP: fprintf(yyout, "(RELOP, \"%s\") \n", yytext);break;

    case DELIMITER: fprintf(yyout, "(DELIMITER, \"%s\") \n", yytext);break;

    case WHILE: fprintf(yyout, "(WHILE, \"%s\")\n", yytext);break;

    case DO: fprintf(yyout, "(DO, \"%s\") \n", yytext);break;

    case IF: fprintf(yyout, "(IF, \"%s\") \n", yytext);break;

    case ELSE: fprintf(yyout, "(ELSE, \"%s\") \n", yytext);break;

    case RETURN: fprintf(yyout, "(RETURN, \"%s\") \n", yytext);break;

    case BREAK: fprintf(yyout, "(BREAK, \"%s\") \n", yytext);break;

    case NUMBER: fprintf(yyout, "(NUM, \"%s\")\n", yytext);break;

    case ID: fprintf(yyout, "(ID, \"%s\")\n", yytext);break;

    default:break;

  }

  return;

}

int main (int argc, char \*\* argv){

    int c,j=0;

    if (argc>=2){

      if ((yyin = fopen(argv[1], "r")) == NULL){

        printf("Can't open file %s\n", argv[1]);

        return 1;

      }

      if (argc>=3){

        yyout=fopen(argv[2], "w");

      }

    }

    while (c = yylex()){

        writeout(c);

        j++;

    }

    if(argc>=2){

      fclose(yyin);

      if (argc>=3) fclose(yyout);

    }

    printf("The symbolTable is listed as follows:\n");

    for(int i = 0; i < loc; i ++){

        printf("the %d symbol is: %s\n", i, symbolTable[i]);

    }

    return 0;

}

# 附录三：lex.yy.c 代码文件

#line 2 "lex.yy.c"

#define  YY\_INT\_ALIGNED short int

/\* A lexical scanner generated by flex \*/

#define FLEX\_SCANNER

#define YY\_FLEX\_MAJOR\_VERSION 2

#define YY\_FLEX\_MINOR\_VERSION 6

#define YY\_FLEX\_SUBMINOR\_VERSION 4

#if YY\_FLEX\_SUBMINOR\_VERSION > 0

#define FLEX\_BETA

#endif

/\* First, we deal with  platform-specific or compiler-specific issues. \*/

/\* begin standard C headers. \*/

#include <stdio.h>

#include <string.h>

#include <errno.h>

#include <stdlib.h>

/\* end standard C headers. \*/

/\* flex integer type definitions \*/

#ifndef FLEXINT\_H

#define FLEXINT\_H

/\* C99 systems have <inttypes.h>. Non-C99 systems may or may not. \*/

#if defined (\_\_STDC\_VERSION\_\_) && \_\_STDC\_VERSION\_\_ >= 199901L

/\* C99 says to define \_\_STDC\_LIMIT\_MACROS before including stdint.h,

 \* if you want the limit (max/min) macros for int types.

 \*/

#ifndef \_\_STDC\_LIMIT\_MACROS

#define \_\_STDC\_LIMIT\_MACROS 1

#endif

#include <inttypes.h>

typedef int8\_t flex\_int8\_t;

typedef uint8\_t flex\_uint8\_t;

typedef int16\_t flex\_int16\_t;

typedef uint16\_t flex\_uint16\_t;

typedef int32\_t flex\_int32\_t;

typedef uint32\_t flex\_uint32\_t;

#else

typedef signed char flex\_int8\_t;

typedef short int flex\_int16\_t;

typedef int flex\_int32\_t;

typedef unsigned char flex\_uint8\_t;

typedef unsigned short int flex\_uint16\_t;

typedef unsigned int flex\_uint32\_t;

/\* Limits of integral types. \*/

#ifndef INT8\_MIN

#define INT8\_MIN               (-128)

#endif

#ifndef INT16\_MIN

#define INT16\_MIN              (-32767-1)

#endif

#ifndef INT32\_MIN

#define INT32\_MIN              (-2147483647-1)

#endif

#ifndef INT8\_MAX

#define INT8\_MAX               (127)

#endif

#ifndef INT16\_MAX

#define INT16\_MAX              (32767)

#endif

#ifndef INT32\_MAX

#define INT32\_MAX              (2147483647)

#endif

#ifndef UINT8\_MAX

#define UINT8\_MAX              (255U)

#endif

#ifndef UINT16\_MAX

#define UINT16\_MAX             (65535U)

#endif

#ifndef UINT32\_MAX

#define UINT32\_MAX             (4294967295U)

#endif

#ifndef SIZE\_MAX

#define SIZE\_MAX               (~(size\_t)0)

#endif

#endif /\* ! C99 \*/

#endif /\* ! FLEXINT\_H \*/

/\* begin standard C++ headers. \*/

/\* TODO: this is always defined, so inline it \*/

#define yyconst const

#if defined(\_\_GNUC\_\_) && \_\_GNUC\_\_ >= 3

#define yynoreturn \_\_attribute\_\_((\_\_noreturn\_\_))

#else

#define yynoreturn

#endif

/\* Returned upon end-of-file. \*/

#define YY\_NULL 0

/\* Promotes a possibly negative, possibly signed char to an

 \*   integer in range [0..255] for use as an array index.

 \*/

#define YY\_SC\_TO\_UI(c) ((YY\_CHAR) (c))

/\* Enter a start condition.  This macro really ought to take a parameter,

 \* but we do it the disgusting crufty way forced on us by the ()-less

 \* definition of BEGIN.

 \*/

#define BEGIN (yy\_start) = 1 + 2 \*

/\* Translate the current start state into a value that can be later handed

 \* to BEGIN to return to the state.  The YYSTATE alias is for lex

 \* compatibility.

 \*/

#define YY\_START (((yy\_start) - 1) / 2)

#define YYSTATE YY\_START

/\* Action number for EOF rule of a given start state. \*/

#define YY\_STATE\_EOF(state) (YY\_END\_OF\_BUFFER + state + 1)

/\* Special action meaning "start processing a new file". \*/

#define YY\_NEW\_FILE yyrestart( yyin  )

#define YY\_END\_OF\_BUFFER\_CHAR 0

/\* Size of default input buffer. \*/

#ifndef YY\_BUF\_SIZE

#ifdef \_\_ia64\_\_

/\* On IA-64, the buffer size is 16k, not 8k.

 \* Moreover, YY\_BUF\_SIZE is 2\*YY\_READ\_BUF\_SIZE in the general case.

 \* Ditto for the \_\_ia64\_\_ case accordingly.

 \*/

#define YY\_BUF\_SIZE 32768

#else

#define YY\_BUF\_SIZE 16384

#endif /\* \_\_ia64\_\_ \*/

#endif

/\* The state buf must be large enough to hold one state per character in the main buffer.

 \*/

#define YY\_STATE\_BUF\_SIZE   ((YY\_BUF\_SIZE + 2) \* sizeof(yy\_state\_type))

#ifndef YY\_TYPEDEF\_YY\_BUFFER\_STATE

#define YY\_TYPEDEF\_YY\_BUFFER\_STATE

typedef struct yy\_buffer\_state \*YY\_BUFFER\_STATE;

#endif

#ifndef YY\_TYPEDEF\_YY\_SIZE\_T

#define YY\_TYPEDEF\_YY\_SIZE\_T

typedef size\_t yy\_size\_t;

#endif

extern int yyleng;

extern FILE \*yyin, \*yyout;

#define EOB\_ACT\_CONTINUE\_SCAN 0

#define EOB\_ACT\_END\_OF\_FILE 1

#define EOB\_ACT\_LAST\_MATCH 2

    #define YY\_LESS\_LINENO(n)

    #define YY\_LINENO\_REWIND\_TO(ptr)

/\* Return all but the first "n" matched characters back to the input stream. \*/

#define yyless(n) \

    do \

        { \

        /\* Undo effects of setting up yytext. \*/ \

        int yyless\_macro\_arg = (n); \

        YY\_LESS\_LINENO(yyless\_macro\_arg);\

        \*yy\_cp = (yy\_hold\_char); \

        YY\_RESTORE\_YY\_MORE\_OFFSET \

        (yy\_c\_buf\_p) = yy\_cp = yy\_bp + yyless\_macro\_arg - YY\_MORE\_ADJ; \

        YY\_DO\_BEFORE\_ACTION; /\* set up yytext again \*/ \

        } \

    while ( 0 )

#define unput(c) yyunput( c, (yytext\_ptr)  )

#ifndef YY\_STRUCT\_YY\_BUFFER\_STATE

#define YY\_STRUCT\_YY\_BUFFER\_STATE

struct yy\_buffer\_state

    {

    FILE \*yy\_input\_file;

    char \*yy\_ch\_buf;        /\* input buffer \*/

    char \*yy\_buf\_pos;       /\* current position in input buffer \*/

    /\* Size of input buffer in bytes, not including room for EOB

     \* characters.

     \*/

    int yy\_buf\_size;

    /\* Number of characters read into yy\_ch\_buf, not including EOB

     \* characters.

     \*/

    int yy\_n\_chars;

    /\* Whether we "own" the buffer - i.e., we know we created it,

     \* and can realloc() it to grow it, and should free() it to

     \* delete it.

