北京郵電大學

C 语言词法分析器

实验报告

班　　级：_____

姓　　名：_____

学　　号：_____

专　　业：计算机科学与技术

2018 年 11 月 4 日

目录

一、 实验目的

1) 能够采用 C 编程语言实现简单的词法分析程序，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；
2) 巩固对词法分析的基本功能和原理的认识，理解并处理词法分析中的异常和错误；
3) 掌握在对程序设计语言源程序进行扫描过程中将其分解为各类单词的词法分析方法；

二、 实验要求

设计并实现 C 语言的词法分析程序，要求如下：

（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）可以识别并读取源程序中的注释；

（3）可以统计源程序中的语句行数、单词个数和字符个数，其中标点和空格不计算为单词，并输出统计结果；

（4）检查源程序中存在的错误，并可以报告错误所在的行列位置；

（5）发现源程序中存在错误后，进行适当的恢复，使词法分析可以继续进行，通过一次词法分析处理，可以检查并报告源程序中存在的所有错误。

三、 实验环境

- Windows10 家庭版
- Dev-C++，TDM-GCC 4.9.2 64-bit Debug
- C/C++ 语言

四、 实验原理

4.1 概述

C 语言词法分析的主要任务是从左至右逐个字符地对源程序进行扫描，按照 C 语言词法的规则识别出一个个单词符号，把识别出来的标识符存入符号表中，并产生用于语法分析的记号序列，在词法分析过程中，还可以完成用户接口有关的一些任务，如跳过源程序的注释和空格，能够检测词法异常，并统计源代码的一些信息。

因此，词法分析的核心是**确定识别的单词类型**以及**如何识别单词**，C 语言单词类型可分为标识符、字符串、常数等，不同的词法分析器可能将其更加细化，有限自动机常常利用状态转移识别单词。

4.2 识别类型

实验中编写的 C 语言词法分析器能够识别的单词类型有七种，分别为：**标识符、常数、操作符、字符串、保留字、边界符、预编译语句**。

（1）标识符

C 语言标识符定义为以字母或下划线开头的由字母、下划线以及数字构成的字符串，这里的标识符不包含保留字。例如：

_abc, abc, abc123, a123bc, _123

（2）保留字

C 语言的保留字共有 32 种，分别为：

auto, break, case, char, const, continue, default, do
double, else, extern, enum, float, for, goto, if
int, long, return, register, static, short, signed, unsigned
struct, switch, sizeof, typedef, union, volatile, void, while

（3）常数

词法分析器可以识别的常数包含整数与浮点数。

其中整数允许前导零，如 **123,0123**。

浮点数通用格式为科学记数法，根据是否有小数部分、是否有尾数部分、尾数部分正负性，如:

1.2, 1.2E3, 1.2e+3,1.2e-3,1e-4。

（4）操作符

C 语言中，操作符是指具有一定运算能力或语句处理能力的符号，词法分析器可以识别的操作符有多种，如下所示：

<, < <, <=, < <=, >, > >, >=, > >=
&, &&, &= |, ||, |=, ^ , ^ =, !
+, ++, +=, -, -=, –, *, *=, /, /=
%, %=, =, ==, ?, , , . , :

(5) 边界符

边界符用来分隔不同表达式或代码段的符号，词法分析器可以识别的边界符如下所示：

{, }, (,), [,], ;

(6) 字符串

C 语言中字符类型有单字符以及字符串，这里统一用“字符串”命名输出结果，词法分析器不仅能够通过单引号、双引号识别出字符串，并且应当避免转义字符的干扰，例如下面几种情况都是属于字符串：

“hello”，“hel\”lo”，‘a’，‘\’’。

(7) 预编译语句

在 C 语言中，由 # 开头的语句称为预编译语句，通常分为头文件、宏定义一些特殊的预编译指令，词法分析器统一用“预编译语句”作为输出命名。如 **#include <stdio.h>, #define, #endif** 等。

4.3 特殊处理

4.3.1 忽略注释

词法分析器能够忽略源程序中的注释，包括单行注释和多行注释，通过查阅资料，大部分编译器不支持注释嵌套，因此该分析器同样不支持注释嵌套，对于嵌套形式的注释，会匹配至第一个注释结束符，下一个注释结束符被解析为单个”*” 和单个”/”。

