**词法分析器设计**

**1. 实验目的**

通过本实验的编程实践，了解词法分析的任务，掌握词法分析程序设计的原理和构造方法，对编译的基本概念、原理和方法有完整的和清楚的理解，并能正确地、熟练地运用。

**2. 实验内容**

用C++语言实现对C++语言子集的源程序进行词法分析。通过输入C++源程序从左到右对字符串进行扫描和分解，依次输出各个单词所属类别及单词本身的值；若遇到无法识别的单词则显示“Unknown”，然后跳过错误部分继续显示；同时进行标识符登记符号表的管理。

**3. 实验原理**

本次实验采用NFA->DFA->minimise DFA的过程:

对待分析的简单的词法（关键词/id/num/运算符/空白符等）先分别建立自己的NFA，然后将他们用产生式连接起来并设置一个唯一的开始符，终结符不合并。

词法可以分为以下几个基本部分

（1）关键字(keyword)：

"include","define","auto","bool","break","case","catch","char","class",

"const","const\_cast","continue","default","delete","do","double",

"dynamic\_cast","else","enum","explicit","extern","false","float","for",

"friend","goto","if","inline","int","long","mutable","namespace","new",

"operator","private","protected","public","register","reinterpret\_cast",

"return","short","signed","sizeof","static","static\_cast","struct",

"switch","template","this","throw","true","try","typedef","typeid",

"typename","union","unsigned","using","virtual","void","volatile","while"

（2）分隔符（separator）

",",";",".","\'","\"","(",")","[","]","{","}","//","/\*","\*/","#"

（3）运算符(operator)

"+","-","\*","/","%","=","==","!=","<","<=",">",">="

（4）标识符（identifier）

（5）数字（number）

（6）注释（note）

（7）字符串常量（constant）

（8）空格有空白、制表符和换行符组成。空格一般用来分隔ID、SUM、运算符、界符和关键字，词法分析阶段通常被忽略。

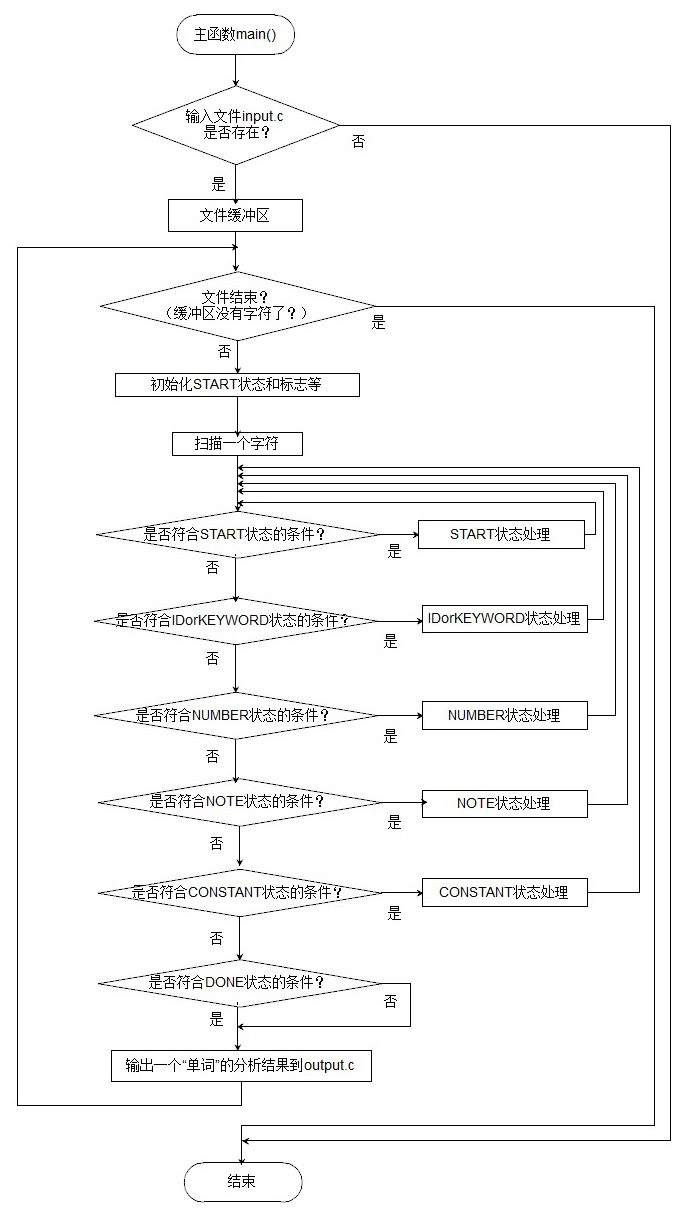
若无法按照以上自动机读取，则认为是未知符号。

**4. 相关自动机描述**

NFA:

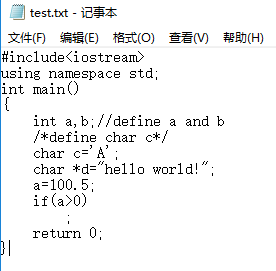
Minimize DFA：

1. **流程图**

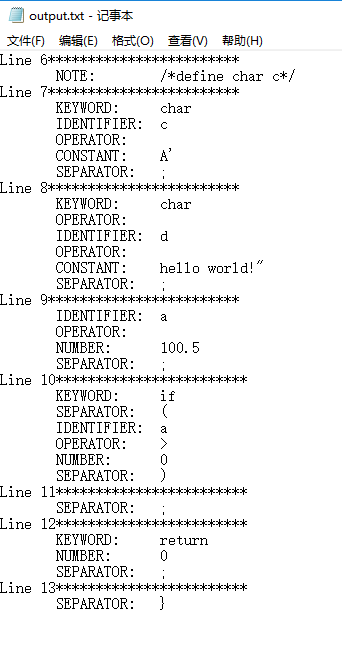
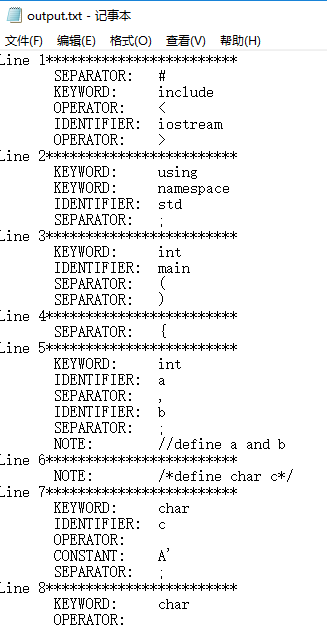


**测试用例**

输入代码：



输出结果：



**8. 出现的问题与解决方案**

本实验的难点就是进行有效地进行状态如转换，先对每一个简单部分，如空白符、id、digit等画出自动机状态，然后由NFA->DFA，添加一个唯一的初始状态，产生式连接。再将DFA中等价的状态合并最后变成DFA0。这样便大大简化了代码量，也使得逻辑思维更加清晰。

**9. 自我体会**

将理论运用到实际，不仅可以帮我们更好地复习理论知识，还可以让我们发发掘到一些更深刻层面上的东西。通过本次实验，我深入了解了词法分析的过程，对NFA,DFA,DFA0之间的转换也更能更加熟练地运用。这次实验还有许多需要加强的地方，比如还可以对id的类型进行明确分类，是函数还是变量，是什么类型的，返回类型是什么等等。之后有机会的话，我一定会更加深入的研究，将这个实验更加完善。

完整代码：

#include "stdio.h"

#include "string.h"

#include<stdlib.h>

#include<iostream>

using namespace std;

#define MAX\_STR 256

#define MAX\_KEYWORDS 62

#define MAX\_OPERATORS 12

#define MAX\_SEPARATORS 15

#define STA\_START 1 /\* START: 开始符号 \*/

#define STA\_IDorKEYWORD 2 /\* IDENTIFIER： 标识符 \*/

#define STA\_NUMBER 3 /\* NUMBER： 数字 \*/

#define STA\_NOTE 4 /\* NOTE: 注释 \*/

#define STA\_CONSTANT 5 /\* CONSTANT: 常量 \*/

#define STA\_DONE 6 /\* DONE： 完成 \*/

#define TYPE\_KEYWORD 1 /\* KEYWORD： 保留字 \*/

#define TYPE\_IDENTIFIER 2 /\* IDENTIFIER： 标识符 \*/

#define TYPE\_NUMBER 3 /\* NUMBER： 数字 \*/

#define TYPE\_NOTE 4 /\* NOTE: 注释 \*/

#define TYPE\_CONSTANT 5 /\* CONSTANT: 常量 \*/

#define TYPE\_OPERATOR 6 /\* OPERATOR: 运算符 \*/

#define TYPE\_SEPARATOR 7 /\* SEPARATOR: 分隔符 \*/

#define TYPE\_ERROR 8 /\* ERROR: 错误 \*/

#define TYPE\_UNKNOWN 9 /\* UNKNOWN: 未知 \*/

#define TYPE\_ENDFILE 10 /\* ENDFILE: 文件结束 \*/

char \*Operators[MAX\_OPERATORS] = { "+","-","\*","/","%","=","==","!=","<","<=",">",">=" };

char \*Separators[MAX\_SEPARATORS] = { ",",";",".","\'","\"","(",")","[","]","{","}","//","/\*","\*/","#" };

char \*Keywords[MAX\_KEYWORDS] = { "include","define","auto","bool","break","case","catch","char","class",