     \*/

    int yy\_is\_our\_buffer;

    /\* Whether this is an "interactive" input source; if so, and

     \* if we're using stdio for input, then we want to use getc()

     \* instead of fread(), to make sure we stop fetching input after

     \* each newline.

     \*/

    int yy\_is\_interactive;

    /\* Whether we're considered to be at the beginning of a line.

     \* If so, '^' rules will be active on the next match, otherwise

     \* not.

     \*/

    int yy\_at\_bol;

    int yy\_bs\_lineno; /\*\*< The line count. \*/

    int yy\_bs\_column; /\*\*< The column count. \*/

    /\* Whether to try to fill the input buffer when we reach the

     \* end of it.

     \*/

    int yy\_fill\_buffer;

    int yy\_buffer\_status;

#define YY\_BUFFER\_NEW 0

#define YY\_BUFFER\_NORMAL 1

    /\* When an EOF's been seen but there's still some text to process

     \* then we mark the buffer as YY\_EOF\_PENDING, to indicate that we

     \* shouldn't try reading from the input source any more.  We might

     \* still have a bunch of tokens to match, though, because of

     \* possible backing-up.

     \*

     \* When we actually see the EOF, we change the status to "new"

     \* (via yyrestart()), so that the user can continue scanning by

     \* just pointing yyin at a new input file.

     \*/

#define YY\_BUFFER\_EOF\_PENDING 2

    };

#endif /\* !YY\_STRUCT\_YY\_BUFFER\_STATE \*/

/\* Stack of input buffers. \*/

static size\_t yy\_buffer\_stack\_top = 0; /\*\*< index of top of stack. \*/

static size\_t yy\_buffer\_stack\_max = 0; /\*\*< capacity of stack. \*/

static YY\_BUFFER\_STATE \* yy\_buffer\_stack = NULL; /\*\*< Stack as an array. \*/

/\* We provide macros for accessing buffer states in case in the

 \* future we want to put the buffer states in a more general

 \* "scanner state".

 \*

 \* Returns the top of the stack, or NULL.

 \*/

#define YY\_CURRENT\_BUFFER ( (yy\_buffer\_stack) \

                          ? (yy\_buffer\_stack)[(yy\_buffer\_stack\_top)] \

                          : NULL)

/\* Same as previous macro, but useful when we know that the buffer stack is not

 \* NULL or when we need an lvalue. For internal use only.

 \*/

#define YY\_CURRENT\_BUFFER\_LVALUE (yy\_buffer\_stack)[(yy\_buffer\_stack\_top)]

/\* yy\_hold\_char holds the character lost when yytext is formed. \*/

static char yy\_hold\_char;

static int yy\_n\_chars;      /\* number of characters read into yy\_ch\_buf \*/

int yyleng;

/\* Points to current character in buffer. \*/

static char \*yy\_c\_buf\_p = NULL;

static int yy\_init = 0;     /\* whether we need to initialize \*/

static int yy\_start = 0;    /\* start state number \*/

/\* Flag which is used to allow yywrap()'s to do buffer switches

 \* instead of setting up a fresh yyin.  A bit of a hack ...

 \*/

static int yy\_did\_buffer\_switch\_on\_eof;

void yyrestart ( FILE \*input\_file  );

void yy\_switch\_to\_buffer ( YY\_BUFFER\_STATE new\_buffer  );

YY\_BUFFER\_STATE yy\_create\_buffer ( FILE \*file, int size  );

void yy\_delete\_buffer ( YY\_BUFFER\_STATE b  );

void yy\_flush\_buffer ( YY\_BUFFER\_STATE b  );

void yypush\_buffer\_state ( YY\_BUFFER\_STATE new\_buffer  );

void yypop\_buffer\_state ( void );

static void yyensure\_buffer\_stack ( void );

static void yy\_load\_buffer\_state ( void );

static void yy\_init\_buffer ( YY\_BUFFER\_STATE b, FILE \*file  );

#define YY\_FLUSH\_BUFFER yy\_flush\_buffer( YY\_CURRENT\_BUFFER )

YY\_BUFFER\_STATE yy\_scan\_buffer ( char \*base, yy\_size\_t size  );

YY\_BUFFER\_STATE yy\_scan\_string ( const char \*yy\_str  );

YY\_BUFFER\_STATE yy\_scan\_bytes ( const char \*bytes, int len  );

void \*yyalloc ( yy\_size\_t  );

void \*yyrealloc ( void \*, yy\_size\_t  );

void yyfree ( void \*  );

#define yy\_new\_buffer yy\_create\_buffer

#define yy\_set\_interactive(is\_interactive) \

    { \

    if ( ! YY\_CURRENT\_BUFFER ){ \

        yyensure\_buffer\_stack (); \

        YY\_CURRENT\_BUFFER\_LVALUE =    \

            yy\_create\_buffer( yyin, YY\_BUF\_SIZE ); \

    } \

    YY\_CURRENT\_BUFFER\_LVALUE->yy\_is\_interactive = is\_interactive; \

    }

#define yy\_set\_bol(at\_bol) \

    { \

    if ( ! YY\_CURRENT\_BUFFER ){\

        yyensure\_buffer\_stack (); \

        YY\_CURRENT\_BUFFER\_LVALUE =    \

            yy\_create\_buffer( yyin, YY\_BUF\_SIZE ); \

    } \

    YY\_CURRENT\_BUFFER\_LVALUE->yy\_at\_bol = at\_bol; \

    }

#define YY\_AT\_BOL() (YY\_CURRENT\_BUFFER\_LVALUE->yy\_at\_bol)

/\* Begin user sect3 \*/

typedef flex\_uint8\_t YY\_CHAR;

FILE \*yyin = NULL, \*yyout = NULL;

typedef int yy\_state\_type;

extern int yylineno;

int yylineno = 1;

extern char \*yytext;

#ifdef yytext\_ptr

#undef yytext\_ptr

#endif

#define yytext\_ptr yytext

static yy\_state\_type yy\_get\_previous\_state ( void );

static yy\_state\_type yy\_try\_NUL\_trans ( yy\_state\_type current\_state  );

static int yy\_get\_next\_buffer ( void );

static void yynoreturn yy\_fatal\_error ( const char\* msg  );

/\* Done after the current pattern has been matched and before the

 \* corresponding action - sets up yytext.

 \*/

#define YY\_DO\_BEFORE\_ACTION \

    (yytext\_ptr) = yy\_bp; \

    yyleng = (int) (yy\_cp - yy\_bp); \

    (yy\_hold\_char) = \*yy\_cp; \

    \*yy\_cp = '\0'; \

    (yy\_c\_buf\_p) = yy\_cp;

#define YY\_NUM\_RULES 27

#define YY\_END\_OF\_BUFFER 28

/\* This struct is not used in this scanner,

   but its presence is necessary. \*/

struct yy\_trans\_info

    {

    flex\_int32\_t yy\_verify;

    flex\_int32\_t yy\_nxt;

    };

static const flex\_int16\_t yy\_accept[55] =

    {   0,

        0,    0,   28,   26,    1,    1,   23,   24,   15,   18,

        9,   25,   10,   12,   13,    8,    8,    8,    8,    8,

        8,    8,   21,   22,    1,   16,   17,   19,   20,    0,

        9,   11,   14,    8,    8,    3,    8,    4,    8,    8,

        9,    8,    8,    8,    8,    8,    5,    8,    8,    7,

        8,    2,    6,    0

    } ;

static const YY\_CHAR yy\_ec[256] =

    {   0,

        1,    1,    1,    1,    1,    1,    1,    1,    2,    3,

        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,

        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,

        1,    2,    1,    1,    1,    1,    1,    1,    1,    4,

        5,    1,    6,    1,    7,    8,    1,    9,    9,    9,

        9,    9,    9,    9,    9,    9,    9,    1,   10,   11,

       12,   13,    1,    1,   14,   14,   14,   14,   14,   14,

       14,   14,   14,   14,   14,   14,   14,   14,   14,   14,

       14,   14,   14,   14,   14,   14,   14,   14,   14,   14,

        1,    1,    1,    1,   14,    1,   15,   16,   14,   17,

       18,   19,   14,   20,   21,   14,   22,   23,   14,   24,

       25,   14,   14,   26,   27,   28,   29,   14,   30,   14,

       14,   14,   31,    1,   32,    1,    1,    1,    1,    1,

        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,

        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,

        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,

        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,

        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,

        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,

        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,

        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,

        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,

        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,

        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,

        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,

        1,    1,    1,    1,    1

    } ;

static const YY\_CHAR yy\_meta[33] =

    {   0,

        1,    1,    1,    1,    1,    1,    1,    1,    2,    1,

        1,    1,    1,    2,    2,    2,    2,    2,    2,    2,

        2,    2,    2,    2,    2,    2,    2,    2,    2,    2,

        1,    1

    } ;

static const flex\_int16\_t yy\_base[56] =

    {   0,

        0,    0,   70,   71,   31,   33,   71,   71,   31,   32,

       32,   71,   57,   71,   56,    0,   41,   41,   42,   45,

       45,   42,   71,   71,   43,   71,   71,   71,   71,   52,

       39,   71,   71,    0,   42,    0,   32,    0,   30,   36,

       47,   40,   36,   24,   29,   29,    0,   24,   31,    0,

       18,    0,    0,   71,   36

    } ;