4.3.2 忽略无意义空格

词法分析器能够忽略源程序中无意义的空格，无意义指删除后对程序语义没有影响，除了字符串的空格被保留以外，其他空格均被忽略，但是作为单词分隔符的空格（以及换行符）在词法分析的过程中会至多保留一个以区分不同单词，不会显式出现在输出结果中。

4.3.3 数据统计

词法分析器能够进行一定的数据统计，如统计字符总数、各类单词总数、程序行数，以及分析过程中记录位置。实验中的词法分析器通过对最后生成的符号表分析以及注释语句的特殊处理，能够获得这些统计信息，位置信息则通过在分析过程中维护两个变量分别保存行列信息来获得。

4.3.4 异常处理

词法分析器如果遇到不满足词法规则的字符，应当能够从错误中恢复并继续进行分析，还应指出错误的位置与类型。实验中的词法分析器遇到异常情况会自动从下一个字符开始新的识别过程，其能够识别的异常类型共有四种：

a. 未知符号，例如“@”等；

b. 指数后非法符号，例如 **1E30** 写为 **1Ea30**；

c. 小数点后非法符号，例如 **1.a**；

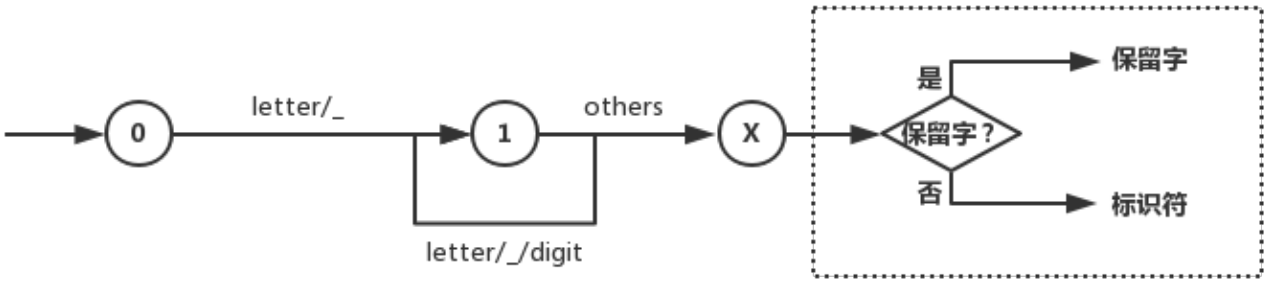
d. 尾数中非法符号，例如 **1E+30** 写为 **1E+a30**；

4.4 确定有限自动机（DFA）

通过自动机的状态转移，我们能够清晰有力地解决识别单词以及流程处理问题，下面针对不同的单词识别任务给出特定的确定有限自动机, 其中状态 X 表示识别结束，虚线框表示后处理过程。