"const","const\_cast","continue","default","delete","do","double",

"dynamic\_cast","else","enum","explicit","extern","false","float","for",

"friend","goto","if","inline","int","long","mutable","namespace","new",

"operator","private","protected","public","register","reinterpret\_cast",

"return","short","signed","sizeof","static","static\_cast","struct",

"switch","template","this","throw","true","try","typedef","typeid",

"typename","union","unsigned","using","virtual","void","volatile","while" };

bool IsOperator(char c);

bool IsSeparator(char c);

bool IsKeyWord(char \*str);

bool IsAlpha(char c);

bool IsNum(char c);

void OutputOneDone(FILE\*outf, int type, char \* str);

void LexAnalyse(FILE \*inf, FILE \*outf);

int main()

{

FILE \* inputfile, \*outputfile;

if ((inputfile = fopen("test.txt", "r")) == NULL){

cout << "File doesn't exist!" << endl;

system("pause");

return 0;

}

else{

outputfile = fopen("output.txt", "w");

LexAnalyse(inputfile, outputfile);

fclose(inputfile);

fclose(outputfile);

cout << "Finish lexis analysis" << endl;

}

system("pause");

return 0;

}

bool IsOperator(char c)

{

for (int i = 0; i < MAX\_OPERATORS; i++)

if (Operators[i][0] == c)

return true;

return false;

}

bool IsSeparator(char c)

{

for (int i = 0; i < MAX\_SEPARATORS; i++)

if (Separators[i][0] == c)

return true;

return false;

}

bool IsKeyWord(char \* str)

{

for (int i = 0; i < MAX\_KEYWORDS; i++){

if (strcmp(Keywords[i], str) == 0)

return true;

}

return false;

}

bool IsAlpha(char c)

{

if ((c >= 'a'&&c <= 'z') || (c >= 'A'&&c <= 'Z'))

return true;

return false;

}

bool IsNum(char c)

{

if (c >= '0'&&c <= '9')

return true;

return false;

}

void OutputOneDone(FILE \* outf, int type, char \* str)

{

if (IsKeyWord(str))

type = TYPE\_KEYWORD;

switch (type){

case TYPE\_KEYWORD: fprintf(outf, " KEYWORD: "); break;

case TYPE\_IDENTIFIER: fprintf(outf, " IDENTIFIER: "); break;

case TYPE\_NUMBER: fprintf(outf, " NUMBER: "); break;

case TYPE\_NOTE: fprintf(outf, " NOTE: "); break;

case TYPE\_CONSTANT: fprintf(outf, " CONSTANT: "); break;

case TYPE\_OPERATOR: fprintf(outf, " OPERATOR: "); break;

case TYPE\_SEPARATOR: fprintf(outf, " SEPARATOR: "); break;

case TYPE\_ERROR: fprintf(outf, " ERROR: "); break;

case TYPE\_UNKNOWN: fprintf(outf, " UNKNOWN: "); break;

default:break;

}

fprintf(outf, "%s\n", str);

}

void LexAnalyse(FILE \* inf, FILE \* outf)

{

char ch;

char str[MAX\_STR] = "";

int i, line\_num = 1;

int state, type;

char flag\_noteline\_one = 0;

char flag\_dot = 0;

fprintf(outf, "Line %d\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n", line\_num);

while (!feof(inf))

{

i = 0;

state = STA\_START; //innitially state

flag\_noteline\_one = 0;

flag\_dot = 0;

while (state != STA\_DONE) //still search

{

ch = fgetc(inf); //get current char

switch (state)

{

case STA\_START:

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

{}

else if (ch == '\n'){

line\_num++;

fprintf(outf, "Line %d\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n", line\_num);

}

else if (IsAlpha(ch)){

state = STA\_IDorKEYWORD;

type = TYPE\_IDENTIFIER;

str[i] = ch;

++i;

}

else if (IsNum(ch)){

state = STA\_NUMBER;

type = TYPE\_NUMBER;

str[i] = ch;

++i;

}

else if (ch == '/'){

str[i] = ch;

i++;

ch = fgetc(inf);

if (ch == '/') { //note //

flag\_noteline\_one = 1;

state = STA\_NOTE;

type = TYPE\_NOTE;

str[i] = ch;

i++;

}

else if (ch == '\*') { // note /\*

state = STA\_NOTE;

type = TYPE\_NOTE;

str[i] = ch;

i++;

}

else //divide operator like / in 1/3

{

state = STA\_DONE;

type = TYPE\_OPERATOR;

fseek(inf, -1, SEEK\_CUR);

i = 1;

str[1] = '\0';