static const flex\_int16\_t yy\_def[56] =

    {   0,

       54,    1,   54,   54,   54,   54,   54,   54,   54,   54,

       54,   54,   54,   54,   54,   55,   55,   55,   55,   55,

       55,   55,   54,   54,   54,   54,   54,   54,   54,   54,

       54,   54,   54,   55,   55,   55,   55,   55,   55,   55,

       54,   55,   55,   55,   55,   55,   55,   55,   55,   55,

       55,   55,   55,    0,   54

    } ;

static const flex\_int16\_t yy\_nxt[104] =

    {   0,

        4,    5,    6,    7,    8,    9,   10,    4,   11,   12,

       13,   14,   15,   16,   16,   17,   18,   19,   16,   16,

       20,   16,   16,   16,   16,   21,   16,   16,   16,   22,

       23,   24,   25,   25,   25,   25,   26,   34,   28,   30,

       31,   53,   27,   29,   25,   25,   30,   31,   52,   51,

       50,   49,   48,   47,   46,   41,   45,   44,   43,   42,

       41,   40,   39,   38,   37,   36,   35,   33,   32,   54,

        3,   54,   54,   54,   54,   54,   54,   54,   54,   54,

       54,   54,   54,   54,   54,   54,   54,   54,   54,   54,

       54,   54,   54,   54,   54,   54,   54,   54,   54,   54,

       54,   54,   54

    } ;

static const flex\_int16\_t yy\_chk[104] =

    {   0,

        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,

        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,

        1,    1,    1,    1,    1,    1,    1,    1,    1,    1,

        1,    1,    5,    5,    6,    6,    9,   55,   10,   11,

       11,   51,    9,   10,   25,   25,   31,   31,   49,   48,

       46,   45,   44,   43,   42,   41,   40,   39,   37,   35,

       30,   22,   21,   20,   19,   18,   17,   15,   13,    3,

       54,   54,   54,   54,   54,   54,   54,   54,   54,   54,

       54,   54,   54,   54,   54,   54,   54,   54,   54,   54,

       54,   54,   54,   54,   54,   54,   54,   54,   54,   54,

       54,   54,   54

    } ;

static yy\_state\_type yy\_last\_accepting\_state;

static char \*yy\_last\_accepting\_cpos;

extern int yy\_flex\_debug;

int yy\_flex\_debug = 0;

/\* The intent behind this definition is that it'll catch

 \* any uses of REJECT which flex missed.

 \*/

#define REJECT reject\_used\_but\_not\_detected

#define yymore() yymore\_used\_but\_not\_detected

#define YY\_MORE\_ADJ 0

#define YY\_RESTORE\_YY\_MORE\_OFFSET

char \*yytext;

#line 1 "mylex.i"

/\*

\* @project: wordAnalysis(lex version)

\* @IDE:    Visual Studio

\* @Author: Jiang Shutong

\* @Date:   Oct 10th

\*/

#line 9 "mylex.i"

#include <stdio.h>

#include <string.h>

/\* 比较运算符 \*/

#define LT                  1

#define LE                  2

#define GT                  3

#define GE                  4

#define EQ                  5

#define PL                  6

#define PP                  7

#define PE                  8

#define MS                  8

#define MM                  9

#define ME                  10

/\* 保留字 \*/

#define IF                  11

#define ELSE                12

#define RETURN              13

#define BREAK               14

#define WHILE               15

#define DO                  16

#define ID                  17

#define NUMBER              18

#define RELOP               19

#define DELIMITER           20

/\* 其他情况 \*/

#define ERRORCHAR           21

int yylval;

char symbolTable[1024][1024];  //表示符号表

int loc = 0;                   //表示符号表的大小

#line 518 "lex.yy.c"

#line 519 "lex.yy.c"

#define INITIAL 0

#ifndef YY\_NO\_UNISTD\_H

/\* Special case for "unistd.h", since it is non-ANSI. We include it way

 \* down here because we want the user's section 1 to have been scanned first.

 \* The user has a chance to override it with an option.

 \*/

#include <unistd.h>

#endif

#ifndef YY\_EXTRA\_TYPE

#define YY\_EXTRA\_TYPE void \*

#endif

static int yy\_init\_globals ( void );

/\* Accessor methods to globals.

   These are made visible to non-reentrant scanners for convenience. \*/

int yylex\_destroy ( void );

int yyget\_debug ( void );

void yyset\_debug ( int debug\_flag  );

YY\_EXTRA\_TYPE yyget\_extra ( void );

void yyset\_extra ( YY\_EXTRA\_TYPE user\_defined  );

FILE \*yyget\_in ( void );

void yyset\_in  ( FILE \* \_in\_str  );

FILE \*yyget\_out ( void );

void yyset\_out  ( FILE \* \_out\_str  );

            int yyget\_leng ( void );

char \*yyget\_text ( void );

int yyget\_lineno ( void );

void yyset\_lineno ( int \_line\_number  );

/\* Macros after this point can all be overridden by user definitions in

 \* section 1.

 \*/

#ifndef YY\_SKIP\_YYWRAP

#ifdef \_\_cplusplus

extern "C" int yywrap ( void );

#else

extern int yywrap ( void );

#endif

#endif

#ifndef YY\_NO\_UNPUT

    static void yyunput ( int c, char \*buf\_ptr  );

#endif

#ifndef yytext\_ptr

static void yy\_flex\_strncpy ( char \*, const char \*, int );

#endif

#ifdef YY\_NEED\_STRLEN

static int yy\_flex\_strlen ( const char \* );

#endif

#ifndef YY\_NO\_INPUT

#ifdef \_\_cplusplus

static int yyinput ( void );

#else

static int input ( void );

#endif

#endif

/\* Amount of stuff to slurp up with each read. \*/

#ifndef YY\_READ\_BUF\_SIZE

#ifdef \_\_ia64\_\_

/\* On IA-64, the buffer size is 16k, not 8k \*/

#define YY\_READ\_BUF\_SIZE 16384

#else

#define YY\_READ\_BUF\_SIZE 8192

#endif /\* \_\_ia64\_\_ \*/

#endif

/\* Copy whatever the last rule matched to the standard output. \*/

#ifndef ECHO

/\* This used to be an fputs(), but since the string might contain NUL's,

 \* we now use fwrite().

 \*/

#define ECHO do { if (fwrite( yytext, (size\_t) yyleng, 1, yyout )) {} } while (0)

#endif

/\* Gets input and stuffs it into "buf".  number of characters read, or YY\_NULL,

 \* is returned in "result".

 \*/

#ifndef YY\_INPUT

#define YY\_INPUT(buf,result,max\_size) \

    if ( YY\_CURRENT\_BUFFER\_LVALUE->yy\_is\_interactive ) \

        { \

        int c = '\*'; \

        int n; \

        for ( n = 0; n < max\_size && \

                 (c = getc( yyin )) != EOF && c != '\n'; ++n ) \

            buf[n] = (char) c; \

        if ( c == '\n' ) \

            buf[n++] = (char) c; \

        if ( c == EOF && ferror( yyin ) ) \

            YY\_FATAL\_ERROR( "input in flex scanner failed" ); \

        result = n; \

        } \

    else \

        { \

        errno=0; \

        while ( (result = (int) fread(buf, 1, (yy\_size\_t) max\_size, yyin)) == 0 && ferror(yyin)) \

            { \

            if( errno != EINTR) \

                { \

                YY\_FATAL\_ERROR( "input in flex scanner failed" ); \

                break; \

                } \

            errno=0; \

            clearerr(yyin); \

            } \

        }\

\

#endif

/\* No semi-colon after return; correct usage is to write "yyterminate();" -

 \* we don't want an extra ';' after the "return" because that will cause

 \* some compilers to complain about unreachable statements.

 \*/

#ifndef yyterminate

#define yyterminate() return YY\_NULL

#endif

/\* Number of entries by which start-condition stack grows. \*/

#ifndef YY\_START\_STACK\_INCR

#define YY\_START\_STACK\_INCR 25

#endif

/\* Report a fatal error. \*/

#ifndef YY\_FATAL\_ERROR

#define YY\_FATAL\_ERROR(msg) yy\_fatal\_error( msg )

#endif

/\* end tables serialization structures and prototypes \*/

/\* Default declaration of generated scanner - a define so the user can

 \* easily add parameters.

 \*/

#ifndef YY\_DECL

#define YY\_DECL\_IS\_OURS 1

extern int yylex (void);

#define YY\_DECL int yylex (void)

#endif /\* !YY\_DECL \*/

/\* Code executed at the beginning of each rule, after yytext and yyleng

 \* have been set up.

 \*/

#ifndef YY\_USER\_ACTION

#define YY\_USER\_ACTION

#endif

/\* Code executed at the end of each rule. \*/

#ifndef YY\_BREAK

#define YY\_BREAK /\*LINTED\*/break;

#endif

#define YY\_RULE\_SETUP \

    YY\_USER\_ACTION

/\*\* The main scanner function which does all the work.