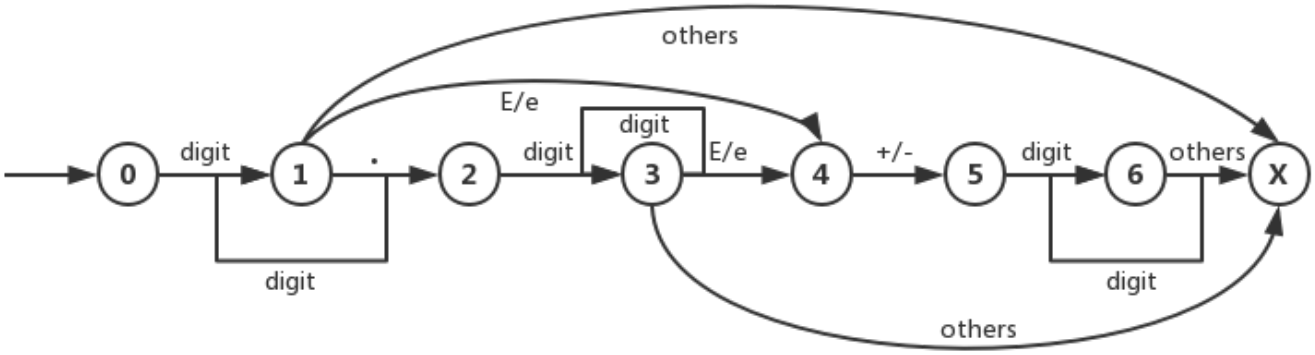
（1）标识符与保留字

C 语言标识符可由下列状态转移图识别，在后处理过程中与 32 个保留字做匹配，如果是保留字则被识别为保留字，否则识别为标识符，如图：



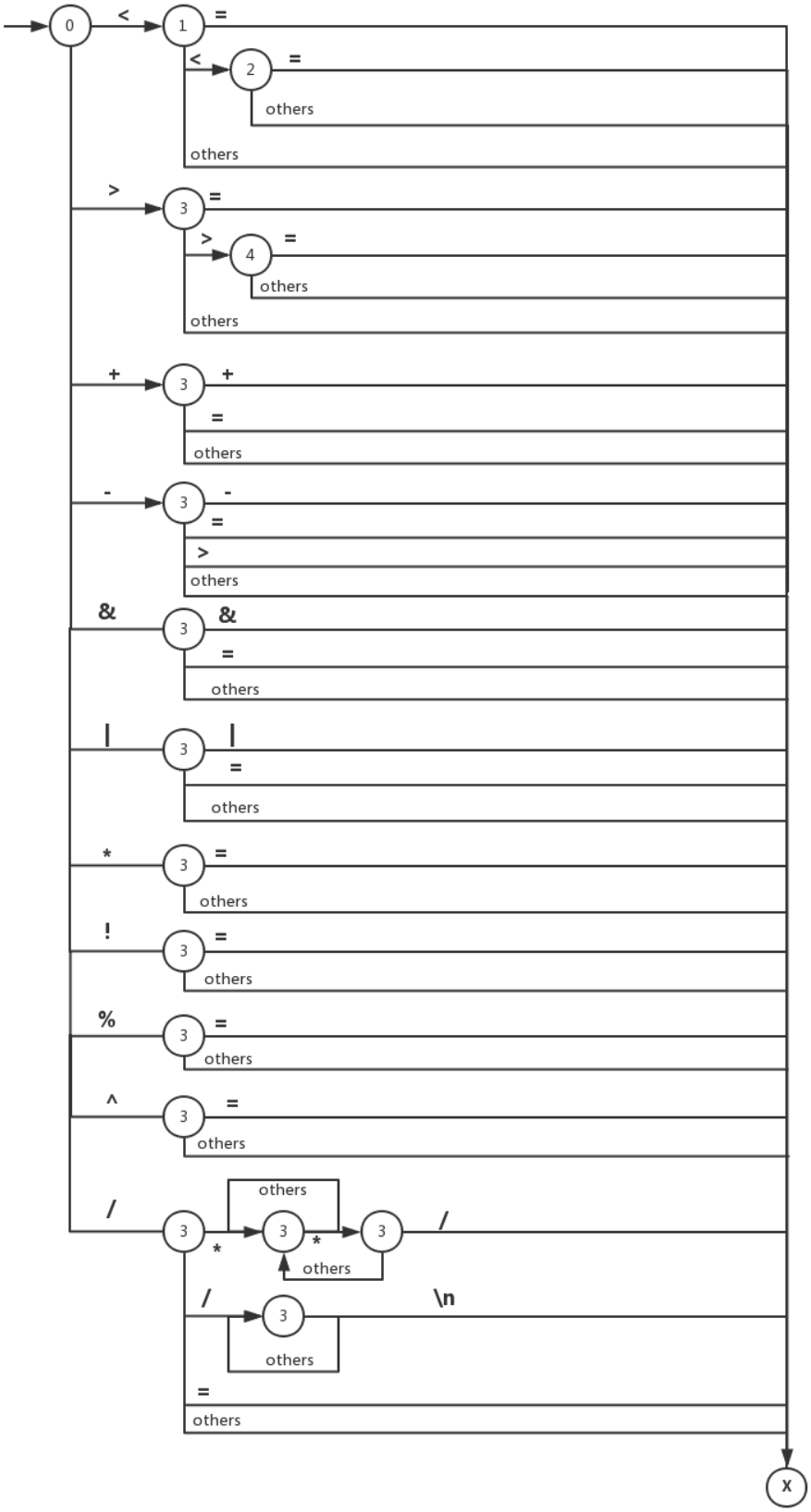
（2）常数

识别常数按照科学记数法进行识别，同时根据合理的情况结束识别或不合理的情况抛出异常，如图：



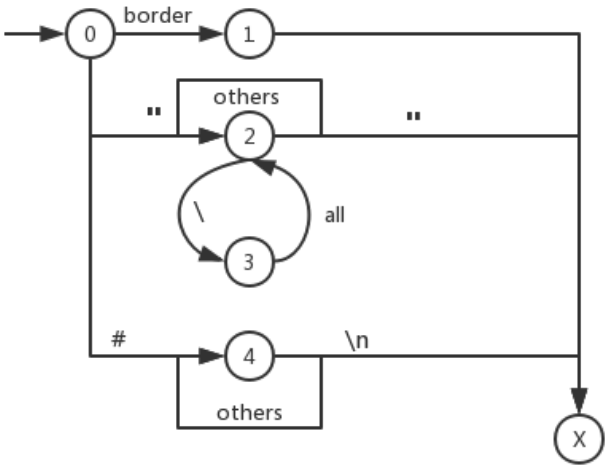
（3）操作符与注释

操作符种类众多，经过整理，其与注释的状态转移如图所示：



（4）边界符、字符串、预编译语句

边界符、字符串与预编译语句识别较为简单，其自动机如下图所示，其中字符（单引号）处理等同字符串（双引号），不再赘述。

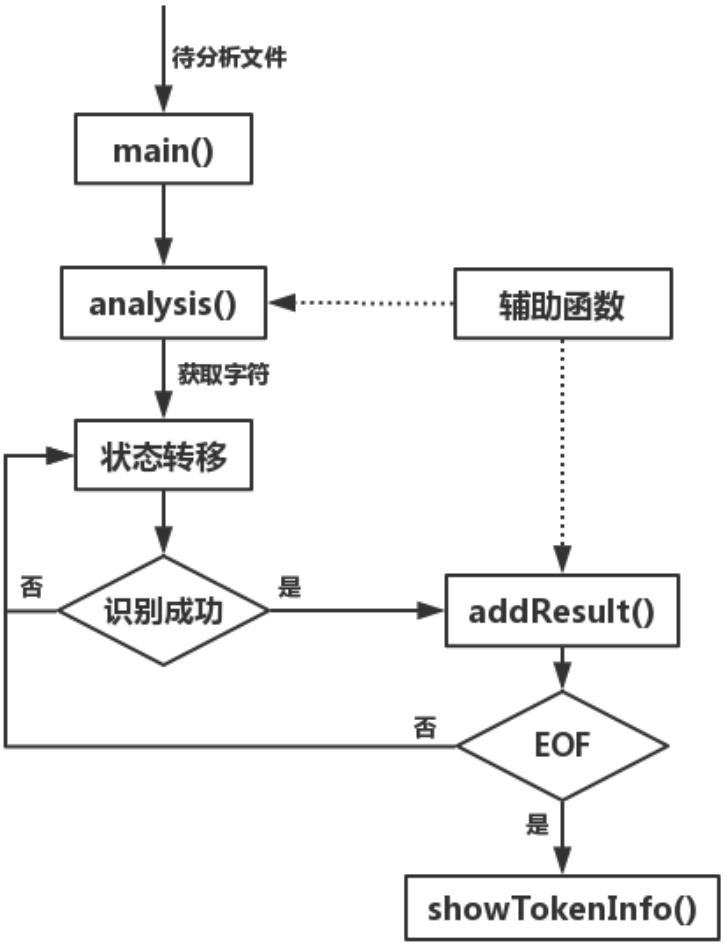


五、 代码分析

5.1 函数功能

函数	描述
main ()	主函数，读入文件送入分析函数
initialize()	初始化符号表
analysis()	根据当前字符决定状态转移
id_table_insert()	将标识符插入记录表，并返回序号
isKey()	判断是否保留字
isBorder()	判断是否边界符
isOperator()	判断是否操作符
getCharacterType()	获取字符类型
nextChar()	获取下一个字符
addResult()	保存识别结果，异常直接输出
showTokenInfo()	格式化输出结果

5.2 总体流程



5.3 自动机状态转移

词法分析器的自动机转移主要由函数”analysis()”，完成。该函数返回 RESULT* 类型的变量，其结构体定义如下：

```
typedef struct RESULT {
  vector<string> tokens; //符号表
  int line;              //行数
  int char_num;          //字符数
}RESULT;
```

函数内部首先初始化当前字符（cur_c），当前状态（state），返回结果以及行列信息。然后读入字符开始进行状态转移，状态转移通过 switch-case 语句完成，图例参见上文，具体代码可参见附录, 大致结构如下：

```
RESULT* analysis(FILE *fp, RESULT* output) {
  char cur_c = ' ';
```

```
  int state = 0;
  string token = "";
  initialize(output);
  int col = 0;
  int line = 1;
  while ((cur_c = nextChar(fp)) && !feof(fp)) {
    int c_type = getCharacterType(cur_c);
    token += cur_c;
    col += 1;
    switch (state) {
      case X: 处理不同状态
      case XX:....
    }
  }
  return output;
}
```

