**编译原理实验报告**

实验名称：词法分析器

指导教师：王莹洁

专业班级：计198-2

姓 名：刘鑫

学 号：201958508239

电子邮件：1579664489@qq.com

实验地点：科技馆5405

实验成绩：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

日期：2021年 10月 29日

**一、实验题目**

设计、编制、调试一个识别一简单语言单词的词法分析程序。程序能够识别基本字、标识符、无符号整数、浮点数、运算符和界符）。单词符号及种别表如下：

|  |  |
| --- | --- |
| 单词符号 | 种别编码 |
| begin | 1 |
| if | 2 |
| then | 3 |
| while | 4 |
| do | 5 |
| end | 6 |
| l(l|d)\* | 10 |
| dd\* | 11 |
| + | 13 |
| - | 14 |
| \* | 15 |
| / | 16 |
| : | 17 |
| := | 18 |
| < | 20 |
| <> | 21 |
| <= | 22 |
| > | 23 |
| >= | 24 |
| = | 25 |
| ; | 26 |
| ( | 27 |
| ) | 28 |
| # | 0 |

**二、实验目的**

设计、编制并调试一个词法分析程序，加深对词法分析原理的理解。

**三、实验要求**

词法分析程序需具备词法分析的功能：

输入：所给文法的源程序字符串。（字符串以“#”号结束）

输出：二元组（syn,token或sum）构成的序列。

其中：syn为单词种别码；

token为存放的单词自身字符串；

sum为整型常数。

例如：对源程序begin x:=9: if x>9 then x:=2\*x+1/3; end #的源文件，经过词法分析后输出如下序列：

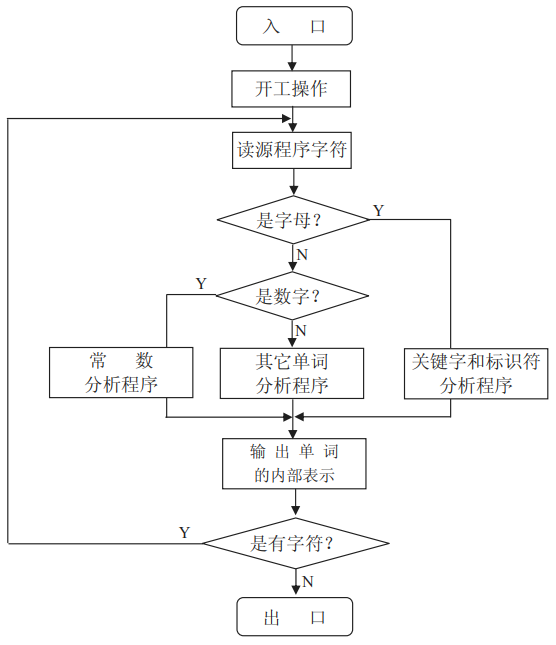
(1,begin)(10,x)(18,:=)(11,9)(26,;)(2,if)……

**四、实验步骤**

（包括基本设计思路、流程框图、算法设计、函数相关说明、输入与输出以及程序运行结果）

设计思路：将关键字和算符用哈希表存储，然后逐字符扫描。如果是单个算符（+\*/-等）或#则直接返回对应的Token；对于复合算符（<>>=等）情况，需要再读取下一个字符来确定应该是单个算符还是复合算符，返回对于的Token；如果是数字则继续读取下一个字符，并不断更新当前数值，直到下一个字符不是数字；如果是字母，则继续读取下一个字符，并更新当前字符串，直到下一个字符既不是数字也不是字母的时候停止，判断哈希表中是否有该字符串，如果有则作为关键字返回，否则作为标识符返回。

流程框图：



函数说明：

Lexer::scan():实现关键字、标识符、数字或其他单词的分析过程;

Lexer::readch():读取一个字符;

bool Lexer::readch(char f):读取下一个字符并判断是否为f;

输入：begin x:=9: if x>9 then x:=2\*x+1/3; end #

程序输出：(1,begin)(10,x)(18,:=)(11,9)(17,:)(2,if)(10,x)(23,>)(11,9)(3,then)(10,x)(18,:=)(11,2)(15,\*)(10,x)(13,+)(11,1)(16,/)(11,3)(26,;)(6,end)(0,#)

**五、实验心得体会**

通过动手学习编写词法分析器，巩固了词法分析器的任务以及作用，就是将去除源文件中的空格以及换行并翻译成若干个词法单元，词法单元以（种别编码，值）的形式组成，方便语法分析进行语法分析，方便构造语法树， 计算相关属性值。

**六、源程序清单（代码）**

#include <iostream>

#include <cstdio>

#include <cstring>

#include <string>

#include <unordered\_map>

using namespace std;

unordered\_map<string, int> Tag = {{"begin", 1}, {"if", 2}, {"then", 3},

{"while", 4}, {"do", 5}, {"end", 6},

{"id", 10}, {"num", 11}, {"+", 13}, {"-", 14},

{"\*", 15}, {"/", 16}, {":", 17}, {":=", 18},

{"<", 20}, {"<>", 21}, {"<=", 22}, {">", 23},

{">=", 24}, {"=", 25}, {";", 26},

{"(", 27}, {")", 28}, {"#", 0}};

typedef pair<int, string> PIS;

const PIS ERROR = {-1, "error"};

class Lexer

{

public:

int line = 1;

PIS scan()

{

// 跳过空格，制表符和换行

for (; !isError; readch())

{

if (peak == ' ' || peak == '\t')

continue;

else if (peak == '\n')

line++;

else

break;

}

if (isError)

return ERROR;

// 处理基本运算符

switch (peak)

{

case '<':

if (readch('='))

return {Tag["<="], "<="};

else if (peak == '>')

return {Tag["<>"], "<>"};

else

return {Tag["<"], "<"};

case ':':

if (readch('='))

return {Tag[":="], ":="};

else

return {Tag[":"], ":"};

case '>':

if (readch('='))

return {Tag[">="], ">="};

else

return {Tag[">"], ">"};

case '=':

peak = ' ';

return {Tag["="], "="};

case '+':

peak = ' ';

return {Tag["+"], "+"};

case '-':

peak = ' ';

return {Tag["-"], "-"};

case '\*':

peak = ' ';

return {Tag["\*"], "\*"};

case '/':

peak = ' ';

return {Tag["/"], "/"};

case ';':

peak = ' ';

return {Tag[";"], ";"};

case '(':

peak = ' ';

return {Tag["("], "("};

case ')':

peak = ' ';

return {Tag[")"], ")"};

case '#':

peak = ' ';

return {Tag["#"], "#"};

}

// 处理数字

if (isdigit(peak))

{

int v = 0;

do

{

v = 10 \* v + peak - '0';

readch();

} while (isdigit(peak));

return {Tag["num"], to\_string(v)};

}

// 处理关键字和标识符

if (isalpha(peak))

{

string buffer;

do

{

buffer.push\_back(peak);

readch();

} while (isalpha(peak) || isdigit(peak));

if (Tag.count(buffer) > 0)

return {Tag[buffer], buffer};

return {Tag["id"], buffer};

}

return ERROR;

}

private:

char peak = ' ';

bool isError = false;

void readch()

{

if (!cin.get(peak))

peak = ' ', isError = true;

}

bool readch(char c)

{

readch();

if (c != peak)

return false;

peak = ' ';

return true;

}

};

int main()

{

Lexer lex;

freopen("in.txt", "r", stdin);

freopen("out.txt", "w", stdout);

PIS curToken;

while ((curToken = lex.scan()) != ERROR)

cout << "(" + to\_string(curToken.first) +

"," + curToken.second + ")";

return 0;

}