 \*/

YY\_DECL

{

    yy\_state\_type yy\_current\_state;

    char \*yy\_cp, \*yy\_bp;

    int yy\_act;

    if ( !(yy\_init) )

        {

        (yy\_init) = 1;

#ifdef YY\_USER\_INIT

        YY\_USER\_INIT;

#endif

        if ( ! (yy\_start) )

            (yy\_start) = 1; /\* first start state \*/

        if ( ! yyin )

            yyin = stdin;

        if ( ! yyout )

            yyout = stdout;

        if ( ! YY\_CURRENT\_BUFFER ) {

            yyensure\_buffer\_stack ();

            YY\_CURRENT\_BUFFER\_LVALUE =

                yy\_create\_buffer( yyin, YY\_BUF\_SIZE );

        }

        yy\_load\_buffer\_state(  );

        }

    {

#line 49 "mylex.i"

#line 739 "lex.yy.c"

    while ( /\*CONSTCOND\*/1 )        /\* loops until end-of-file is reached \*/

        {

        yy\_cp = (yy\_c\_buf\_p);

        /\* Support of yytext. \*/

        \*yy\_cp = (yy\_hold\_char);

        /\* yy\_bp points to the position in yy\_ch\_buf of the start of

         \* the current run.

         \*/

        yy\_bp = yy\_cp;

        yy\_current\_state = (yy\_start);

yy\_match:

        do

            {

            YY\_CHAR yy\_c = yy\_ec[YY\_SC\_TO\_UI(\*yy\_cp)] ;

            if ( yy\_accept[yy\_current\_state] )

                {

                (yy\_last\_accepting\_state) = yy\_current\_state;

                (yy\_last\_accepting\_cpos) = yy\_cp;

                }

            while ( yy\_chk[yy\_base[yy\_current\_state] + yy\_c] != yy\_current\_state )

                {

                yy\_current\_state = (int) yy\_def[yy\_current\_state];

                if ( yy\_current\_state >= 55 )

                    yy\_c = yy\_meta[yy\_c];

                }

            yy\_current\_state = yy\_nxt[yy\_base[yy\_current\_state] + yy\_c];

            ++yy\_cp;

            }

        while ( yy\_base[yy\_current\_state] != 71 );

yy\_find\_action:

        yy\_act = yy\_accept[yy\_current\_state];

        if ( yy\_act == 0 )

            { /\* have to back up \*/

            yy\_cp = (yy\_last\_accepting\_cpos);

            yy\_current\_state = (yy\_last\_accepting\_state);

            yy\_act = yy\_accept[yy\_current\_state];

            }

        YY\_DO\_BEFORE\_ACTION;

do\_action:  /\* This label is used only to access EOF actions. \*/

        switch ( yy\_act )

    { /\* beginning of action switch \*/

            case 0: /\* must back up \*/

            /\* undo the effects of YY\_DO\_BEFORE\_ACTION \*/

            \*yy\_cp = (yy\_hold\_char);

            yy\_cp = (yy\_last\_accepting\_cpos);

            yy\_current\_state = (yy\_last\_accepting\_state);

            goto yy\_find\_action;

case 1:

/\* rule 1 can match eol \*/

YY\_RULE\_SETUP

#line 51 "mylex.i"

{;/\* 此时词法分析器没有动作，也不返回，而是继续分析。 \*/}

    YY\_BREAK

case 2:

YY\_RULE\_SETUP

#line 52 "mylex.i"

{return (WHILE);}

    YY\_BREAK

case 3:

YY\_RULE\_SETUP

#line 53 "mylex.i"

{return (DO);}

    YY\_BREAK

case 4:

YY\_RULE\_SETUP

#line 54 "mylex.i"

{return (IF);}

    YY\_BREAK

case 5:

YY\_RULE\_SETUP

#line 55 "mylex.i"

{return (ELSE);}

    YY\_BREAK

case 6:

YY\_RULE\_SETUP

#line 56 "mylex.i"

{return (RETURN);}

    YY\_BREAK

case 7:

YY\_RULE\_SETUP

#line 57 "mylex.i"

{return (BREAK);}

    YY\_BREAK

case 8:

YY\_RULE\_SETUP

#line 58 "mylex.i"

{yylval = installID (); return (ID);}

    YY\_BREAK

case 9:

YY\_RULE\_SETUP

#line 59 "mylex.i"

{return (NUMBER);}

    YY\_BREAK

case 10:

YY\_RULE\_SETUP

#line 60 "mylex.i"

{yylval = LT; return (RELOP);}

    YY\_BREAK

case 11:

YY\_RULE\_SETUP

#line 61 "mylex.i"

{yylval = LE; return (RELOP);}

    YY\_BREAK

case 12:

YY\_RULE\_SETUP

#line 62 "mylex.i"

{yylval = EQ; return (RELOP);}

    YY\_BREAK

case 13:

YY\_RULE\_SETUP

#line 63 "mylex.i"

{yylval = GT; return (RELOP);}

    YY\_BREAK

case 14:

YY\_RULE\_SETUP

#line 64 "mylex.i"

{yylval = GE; return (RELOP);}

    YY\_BREAK

case 15:

YY\_RULE\_SETUP

#line 65 "mylex.i"

{yylval = PL; return (RELOP);}

    YY\_BREAK

case 16:

YY\_RULE\_SETUP

#line 66 "mylex.i"

{yylval = PP; return (RELOP);}

    YY\_BREAK

case 17:

YY\_RULE\_SETUP

#line 67 "mylex.i"

{yylval = PE; return (RELOP);}

    YY\_BREAK

case 18:

YY\_RULE\_SETUP

#line 68 "mylex.i"

{yylval = MS; return (RELOP);}

    YY\_BREAK

case 19:

YY\_RULE\_SETUP

#line 69 "mylex.i"

{yylval = MM; return (RELOP);}

    YY\_BREAK

case 20:

YY\_RULE\_SETUP

#line 70 "mylex.i"

{yylval = ME; return (RELOP);}

    YY\_BREAK

case 21:

YY\_RULE\_SETUP

#line 71 "mylex.i"

{return (DELIMITER);}

    YY\_BREAK

case 22:

YY\_RULE\_SETUP

#line 72 "mylex.i"

{return (DELIMITER);}

    YY\_BREAK

case 23:

YY\_RULE\_SETUP

#line 73 "mylex.i"

{return (DELIMITER);}

    YY\_BREAK

case 24:

YY\_RULE\_SETUP

#line 74 "mylex.i"

{return (DELIMITER);}

    YY\_BREAK

case 25:

YY\_RULE\_SETUP

#line 75 "mylex.i"

{return (DELIMITER);}

    YY\_BREAK

case 26:

YY\_RULE\_SETUP

#line 77 "mylex.i"

{yylval = ERRORCHAR; return ERRORCHAR;}

    YY\_BREAK

/\*.匹配除换行之外的任何字符，一般可作为最后一条翻译规则。\*/

case 27:

YY\_RULE\_SETUP

#line 80 "mylex.i"

ECHO;

    YY\_BREAK

#line 933 "lex.yy.c"

case YY\_STATE\_EOF(INITIAL):

    yyterminate();

    case YY\_END\_OF\_BUFFER:

        {

        /\* Amount of text matched not including the EOB char. \*/

        int yy\_amount\_of\_matched\_text = (int) (yy\_cp - (yytext\_ptr)) - 1;

        /\* Undo the effects of YY\_DO\_BEFORE\_ACTION. \*/

        \*yy\_cp = (yy\_hold\_char);

        YY\_RESTORE\_YY\_MORE\_OFFSET

        if ( YY\_CURRENT\_BUFFER\_LVALUE->yy\_buffer\_status == YY\_BUFFER\_NEW )

            {

            /\* We're scanning a new file or input source.  It's

             \* possible that this happened because the user

             \* just pointed yyin at a new source and called

             \* yylex().  If so, then we have to assure

             \* consistency between YY\_CURRENT\_BUFFER and our

             \* globals.  Here is the right place to do so, because

             \* this is the first action (other than possibly a

             \* back-up) that will match for the new input source.

             \*/

            (yy\_n\_chars) = YY\_CURRENT\_BUFFER\_LVALUE->yy\_n\_chars;

            YY\_CURRENT\_BUFFER\_LVALUE->yy\_input\_file = yyin;

            YY\_CURRENT\_BUFFER\_LVALUE->yy\_buffer\_status = YY\_BUFFER\_NORMAL;

            }

        /\* Note that here we test for yy\_c\_buf\_p "<=" to the position

         \* of the first EOB in the buffer, since yy\_c\_buf\_p will

         \* already have been incremented past the NUL character

         \* (since all states make transitions on EOB to the

         \* end-of-buffer state).  Contrast this with the test

         \* in input().

         \*/

        if ( (yy\_c\_buf\_p) <= &YY\_CURRENT\_BUFFER\_LVALUE->yy\_ch\_buf[(yy\_n\_chars)] )

            { /\* This was really a NUL. \*/

            yy\_state\_type yy\_next\_state;

            (yy\_c\_buf\_p) = (yytext\_ptr) + yy\_amount\_of\_matched\_text;

            yy\_current\_state = yy\_get\_previous\_state(  );

            /\* Okay, we're now positioned to make the NUL

             \* transition.  We couldn't have

             \* yy\_get\_previous\_state() go ahead and do it

             \* for us because it doesn't know how to deal

             \* with the possibility of jamming (and we don't

             \* want to build jamming into it because then it

             \* will run more slowly).