值得注意的是两个宏定义，BACK 与 ZERO，其定义如下：

```
#define BACK fseek(fp,-1L,1);token.pop_back();col--;
#define ZERO state = 0
```

当某单词识别完成后，此时当前字符并不属于已识别的单词，因此需要将当前字符退回字符缓冲区，即使用 BACK，在添加结果完成后，状态重新清零，使用 ZERO，重新开始识别。

5.4 主要辅助函数分析

（1）nextChar(FILE *fp, int* col)

该函数功能为读取下一个字符，连续的空白字符视为一个，返回给状态转移函数进行分析。

```
int nextChar(FILE *fp, int* col) {
  char c = fgetc(fp);
  int i = 0;
  while (c == ' ' || c == '\t') {
    i++;
    col++;
    c = fgetc(fp);
  }
  if (i > 0) {c = ' '; fseek(fp,-1L,1);col--}
  return c;
}
```

（2）addResult(RESULT* output, string token, char cur_c, int output_type, int c_type, int

line, int col)

该函数能够将识别的单词加入符号表，若是异常，则直接输出。

参数中，output 是指要存放的位置，token 是指要存放的字符串，cur_c 是指要存放的字符，output_type 是指输出类型，指明是异常（ERROR）还是单词（TOKEN）；c_type 是指符号类型，可分为操作符、字符串等上文提到的七种符号类型，结构如下，具体代码参见附录。

```
int addResult(RESULT* output, string token, char cur_c, int output_type, int c_type, int line,
    int col) {
  char cons[1024]; col--;
  if (output_type == ERROR) {
    if (cur_c == ' ')
      sprintf(cons, "[Error!] Line %d, Col %d, %s\n",line,col,token.c_str());
    else
      sprintf(cons, "[Error!] Line %d, Col %d, %s,[%c]\n",line,col,token.c_str(),cur_c);
    output->line = line;
    output->char_num += token.size();
    printf("%s",cons);
  }
  else {
    if (c_type == Border) { //省略
    }
    else if (c_type == Operator) {
      if (cur_c == ' ')
        //省略
      else //省略;
    }
    else if (c_type == ID) {
      if (!isKey(token)) { //省略
      }
      else { //省略
      }
    }
    else if (c_type == Digit) { //省略
    }
    else if (c_type == String) { //省略
    }
    else if (c_type == Macro) { //省略
    }
    output->line = line;
    output->char_num += token.size();
    output->tokens.push\_back(cons);
  }
  return 0;
}
```

（3）showTokenInfo(RESULT *output, int line, int char_num)

该函数能够将符号表以及源程序数据信息以格式化的方式进行输出，首先根据符号表获取各个类型的单词个数，然后将行数、单词个数、字符总数输出，最后将符号表格式化输出。

```
int showTokenInfo(vector<string> tokens, int line, int char_num) {
  /*
  字符串：0 头文件： 1 操作符： 2
  标识符：3 保留字： 4 常数： 5
  边界符：6 宏定义： 7
  */
  int info_num[7] = {0};
  string info_type[7] = {"字符串","预编译语句","操作符","标识符","保留字","常数","边界符"};
  for (int i = 0;i < tokens.size(); i++) {
    for (int j = 0;j<7;j++) {
      if (tokens[i].find(info_type[j]) != string::npos) {
        info_num[j]++;
        break;
      }
    }
  }
  printf("\n-----------程序信息统计--------------\n");
  printf("行数：%d，字符总数(不包含空白字符)：%d\n各类单词个数如下：\n",line,char_num);
  int j = 0;
  for (int i = 0; i < 7;i++) {
    if (info_num[i]) {
      j++;
      char temp[1024];
      sprintf(temp, "%s：%d", info_type[i].c_str(),info_num[i]);
      printf("%-20s",temp);
      if (j % 3 == 0) printf("\n");
    }
  }
  printf("\n\n-----------记号表--------------------\n");
  for (int i = 0;i < tokens.size();i++) {
    tokens[i].pop_back();
    printf("%-50s",tokens[i].c_str());
    if ((i+1) % 3 == 0) printf("\n");
  }
}
```