}

}

else if (ch == '<' || ch == '>'){

state = STA\_DONE;

type = TYPE\_OPERATOR;

str[0] = ch;

ch = fgetc(inf);

if (ch == '='){

str[1] = ch;

str[2] = '\0';

i = 2;

}

else{

fseek(inf, -1, SEEK\_CUR);

i = 1;

}

}

else if (ch == '!'){

state = STA\_DONE;

str[0] = ch;

ch = fgetc(inf);

if (ch == '=')

{

type = TYPE\_OPERATOR; // recognize !=

str[1] = '=';

str[2] = '\0';

i = 2;

}

else{

type = TYPE\_UNKNOWN;

fseek(inf, -1, SEEK\_CUR);

i = 1;

}

}

else if (ch == '\"' || ch == '\''){

state = STA\_CONSTANT;

type = TYPE\_CONSTANT;

}

else if (IsOperator(ch)){

state = STA\_DONE;

type = TYPE\_OPERATOR;

str[0] = 0;

i = 1;

}

else if (IsSeparator(ch)){

state = STA\_DONE;

type = TYPE\_SEPARATOR;

str[0] = ch;

i = 1;

}

else if (ch == EOF){

state = STA\_DONE;

type = TYPE\_ENDFILE;

}

else{

state = STA\_DONE;

type = TYPE\_UNKNOWN;

}

break; //end of case state STA\_START

case STA\_IDorKEYWORD:

if (IsAlpha(ch)) { //start with aplha

str[i] = ch;

i++;

}

else if (ch == '.'){//such as a.b

str[i] = ch;

i++;

ch = fgetc(inf);

if (IsAlpha(ch)){

str[i] == ch;

i++;

}

else if (ch == ' ' || ch == '\t' || ch == '\n' || IsOperator(ch) || IsSeparator(ch)){

fseek(inf, -1, SEEK\_CUR);

state = STA\_DONE;

type = TYPE\_ERROR; //like a. a.1

}

else{

str[i] = ch;

i++;

type = TYPE\_ERROR;

}

} //end of ch=='.'

else if (ch == ' ' || ch == '\t' || ch == '\n' || IsOperator(ch) || IsSeparator(ch)){

state = STA\_DONE;

fseek(inf, -1, SEEK\_CUR);

}

else{

state = STA\_DONE;

type = TYPE\_ERROR;

}

break;

case STA\_NUMBER:

if (IsNum(ch)){

str[i] = ch;

i++;

}

else if (ch == '.'){

str[i] = ch;

i++;

ch = fgetc(inf); //next char

if (flag\_dot == 0){

if (IsNum(ch)){

str[i] = ch;

i++;

flag\_dot = 1;

}

else{

fseek(inf, -1, SEEK\_CUR);

state = STA\_DONE;

type = TYPE\_ERROR;

}

}

else{

type = TYPE\_ERROR;

str[i] = ch;

i++;

}

}//end of ch=='.'

else if (ch == ' ' || ch == '\t' || ch == '\n' || IsOperator(ch) || IsSeparator(ch)){

state = STA\_DONE;

fseek(inf, -1, SEEK\_CUR);

}

else{

type = TYPE\_ERROR;

str[i] = ch;

i++;

}

break;

case STA\_NOTE:

if (flag\_noteline\_one == 1) { // note //

if (ch == '\n'){

state = STA\_DONE;

fseek(inf, -1, SEEK\_CUR);

}

else{

str[i] = ch;

i++;

}

}//end of flag==1

else {// note /\*

if (feof(inf)){

state = STA\_DONE;

type = TYPE\_ERROR;

}

else if (ch == '\n'){

line\_num++;

str[i] = ch;

i++;

}

else if (ch == '\*'){

str[i] = ch;

i++;

ch = fgetc(inf);

if (ch == '/') {

state = STA\_DONE;

str[i] = ch;

i++;

}

else{

if (feof(inf)){

state = STA\_DONE;

type = TYPE\_ERROR;

}

fseek(inf, -1, SEEK\_CUR);

}

} //end of ch==\*

else{

str[i] = ch;

i++;

}

}

break;

case STA\_CONSTANT:

if (feof(inf)){

state = STA\_DONE;

type = TYPE\_ERROR;

}

else if (ch == '\n'){

line\_num++;

str[i] = ch;

i++;

}

else if (ch == '\"' || ch == '\''){

state = STA\_DONE;

str[i] = ch;

i++;

}

else if (ch == '\\'){

str[i] = ch;

i++;

ch = fgetc(inf);

str[i] = ch;

i++;

}

else{

str[i] = ch;

i++;

}

break;

case STA\_DONE:

break;

default:

break;

}//END OF switch

}//end of inner while

str[i] = '\0';//end of string

OutputOneDone(outf, type, str);

}//end of out while

}//end of function