             \*/

            yy\_next\_state = yy\_try\_NUL\_trans( yy\_current\_state );

            yy\_bp = (yytext\_ptr) + YY\_MORE\_ADJ;

            if ( yy\_next\_state )

                {

                /\* Consume the NUL. \*/

                yy\_cp = ++(yy\_c\_buf\_p);

                yy\_current\_state = yy\_next\_state;

                goto yy\_match;

                }

            else

                {

                yy\_cp = (yy\_c\_buf\_p);

                goto yy\_find\_action;

                }

            }

        else switch ( yy\_get\_next\_buffer(  ) )

            {

            case EOB\_ACT\_END\_OF\_FILE:

                {

                (yy\_did\_buffer\_switch\_on\_eof) = 0;

                if ( yywrap(  ) )

                    {

                    /\* Note: because we've taken care in

                     \* yy\_get\_next\_buffer() to have set up

                     \* yytext, we can now set up

                     \* yy\_c\_buf\_p so that if some total

                     \* hoser (like flex itself) wants to

                     \* call the scanner after we return the

                     \* YY\_NULL, it'll still work - another

                     \* YY\_NULL will get returned.

                     \*/

                    (yy\_c\_buf\_p) = (yytext\_ptr) + YY\_MORE\_ADJ;

                    yy\_act = YY\_STATE\_EOF(YY\_START);

                    goto do\_action;

                    }

                else

                    {

                    if ( ! (yy\_did\_buffer\_switch\_on\_eof) )

                        YY\_NEW\_FILE;

                    }

                break;

                }

            case EOB\_ACT\_CONTINUE\_SCAN:

                (yy\_c\_buf\_p) =

                    (yytext\_ptr) + yy\_amount\_of\_matched\_text;

                yy\_current\_state = yy\_get\_previous\_state(  );

                yy\_cp = (yy\_c\_buf\_p);

                yy\_bp = (yytext\_ptr) + YY\_MORE\_ADJ;

                goto yy\_match;

            case EOB\_ACT\_LAST\_MATCH:

                (yy\_c\_buf\_p) =

                &YY\_CURRENT\_BUFFER\_LVALUE->yy\_ch\_buf[(yy\_n\_chars)];

                yy\_current\_state = yy\_get\_previous\_state(  );

                yy\_cp = (yy\_c\_buf\_p);

                yy\_bp = (yytext\_ptr) + YY\_MORE\_ADJ;

                goto yy\_find\_action;

            }

        break;

        }

    default:

        YY\_FATAL\_ERROR(

            "fatal flex scanner internal error--no action found" );

    } /\* end of action switch \*/

        } /\* end of scanning one token \*/

    } /\* end of user's declarations \*/

} /\* end of yylex \*/

/\* yy\_get\_next\_buffer - try to read in a new buffer

 \*

 \* Returns a code representing an action:

 \*  EOB\_ACT\_LAST\_MATCH -

 \*  EOB\_ACT\_CONTINUE\_SCAN - continue scanning from current position

 \*  EOB\_ACT\_END\_OF\_FILE - end of file

 \*/

static int yy\_get\_next\_buffer (void)

{

        char \*dest = YY\_CURRENT\_BUFFER\_LVALUE->yy\_ch\_buf;

    char \*source = (yytext\_ptr);

    int number\_to\_move, i;

    int ret\_val;

    if ( (yy\_c\_buf\_p) > &YY\_CURRENT\_BUFFER\_LVALUE->yy\_ch\_buf[(yy\_n\_chars) + 1] )

        YY\_FATAL\_ERROR(

        "fatal flex scanner internal error--end of buffer missed" );

    if ( YY\_CURRENT\_BUFFER\_LVALUE->yy\_fill\_buffer == 0 )

        { /\* Don't try to fill the buffer, so this is an EOF. \*/

        if ( (yy\_c\_buf\_p) - (yytext\_ptr) - YY\_MORE\_ADJ == 1 )

            {

            /\* We matched a single character, the EOB, so

             \* treat this as a final EOF.

             \*/

            return EOB\_ACT\_END\_OF\_FILE;

            }

        else

            {

            /\* We matched some text prior to the EOB, first

             \* process it.

             \*/

            return EOB\_ACT\_LAST\_MATCH;

            }

        }

    /\* Try to read more data. \*/

    /\* First move last chars to start of buffer. \*/

    number\_to\_move = (int) ((yy\_c\_buf\_p) - (yytext\_ptr) - 1);

    for ( i = 0; i < number\_to\_move; ++i )

        \*(dest++) = \*(source++);

    if ( YY\_CURRENT\_BUFFER\_LVALUE->yy\_buffer\_status == YY\_BUFFER\_EOF\_PENDING )

        /\* don't do the read, it's not guaranteed to return an EOF,

         \* just force an EOF

         \*/

        YY\_CURRENT\_BUFFER\_LVALUE->yy\_n\_chars = (yy\_n\_chars) = 0;

    else

        {

            int num\_to\_read =

            YY\_CURRENT\_BUFFER\_LVALUE->yy\_buf\_size - number\_to\_move - 1;

        while ( num\_to\_read <= 0 )

            { /\* Not enough room in the buffer - grow it. \*/

            /\* just a shorter name for the current buffer \*/

            YY\_BUFFER\_STATE b = YY\_CURRENT\_BUFFER\_LVALUE;

            int yy\_c\_buf\_p\_offset =

                (int) ((yy\_c\_buf\_p) - b->yy\_ch\_buf);

            if ( b->yy\_is\_our\_buffer )

                {

                int new\_size = b->yy\_buf\_size \* 2;

                if ( new\_size <= 0 )

                    b->yy\_buf\_size += b->yy\_buf\_size / 8;

                else

                    b->yy\_buf\_size \*= 2;

                b->yy\_ch\_buf = (char \*)

                    /\* Include room in for 2 EOB chars. \*/

                    yyrealloc( (void \*) b->yy\_ch\_buf,

                             (yy\_size\_t) (b->yy\_buf\_size + 2)  );

                }

            else

                /\* Can't grow it, we don't own it. \*/

                b->yy\_ch\_buf = NULL;

            if ( ! b->yy\_ch\_buf )

                YY\_FATAL\_ERROR(

                "fatal error - scanner input buffer overflow" );

            (yy\_c\_buf\_p) = &b->yy\_ch\_buf[yy\_c\_buf\_p\_offset];

            num\_to\_read = YY\_CURRENT\_BUFFER\_LVALUE->yy\_buf\_size -

                        number\_to\_move - 1;

            }

        if ( num\_to\_read > YY\_READ\_BUF\_SIZE )

            num\_to\_read = YY\_READ\_BUF\_SIZE;

        /\* Read in more data. \*/

        YY\_INPUT( (&YY\_CURRENT\_BUFFER\_LVALUE->yy\_ch\_buf[number\_to\_move]),

            (yy\_n\_chars), num\_to\_read );

        YY\_CURRENT\_BUFFER\_LVALUE->yy\_n\_chars = (yy\_n\_chars);

        }

    if ( (yy\_n\_chars) == 0 )

        {

        if ( number\_to\_move == YY\_MORE\_ADJ )

            {

            ret\_val = EOB\_ACT\_END\_OF\_FILE;

            yyrestart( yyin  );

            }

        else

            {

            ret\_val = EOB\_ACT\_LAST\_MATCH;

            YY\_CURRENT\_BUFFER\_LVALUE->yy\_buffer\_status =

                YY\_BUFFER\_EOF\_PENDING;

            }

        }

    else

        ret\_val = EOB\_ACT\_CONTINUE\_SCAN;

    if (((yy\_n\_chars) + number\_to\_move) > YY\_CURRENT\_BUFFER\_LVALUE->yy\_buf\_size) {

        /\* Extend the array by 50%, plus the number we really need. \*/

        int new\_size = (yy\_n\_chars) + number\_to\_move + ((yy\_n\_chars) >> 1);

        YY\_CURRENT\_BUFFER\_LVALUE->yy\_ch\_buf = (char \*) yyrealloc(

            (void \*) YY\_CURRENT\_BUFFER\_LVALUE->yy\_ch\_buf, (yy\_size\_t) new\_size  );

        if ( ! YY\_CURRENT\_BUFFER\_LVALUE->yy\_ch\_buf )

            YY\_FATAL\_ERROR( "out of dynamic memory in yy\_get\_next\_buffer()" );

        /\* "- 2" to take care of EOB's \*/

        YY\_CURRENT\_BUFFER\_LVALUE->yy\_buf\_size = (int) (new\_size - 2);

    }

    (yy\_n\_chars) += number\_to\_move;

    YY\_CURRENT\_BUFFER\_LVALUE->yy\_ch\_buf[(yy\_n\_chars)] = YY\_END\_OF\_BUFFER\_CHAR;

    YY\_CURRENT\_BUFFER\_LVALUE->yy\_ch\_buf[(yy\_n\_chars) + 1] = YY\_END\_OF\_BUFFER\_CHAR;

    (yytext\_ptr) = &YY\_CURRENT\_BUFFER\_LVALUE->yy\_ch\_buf[0];

    return ret\_val;

}

/\* yy\_get\_previous\_state - get the state just before the EOB char was reached \*/

    static yy\_state\_type yy\_get\_previous\_state (void)