六、 代码测试与结果

6.1 测试用例

为测试代码正确性，我根据代码功能人为构造测试案例，注释处表明该处测试的功能，测试用例如下:

```
/*预编译语句测试*/
#include <stdio.h>
#define True 1

//注释测试
/*Hello,
    this is a comment*/

// 保留字: int, 标识符main 测试
int main(void)
{
   // 异常处理测试
   int a = 1.x;
   int b = 1.2Ea;
   int c = 1.2E+p;
   int d = @;

   // 字符串测试
   char string[256] = "Hello,World!";
   char ss[256] = "Hello,\"wolrd!";
   char s = 'hi';
   char sss = '\'';

   // 常数测试
   1;
   1.2;
   1.2e4;
   1.2e-4;

   // 操作符,边界符测试
   {1<=0,1<<=2,a++,a->p,a&&p};
   [1!=2,1^2,1/=0,*p];

   return 0;
}
```

6.2 测试结果

（1）异常信息与数据统计

```
词法分析中......
[Error!] Line 13, Col 11, 小数点后非法字符,[x]
[Error!] Line 14, Col 13, 指数后非法符号,[a]
[Error!] Line 15, Col 14, 尾数后非法符号,[p]
[Error!] Line 16, Col 10, 未知符号,[@]

----------程序信息统计----------
行数: 35, 非注释字符总数(不包含空白字符): 283, 注释字符总数(不包含空包字符): 136
各类单词个数如下:
字符串: 4          预编译语句: 2          操作符: 24
标识符: 18         保留字: 11             常数: 17
边界符: 27
```

（2）符号表

```
----------记号表----------
Line 2: < 预编译语句: #include <stdio.h> >
Line 3: < 预编译语句: #define True 1 >
Line 10: < 保留字: int >
Line 10: < 标识符: main , 1 >
Line 10: < 边界符: ( >
Line 10: < 保留字: void >
Line 10: < 边界符: ) >
Line 11: < 边界符: { >
Line 13: < 保留字: int >
Line 13: < 标识符: a , 2 >
Line 13: < 操作符: = >
Line 13: < 标识符: x , 3 >
Line 13: < 边界符: ; >
Line 14: < 保留字: int >
Line 14: < 标识符: b , 4 >
Line 14: < 操作符: = >
Line 14: < 标识符: a , 2 >
Line 14: < 边界符: ; >
Line 15: < 保留字: int >
Line 15: < 标识符: c , 5 >
Line 15: < 操作符: = >
Line 15: < 标识符: p , 6 >
Line 15: < 边界符: ; >
Line 16: < 保留字: int >
Line 16: < 标识符: d , 7 >
Line 16: < 操作符: = >
Line 16: < 边界符: ; >
Line 19: < 保留字: char >
Line 19: < 标识符: _string , 8 >
Line 19: < 边界符: [ >
Line 19: < 常数: 256 >
Line 19: < 边界符: ] >
Line 19: < 操作符: = >
Line 19: < 字符串: "Hello,World!" >
Line 19: < 边界符: ; >
Line 20: < 保留字: char >
Line 20: < 标识符: ss , 9 >
Line 20: < 边界符: [ >
Line 20: < 常数: 256 >
Line 20: < 边界符: ] >
Line 20: < 操作符: = >
Line 20: < 字符串: "Hello,\"wolrd!" >
Line 20: < 边界符: ; >
Line 21: < 保留字: char >
Line 21: < 标识符: s , 10 >
Line 21: < 操作符: = >
Line 21: < 字符串: 'hi' >
Line 21: < 边界符: ; >
Line 22: < 保留字: char >
Line 22: < 标识符: sss , 11 >
Line 22: < 操作符: = >
Line 22: < 字符串: '\'' >
Line 22: < 边界符: ; >
Line 25: < 常数: 1 >
Line 25: < 边界符: ; >
Line 26: < 常数: 1.2 >
Line 26: < 边界符: ; >
Line 27: < 常数: 1.2e4 >
Line 27: < 边界符: ; >
Line 28: < 常数: 1.2e-4 >
Line 28: < 边界符: ; >
Line 31: < 边界符: { >
Line 31: < 常数: 1 >
Line 31: < 操作符: <= >
Line 31: < 常数: 0 >
Line 31: < 操作符: , >
Line 31: < 常数: 1 >
Line 31: < 操作符: << >
Line 31: < 常数: 2 >
Line 31: < 操作符: , >
Line 31: < 标识符: a , 2 >
Line 31: < 操作符: ++ >
Line 31: < 操作符: , >
Line 31: < 标识符: a , 2 >
Line 31: < 操作符: -> >
Line 31: < 标识符: p , 6 >
Line 31: < 操作符: , >
Line 31: < 标识符: a , 2 >
Line 31: < 操作符: && >
Line 31: < 标识符: p , 6 >
Line 31: < 边界符: } >
Line 31: < 边界符: ; >
Line 32: < 边界符: [ >
Line 32: < 常数: 1 >
Line 32: < 操作符: ! >
Line 32: < 常数: 2 >
Line 32: < 操作符: , >
Line 32: < 常数: 1 >
Line 32: < 操作符: ^ >
Line 32: < 常数: 2 >
Line 32: < 操作符: , >
Line 32: < 常数: 1 >
Line 32: < 操作符: /= >
Line 32: < 常数: 0 >
Line 32: < 操作符: , >
Line 32: < 操作符: * >
Line 32: < 标识符: p , 6 >
Line 32: < 边界符: ] >
Line 32: < 边界符: ; >
Line 34: < 保留字: return >
Line 34: < 常数: 0 >
Line 34: < 边界符: ; >
Line 35: < 边界符: } >
```

可以看出，词法分析器能够正常运行，并且对词进行了准确的分析。

七、 实验总结

通过本次实验，我巩固了对编译器中词法分析的理解，并学会了利用自动机转移模型高效地解决词法分析中识别单词的任务。通过 C 语言实现词法分析的功能，让编译原理这门课由理论转变为实践，更有助于课程的学习，在实验中，培养了独立发现问题、思考问题、解决问题的能力，更深刻的掌握了实验原理，除此之外，也锻炼了 C/C++ 编程的操作。

附录 A　main.cpp 主程序

```
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <vector>
#include "utils.h"
#include "analyzer.h"
#include <iostream>
using namespace std;

int main(void)
{
   string filename = "program.txt";
   RESULT* output = (RESULT *)malloc(sizeof(RESULT)); // contain result&error
   FILE *fp = fopen(filename.c_str(),"r");

   printf("词法分析中.....\n");
   analysis(fp,output);
   showTokenInfo(output->tokens,output->line,output->char_num,output->comment);
   getchar();
   return 0;
}
```

附录 B　analyzer.h 状态转移

```
#define BACK fseek(fp,-1L,1);token.pop_back();col--;
#define ZERO state = 0


int initialize(RESULT *output) {
   output->char_num = 0;
   output->line = 1;
   output->comment = 0;
}

RESULT* analysis(FILE *fp, RESULT* output) {
   char cur_c = ' ';
   int state = 0;
   string token = "";
   initialize(output);
   int col = 0;
   int line = 1;
```

```
while ((cur_c = nextChar(fp,&col)) && !feof(fp)) {
   int c_type = getCharacterType(cur_c);
   token += cur_c;
   col += 1;
   switch (state) {
      case 0: {
         token = "";
         token += cur_c;
         switch (c_type) {
            case Alphabet: state=1; break;
            case Underline: state = 1; break;
            case Digit: state = 2; break;
            case Border: addResult(output,"",cur_c,TOKENS,Border,line,col); break;
            case Operator: {
               switch (cur_c) {
                  case '<': state = 8; break;
                  case '>': state = 10; break;
                  case '&': state = 12; break;
                  case '+': state = 13; break;
                  case '-': state = 14; break;
                  case '*': state = 16; break;
                  case '/': state = 17; break;
                  case '^': state = 21; break;
                  case '!': state = 22; break;
                  case '=': state = 23; break;
                  case '|': state = 24; break;
                  case '%': state = 25; break;
                  case ',':
                  case '.':
                  case '?':
                  case ':':state = 0; addResult(output,"",cur_c,TOKENS,Operator,line,col);
                      break;
               }
               break;
            }
            case Others: {
               switch (cur_c) {
                  case '#': state = 15; break;
                  case '\'': state = 26; break;
                  case '\"': state = 27; break;
                  case '\n': line++;col = 1; break;
                  case ' ':
                  case '\t': break;
                  default: {
                     addResult(output,"未知符号",cur_c,ERROR,0,line,col);
                     break;
```

```
            }
          }
          break;
        }
      }
      break;
    }
    case 1: {
      switch (c_type) {
        case Alphabet:
        case Underline:
        case Digit: state = 1; break;
        default: {
          BACK;
          ZERO; addResult(output,token,' ',TOKENS,ID,line,col);
          break;
        }
      }
      break;
    }
    case 2: {
      switch (c_type) {
        case Digit: state = 2; break;
        default: {
          if (cur_c == 'E' || cur_c == 'e')
            state = 5;
          else if (cur_c == '.')
            state = 3;
          else {
            BACK;
            ZERO; addResult(output,token,' ',TOKENS,Digit,line,col);
          }
          break;
        }
      }
      break;
    }
    case 3: {
      if (c_type != Digit) {
        BACK;
        ZERO; addResult(output,"小数点后非法字符",cur_c,ERROR,' ',line,col);