{

    yy\_state\_type yy\_current\_state;

    char \*yy\_cp;

    yy\_current\_state = (yy\_start);

    for ( yy\_cp = (yytext\_ptr) + YY\_MORE\_ADJ; yy\_cp < (yy\_c\_buf\_p); ++yy\_cp )

        {

        YY\_CHAR yy\_c = (\*yy\_cp ? yy\_ec[YY\_SC\_TO\_UI(\*yy\_cp)] : 1);

        if ( yy\_accept[yy\_current\_state] )

            {

            (yy\_last\_accepting\_state) = yy\_current\_state;

            (yy\_last\_accepting\_cpos) = yy\_cp;

            }

        while ( yy\_chk[yy\_base[yy\_current\_state] + yy\_c] != yy\_current\_state )

            {

            yy\_current\_state = (int) yy\_def[yy\_current\_state];

            if ( yy\_current\_state >= 55 )

                yy\_c = yy\_meta[yy\_c];

            }

        yy\_current\_state = yy\_nxt[yy\_base[yy\_current\_state] + yy\_c];

        }

    return yy\_current\_state;

}

/\* yy\_try\_NUL\_trans - try to make a transition on the NUL character

 \*

 \* synopsis

 \*  next\_state = yy\_try\_NUL\_trans( current\_state );

 \*/

    static yy\_state\_type yy\_try\_NUL\_trans  (yy\_state\_type yy\_current\_state )

{

    int yy\_is\_jam;

        char \*yy\_cp = (yy\_c\_buf\_p);

    YY\_CHAR yy\_c = 1;

    if ( yy\_accept[yy\_current\_state] )

        {

        (yy\_last\_accepting\_state) = yy\_current\_state;

        (yy\_last\_accepting\_cpos) = yy\_cp;

        }

    while ( yy\_chk[yy\_base[yy\_current\_state] + yy\_c] != yy\_current\_state )

        {

        yy\_current\_state = (int) yy\_def[yy\_current\_state];

        if ( yy\_current\_state >= 55 )

            yy\_c = yy\_meta[yy\_c];

        }

    yy\_current\_state = yy\_nxt[yy\_base[yy\_current\_state] + yy\_c];

    yy\_is\_jam = (yy\_current\_state == 54);

        return yy\_is\_jam ? 0 : yy\_current\_state;

}

#ifndef YY\_NO\_UNPUT

    static void yyunput (int c, char \* yy\_bp )

{

    char \*yy\_cp;

    yy\_cp = (yy\_c\_buf\_p);

    /\* undo effects of setting up yytext \*/

    \*yy\_cp = (yy\_hold\_char);

    if ( yy\_cp < YY\_CURRENT\_BUFFER\_LVALUE->yy\_ch\_buf + 2 )

        { /\* need to shift things up to make room \*/

        /\* +2 for EOB chars. \*/

        int number\_to\_move = (yy\_n\_chars) + 2;

        char \*dest = &YY\_CURRENT\_BUFFER\_LVALUE->yy\_ch\_buf[

                    YY\_CURRENT\_BUFFER\_LVALUE->yy\_buf\_size + 2];

        char \*source =

                &YY\_CURRENT\_BUFFER\_LVALUE->yy\_ch\_buf[number\_to\_move];

        while ( source > YY\_CURRENT\_BUFFER\_LVALUE->yy\_ch\_buf )

            \*--dest = \*--source;

        yy\_cp += (int) (dest - source);

        yy\_bp += (int) (dest - source);

        YY\_CURRENT\_BUFFER\_LVALUE->yy\_n\_chars =

            (yy\_n\_chars) = (int) YY\_CURRENT\_BUFFER\_LVALUE->yy\_buf\_size;

        if ( yy\_cp < YY\_CURRENT\_BUFFER\_LVALUE->yy\_ch\_buf + 2 )

            YY\_FATAL\_ERROR( "flex scanner push-back overflow" );

        }

    \*--yy\_cp = (char) c;

    (yytext\_ptr) = yy\_bp;

    (yy\_hold\_char) = \*yy\_cp;

    (yy\_c\_buf\_p) = yy\_cp;

}

#endif

#ifndef YY\_NO\_INPUT

#ifdef \_\_cplusplus

    static int yyinput (void)

#else

    static int input  (void)

#endif

{

    int c;

    \*(yy\_c\_buf\_p) = (yy\_hold\_char);

    if ( \*(yy\_c\_buf\_p) == YY\_END\_OF\_BUFFER\_CHAR )

        {

        /\* yy\_c\_buf\_p now points to the character we want to return.

         \* If this occurs \*before\* the EOB characters, then it's a

         \* valid NUL; if not, then we've hit the end of the buffer.

         \*/

        if ( (yy\_c\_buf\_p) < &YY\_CURRENT\_BUFFER\_LVALUE->yy\_ch\_buf[(yy\_n\_chars)] )

            /\* This was really a NUL. \*/

            \*(yy\_c\_buf\_p) = '\0';

        else

            { /\* need more input \*/

            int offset = (int) ((yy\_c\_buf\_p) - (yytext\_ptr));

            ++(yy\_c\_buf\_p);

            switch ( yy\_get\_next\_buffer(  ) )

                {

                case EOB\_ACT\_LAST\_MATCH:

                    /\* This happens because yy\_g\_n\_b()

                     \* sees that we've accumulated a

                     \* token and flags that we need to

                     \* try matching the token before

                     \* proceeding.  But for input(),

                     \* there's no matching to consider.

                     \* So convert the EOB\_ACT\_LAST\_MATCH

                     \* to EOB\_ACT\_END\_OF\_FILE.

                     \*/

                    /\* Reset buffer status. \*/

                    yyrestart( yyin );

                    /\*FALLTHROUGH\*/

                case EOB\_ACT\_END\_OF\_FILE:

                    {

                    if ( yywrap(  ) )

                        return 0;

                    if ( ! (yy\_did\_buffer\_switch\_on\_eof) )

                        YY\_NEW\_FILE;

#ifdef \_\_cplusplus

                    return yyinput();

#else

                    return input();

#endif

                    }

                case EOB\_ACT\_CONTINUE\_SCAN:

                    (yy\_c\_buf\_p) = (yytext\_ptr) + offset;

                    break;

                }

            }

        }

    c = \*(unsigned char \*) (yy\_c\_buf\_p);    /\* cast for 8-bit char's \*/

    \*(yy\_c\_buf\_p) = '\0';   /\* preserve yytext \*/

    (yy\_hold\_char) = \*++(yy\_c\_buf\_p);

    return c;

}

#endif  /\* ifndef YY\_NO\_INPUT \*/

/\*\* Immediately switch to a different input stream.

 \* @param input\_file A readable stream.

 \*

 \* @note This function does not reset the start condition to @c INITIAL .

 \*/

    void yyrestart  (FILE \* input\_file )

{

    if ( ! YY\_CURRENT\_BUFFER ){

        yyensure\_buffer\_stack ();

        YY\_CURRENT\_BUFFER\_LVALUE =

            yy\_create\_buffer( yyin, YY\_BUF\_SIZE );

    }

    yy\_init\_buffer( YY\_CURRENT\_BUFFER, input\_file );

    yy\_load\_buffer\_state(  );

}

/\*\* Switch to a different input buffer.

 \* @param new\_buffer The new input buffer.

 \*

 \*/

    void yy\_switch\_to\_buffer  (YY\_BUFFER\_STATE  new\_buffer )

{

    /\* TODO. We should be able to replace this entire function body

     \* with

     \*      yypop\_buffer\_state();

     \*      yypush\_buffer\_state(new\_buffer);

     \*/

    yyensure\_buffer\_stack ();

    if ( YY\_CURRENT\_BUFFER == new\_buffer )

        return;

    if ( YY\_CURRENT\_BUFFER )

        {

        /\* Flush out information for old buffer. \*/

        \*(yy\_c\_buf\_p) = (yy\_hold\_char);

        YY\_CURRENT\_BUFFER\_LVALUE->yy\_buf\_pos = (yy\_c\_buf\_p);

        YY\_CURRENT\_BUFFER\_LVALUE->yy\_n\_chars = (yy\_n\_chars);

        }

    YY\_CURRENT\_BUFFER\_LVALUE = new\_buffer;

    yy\_load\_buffer\_state(  );

    /\* We don't actually know whether we did this switch during

     \* EOF (yywrap()) processing, but the only time this flag

     \* is looked at is after yywrap() is called, so it's safe

     \* to go ahead and always set it.

     \*/

    (yy\_did\_buffer\_switch\_on\_eof) = 1;

}

static void yy\_load\_buffer\_state  (void)

{

        (yy\_n\_chars) = YY\_CURRENT\_BUFFER\_LVALUE->yy\_n\_chars;

    (yytext\_ptr) = (yy\_c\_buf\_p) = YY\_CURRENT\_BUFFER\_LVALUE->yy\_buf\_pos;

    yyin = YY\_CURRENT\_BUFFER\_LVALUE->yy\_input\_file;

    (yy\_hold\_char) = \*(yy\_c\_buf\_p);

}

/\*\* Allocate and initialize an input buffer state.

 \* @param file A readable stream.

 \* @param size The character buffer size in bytes. When in doubt, use @c YY\_BUF\_SIZE.

 \*

 \* @return the allocated buffer state.

 \*/

    YY\_BUFFER\_STATE yy\_create\_buffer  (FILE \* file, int  size )

{

    YY\_BUFFER\_STATE b;

    b = (YY\_BUFFER\_STATE) yyalloc( sizeof( struct yy\_buffer\_state )  );

    if ( ! b )

        YY\_FATAL\_ERROR( "out of dynamic memory in yy\_create\_buffer()" );

    b->yy\_buf\_size = size;

    /\* yy\_ch\_buf has to be 2 characters longer than the size given because

     \* we need to put in 2 end-of-buffer characters.

     \*/

    b->yy\_ch\_buf = (char \*) yyalloc( (yy\_size\_t) (b->yy\_buf\_size + 2)  );

    if ( ! b->yy\_ch\_buf )

        YY\_FATAL\_ERROR( "out of dynamic memory in yy\_create\_buffer()" );

    b->yy\_is\_our\_buffer = 1;

    yy\_init\_buffer( b, file );

    return b;

}

/\*\* Destroy the buffer.

 \* @param b a buffer created with yy\_create\_buffer()

 \*

 \*/

    void yy\_delete\_buffer (YY\_BUFFER\_STATE  b )

{

    if ( ! b )

        return;

    if ( b == YY\_CURRENT\_BUFFER ) /\* Not sure if we should pop here. \*/

        YY\_CURRENT\_BUFFER\_LVALUE = (YY\_BUFFER\_STATE) 0;

    if ( b->yy\_is\_our\_buffer )

        yyfree( (void \*) b->yy\_ch\_buf  );

    yyfree( (void \*) b  );

}

/\* Initializes or reinitializes a buffer.

 \* This function is sometimes called more than once on the same buffer,

 \* such as during a yyrestart() or at EOF.

 \*/

    static void yy\_init\_buffer  (YY\_BUFFER\_STATE  b, FILE \* file )

{

    int oerrno = errno;

    yy\_flush\_buffer( b );

    b->yy\_input\_file = file;

    b->yy\_fill\_buffer = 1;

    /\* If b is the current buffer, then yy\_init\_buffer was \_probably\_

     \* called from yyrestart() or through yy\_get\_next\_buffer.

     \* In that case, we don't want to reset the lineno or column.

     \*/

    if (b != YY\_CURRENT\_BUFFER){

        b->yy\_bs\_lineno = 1;

        b->yy\_bs\_column = 0;

    }

        b->yy\_is\_interactive = file ? (isatty( fileno(file) ) > 0) : 0;

    errno = oerrno;

}

/\*\* Discard all buffered characters. On the next scan, YY\_INPUT will be called.

 \* @param b the buffer state to be flushed, usually @c YY\_CURRENT\_BUFFER.

 \*

 \*/

    void yy\_flush\_buffer (YY\_BUFFER\_STATE  b )

{

        if ( ! b )

        return;

    b->yy\_n\_chars = 0;

    /\* We always need two end-of-buffer characters.  The first causes

     \* a transition to the end-of-buffer state.  The second causes

     \* a jam in that state.

     \*/

    b->yy\_ch\_buf[0] = YY\_END\_OF\_BUFFER\_CHAR;

    b->yy\_ch\_buf[1] = YY\_END\_OF\_BUFFER\_CHAR;

    b->yy\_buf\_pos = &b->yy\_ch\_buf[0];

    b->yy\_at\_bol = 1;

    b->yy\_buffer\_status = YY\_BUFFER\_NEW;

    if ( b == YY\_CURRENT\_BUFFER )

        yy\_load\_buffer\_state(  );

}

/\*\* Pushes the new state onto the stack. The new state becomes

 \*  the current state. This function will allocate the stack

 \*  if necessary.

 \*  @param new\_buffer The new state.

 \*

 \*/

void yypush\_buffer\_state (YY\_BUFFER\_STATE new\_buffer )

{

        if (new\_buffer == NULL)

        return;

    yyensure\_buffer\_stack();

    /\* This block is copied from yy\_switch\_to\_buffer. \*/

    if ( YY\_CURRENT\_BUFFER )

        {

        /\* Flush out information for old buffer. \*/

        \*(yy\_c\_buf\_p) = (yy\_hold\_char);

        YY\_CURRENT\_BUFFER\_LVALUE->yy\_buf\_pos = (yy\_c\_buf\_p);

        YY\_CURRENT\_BUFFER\_LVALUE->yy\_n\_chars = (yy\_n\_chars);

        }

    /\* Only push if top exists. Otherwise, replace top. \*/

    if (YY\_CURRENT\_BUFFER)

        (yy\_buffer\_stack\_top)++;

    YY\_CURRENT\_BUFFER\_LVALUE = new\_buffer;

    /\* copied from yy\_switch\_to\_buffer. \*/

    yy\_load\_buffer\_state(  );

    (yy\_did\_buffer\_switch\_on\_eof) = 1;

}

/\*\* Removes and deletes the top of the stack, if present.

 \*  The next element becomes the new top.

 \*

 \*/

void yypop\_buffer\_state (void)

{

        if (!YY\_CURRENT\_BUFFER)

        return;

    yy\_delete\_buffer(YY\_CURRENT\_BUFFER );

    YY\_CURRENT\_BUFFER\_LVALUE = NULL;

    if ((yy\_buffer\_stack\_top) > 0)

        --(yy\_buffer\_stack\_top);

    if (YY\_CURRENT\_BUFFER) {

        yy\_load\_buffer\_state(  );

        (yy\_did\_buffer\_switch\_on\_eof) = 1;

    }

}

/\* Allocates the stack if it does not exist.

 \*  Guarantees space for at least one push.

 \*/

static void yyensure\_buffer\_stack (void)

{

    yy\_size\_t num\_to\_alloc;

    if (!(yy\_buffer\_stack)) {

        /\* First allocation is just for 2 elements, since we don't know if this

         \* scanner will even need a stack. We use 2 instead of 1 to avoid an

         \* immediate realloc on the next call.

         \*/

      num\_to\_alloc = 1; /\* After all that talk, this was set to 1 anyways... \*/

        (yy\_buffer\_stack) = (struct yy\_buffer\_state\*\*)yyalloc

                                (num\_to\_alloc \* sizeof(struct yy\_buffer\_state\*)

                                );

        if ( ! (yy\_buffer\_stack) )

            YY\_FATAL\_ERROR( "out of dynamic memory in yyensure\_buffer\_stack()" );

        memset((yy\_buffer\_stack), 0, num\_to\_alloc \* sizeof(struct yy\_buffer\_state\*));

        (yy\_buffer\_stack\_max) = num\_to\_alloc;

        (yy\_buffer\_stack\_top) = 0;

        return;

    }

    if ((yy\_buffer\_stack\_top) >= ((yy\_buffer\_stack\_max)) - 1){

        /\* Increase the buffer to prepare for a possible push. \*/

        yy\_size\_t grow\_size = 8 /\* arbitrary grow size \*/;

        num\_to\_alloc = (yy\_buffer\_stack\_max) + grow\_size;

        (yy\_buffer\_stack) = (struct yy\_buffer\_state\*\*)yyrealloc

                                ((yy\_buffer\_stack),

                                num\_to\_alloc \* sizeof(struct yy\_buffer\_state\*)

                                );

        if ( ! (yy\_buffer\_stack) )

            YY\_FATAL\_ERROR( "out of dynamic memory in yyensure\_buffer\_stack()" );

        /\* zero only the new slots.\*/

        memset((yy\_buffer\_stack) + (yy\_buffer\_stack\_max), 0, grow\_size \* sizeof(struct yy\_buffer\_state\*));

        (yy\_buffer\_stack\_max) = num\_to\_alloc;

    }

}

/\*\* Setup the input buffer state to scan directly from a user-specified character buffer.

 \* @param base the character buffer

 \* @param size the size in bytes of the character buffer

 \*

 \* @return the newly allocated buffer state object.

 \*/

YY\_BUFFER\_STATE yy\_scan\_buffer  (char \* base, yy\_size\_t  size )

{

    YY\_BUFFER\_STATE b;

    if ( size < 2 ||

         base[size-2] != YY\_END\_OF\_BUFFER\_CHAR ||

         base[size-1] != YY\_END\_OF\_BUFFER\_CHAR )

        /\* They forgot to leave room for the EOB's. \*/

        return NULL;

    b = (YY\_BUFFER\_STATE) yyalloc( sizeof( struct yy\_buffer\_state )  );

    if ( ! b )

        YY\_FATAL\_ERROR( "out of dynamic memory in yy\_scan\_buffer()" );

    b->yy\_buf\_size = (int) (size - 2);  /\* "- 2" to take care of EOB's \*/

    b->yy\_buf\_pos = b->yy\_ch\_buf = base;

    b->yy\_is\_our\_buffer = 0;

    b->yy\_input\_file = NULL;

    b->yy\_n\_chars = b->yy\_buf\_size;

    b->yy\_is\_interactive = 0;

    b->yy\_at\_bol = 1;

    b->yy\_fill\_buffer = 0;

    b->yy\_buffer\_status = YY\_BUFFER\_NEW;

    yy\_switch\_to\_buffer( b  );

    return b;

}

/\*\* Setup the input buffer state to scan a string. The next call to yylex() will

 \* scan from a @e copy of @a str.

 \* @param yystr a NUL-terminated string to scan

 \*

 \* @return the newly allocated buffer state object.

 \* @note If you want to scan bytes that may contain NUL values, then use

 \*       yy\_scan\_bytes() instead.

 \*/

YY\_BUFFER\_STATE yy\_scan\_string (const char \* yystr )

{

    return yy\_scan\_bytes( yystr, (int) strlen(yystr) );

}

/\*\* Setup the input buffer state to scan the given bytes. The next call to yylex() will

 \* scan from a @e copy of @a bytes.

 \* @param yybytes the byte buffer to scan

 \* @param \_yybytes\_len the number of bytes in the buffer pointed to by @a bytes.

 \*

 \* @return the newly allocated buffer state object.

 \*/

YY\_BUFFER\_STATE yy\_scan\_bytes  (const char \* yybytes, int  \_yybytes\_len )

{

    YY\_BUFFER\_STATE b;

    char \*buf;

    yy\_size\_t n;

    int i;

    /\* Get memory for full buffer, including space for trailing EOB's. \*/

    n = (yy\_size\_t) (\_yybytes\_len + 2);

    buf = (char \*) yyalloc( n  );

    if ( ! buf )

        YY\_FATAL\_ERROR( "out of dynamic memory in yy\_scan\_bytes()" );

    for ( i = 0; i < \_yybytes\_len; ++i )

        buf[i] = yybytes[i];

    buf[\_yybytes\_len] = buf[\_yybytes\_len+1] = YY\_END\_OF\_BUFFER\_CHAR;

    b = yy\_scan\_buffer( buf, n );

    if ( ! b )

        YY\_FATAL\_ERROR( "bad buffer in yy\_scan\_bytes()" );

    /\* It's okay to grow etc. this buffer, and we should throw it

     \* away when we're done.

     \*/

    b->yy\_is\_our\_buffer = 1;

    return b;

}

#ifndef YY\_EXIT\_FAILURE

#define YY\_EXIT\_FAILURE 2

#endif

static void yynoreturn yy\_fatal\_error (const char\* msg )

{

            fprintf( stderr, "%s\n", msg );

    exit( YY\_EXIT\_FAILURE );

}

/\* Redefine yyless() so it works in section 3 code. \*/

#undef yyless

#define yyless(n) \

    do \

        { \

        /\* Undo effects of setting up yytext. \*/ \

        int yyless\_macro\_arg = (n); \

        YY\_LESS\_LINENO(yyless\_macro\_arg);\

        yytext[yyleng] = (yy\_hold\_char); \

        (yy\_c\_buf\_p) = yytext + yyless\_macro\_arg; \

        (yy\_hold\_char) = \*(yy\_c\_buf\_p); \

        \*(yy\_c\_buf\_p) = '\0'; \

        yyleng = yyless\_macro\_arg; \

        } \

    while ( 0 )

/\* Accessor  methods (get/set functions) to struct members. \*/

/\*\* Get the current line number.

 \*

 \*/

int yyget\_lineno  (void)

{

    return yylineno;

}

/\*\* Get the input stream.

 \*

 \*/

FILE \*yyget\_in  (void)

{

        return yyin;

}

/\*\* Get the output stream.

 \*

 \*/

FILE \*yyget\_out  (void)

{

        return yyout;

}

/\*\* Get the length of the current token.

 \*

 \*/

int yyget\_leng  (void)

{

        return yyleng;

}

/\*\* Get the current token.

 \*

 \*/

char \*yyget\_text  (void)

{

        return yytext;

}

/\*\* Set the current line number.

 \* @param \_line\_number line number

 \*

 \*/

void yyset\_lineno (int  \_line\_number )

{

    yylineno = \_line\_number;

}

/\*\* Set the input stream. This does not discard the current

 \* input buffer.

 \* @param \_in\_str A readable stream.

 \*

 \* @see yy\_switch\_to\_buffer

 \*/

void yyset\_in (FILE \*  \_in\_str )

{

        yyin = \_in\_str ;

}

void yyset\_out (FILE \*  \_out\_str )

{

        yyout = \_out\_str ;

}

int yyget\_debug  (void)

{

        return yy\_flex\_debug;

}

void yyset\_debug (int  \_bdebug )

{

        yy\_flex\_debug = \_bdebug ;

}

static int yy\_init\_globals (void)

{

        /\* Initialization is the same as for the non-reentrant scanner.

     \* This function is called from yylex\_destroy(), so don't allocate here.

     \*/

    (yy\_buffer\_stack) = NULL;

    (yy\_buffer\_stack\_top) = 0;

    (yy\_buffer\_stack\_max) = 0;

    (yy\_c\_buf\_p) = NULL;

    (yy\_init) = 0;

    (yy\_start) = 0;

/\* Defined in main.c \*/

#ifdef YY\_STDINIT

    yyin = stdin;

    yyout = stdout;

#else

    yyin = NULL;

    yyout = NULL;

#endif

    /\* For future reference: Set errno on error, since we are called by

     \* yylex\_init()

     \*/

    return 0;

}

/\* yylex\_destroy is for both reentrant and non-reentrant scanners. \*/

int yylex\_destroy  (void)

{

    /\* Pop the buffer stack, destroying each element. \*/

    while(YY\_CURRENT\_BUFFER){

        yy\_delete\_buffer( YY\_CURRENT\_BUFFER  );

        YY\_CURRENT\_BUFFER\_LVALUE = NULL;

        yypop\_buffer\_state();

    }

    /\* Destroy the stack itself. \*/

    yyfree((yy\_buffer\_stack) );

    (yy\_buffer\_stack) = NULL;

    /\* Reset the globals. This is important in a non-reentrant scanner so the next time

     \* yylex() is called, initialization will occur. \*/

    yy\_init\_globals( );

    return 0;

}

/\*

 \* Internal utility routines.

 \*/

#ifndef yytext\_ptr

static void yy\_flex\_strncpy (char\* s1, const char \* s2, int n )

{

    int i;

    for ( i = 0; i < n; ++i )

        s1[i] = s2[i];

}

#endif

#ifdef YY\_NEED\_STRLEN

static int yy\_flex\_strlen (const char \* s )

{

    int n;

    for ( n = 0; s[n]; ++n )

        ;

    return n;

}

#endif

void \*yyalloc (yy\_size\_t  size )

{

            return malloc(size);

}

void \*yyrealloc  (void \* ptr, yy\_size\_t  size )

{

    /\* The cast to (char \*) in the following accommodates both

     \* implementations that use char\* generic pointers, and those

     \* that use void\* generic pointers.  It works with the latter

     \* because both ANSI C and C++ allow castless assignment from

     \* any pointer type to void\*, and deal with argument conversions

     \* as though doing an assignment.

     \*/

    return realloc(ptr, size);

}

void yyfree (void \* ptr )

{

            free( (char \*) ptr );   /\* see yyrealloc() for (char \*) cast \*/

}

#define YYTABLES\_NAME "yytables"

#line 80 "mylex.i"

int installID () {

    for(int i = 0; i < loc; i ++){

        if(strcmp(symbolTable[i], yytext) == 0) //如果符号表中已经存在

        return ID;

    }

    strcpy(symbolTable[loc], yytext);

    //printf("at loc %d, symbol is %s", loc, symbolTable[loc]);

    loc ++;

    return ID;

}

int yywrap (){

  return 1;

}

void writeout(int c){

  switch(c){

    case ERRORCHAR: fprintf(yyout, "(ERRORCHAR, \"%s\")\n", yytext);break;

    case RELOP: fprintf(yyout, "(RELOP, \"%s\") \n", yytext);break;

    case DELIMITER: fprintf(yyout, "(DELIMITER, \"%s\") \n", yytext);break;

    case WHILE: fprintf(yyout, "(WHILE, \"%s\")\n", yytext);break;

    case DO: fprintf(yyout, "(DO, \"%s\") \n", yytext);break;

    case IF: fprintf(yyout, "(IF, \"%s\") \n", yytext);break;

    case ELSE: fprintf(yyout, "(ELSE, \"%s\") \n", yytext);break;

    case RETURN: fprintf(yyout, "(RETURN, \"%s\") \n", yytext);break;

    case BREAK: fprintf(yyout, "(BREAK, \"%s\") \n", yytext);break;

    case NUMBER: fprintf(yyout, "(NUM, \"%s\")\n", yytext);break;

    case ID: fprintf(yyout, "(ID, \"%s\")\n", yytext);break;

    default:break;

  }

  return;

}

int main (int argc, char \*\* argv){

    int c,j=0;

    if (argc>=2){

      if ((yyin = fopen(argv[1], "r")) == NULL){

        printf("Can't open file %s\n", argv[1]);

        return 1;

      }

      if (argc>=3){

        yyout=fopen(argv[2], "w");

      }

    }

    while (c = yylex()){

        writeout(c);

        j++;

    }

    if(argc>=2){

      fclose(yyin);

      if (argc>=3) fclose(yyout);

    }

    printf("The symbolTable is listed as follows:\n");

    for(int i = 0; i < loc; i ++){

        printf("the %d symbol is: %s\n", i, symbolTable[i]);

    }

    return 0;

}