      }
      else {
        state = 4;
      }
      break;
    }
```

```
        case 4: {
          if (c_type != Digit && cur_c != 'E' && cur_c != 'e') {
            BACK;
            ZERO; addResult(output,token,' ',TOKENS,Digit,line,col);
          }
          else if (c_type == Digit)
            state = 4;
          else
            state = 5;
          break;
        }
        case 5: {
          if (c_type == Digit)
            state = 7;
          else if (cur_c == '+' || cur_c == '-')
            state = 6;
          else {
            BACK;
            ZERO; addResult(output,"指数后非法符号",cur_c,ERROR,' ',line,col);
          }
          break;
        }
        case 6: {
          if (c_type == Digit)
            state = 7;
          else {
            BACK;
            ZERO; addResult(output,"尾数后非法符号",cur_c,ERROR,' ',line,col);
          }
          break;
        }
        case 7: {
          if (c_type == Digit)
            state = 7;
          else {
            BACK;
            ZERO; addResult(output,token,' ',TOKENS,Digit,line,col);
          }
          break;
        }
        case 8: {
          if (cur_c == '=') {
            ZERO;
            addResult(output,token,' ',TOKENS,Operator,line,col);
          }
          else if (cur_c == '<')
            state = 9;
```

```
        else {
          BACK; ZERO;
          addResult(output,token,' ',TOKENS,Operator,line,col);
        }
        break;
      }
      case 10: {
        if (cur_c == '=') {
          ZERO;
          addResult(output,token,' ',TOKENS,Operator,line,col);
        }
        else if (cur_c == '>')
          state = 11;
        else {
          BACK; ZERO;
          addResult(output,token,' ',TOKENS,Operator,line,col);
        }
        break;
        break;
      }
      case 9:
      case 11: {
        if (cur_c != '=') BACK;
        ZERO; addResult(output,token,' ',TOKENS,Operator,line,col);
        break;
      }
      case 12: {
        if (cur_c == '&' || cur_c == '=') {
          ZERO;
          addResult(output,token,' ',TOKENS,Operator,line,col);
        }
        else {
          BACK; ZERO;
          addResult(output,token,' ',TOKENS,Operator,line,col);
        }
        break;
      }
      case 24: {
        if (cur_c == '|' || cur_c == '=') {
          ZERO;
          addResult(output,token,' ',TOKENS,Operator,line,col);
        }
        else {
          BACK; ZERO;
          addResult(output,token,' ',TOKENS,Operator,line,col);
        }
        break;
```

```
            break;
        }
        case 13: {
            if (cur_c == '+' || cur_c == '=') {
                ZERO;
                addResult(output,token,' ',TOKENS,Operator,line,col);
            }
            else {
                BACK; ZERO;
                addResult(output,token,' ',TOKENS,Operator,line,col);
            }
            break;
        }
        case 14: {
            if (cur_c == '-' || cur_c == '=' || cur_c == '>') {
                ZERO;
                addResult(output,token,' ',TOKENS,Operator,line,col);
            }
            else {
                BACK; ZERO;
                addResult(output,token,' ',TOKENS,Operator,line,col);
            }
            break;
        }
        case 15: {
            while (cur_c != '\n' && !feof(fp)) {
                cur_c = nextChar(fp,&col);
                token += cur_c;
            }
            BACK; ZERO;
            addResult(output,token,' ',TOKENS,Macro,line,col);
            break;
        }
        case 16:
        case 21:
        case 22:
        case 25:
        case 23: {
            if (cur_c != '=') BACK;
            ZERO;
            addResult(output,token,' ',TOKENS,Operator,line,col);
            break;
        }
        case 17: {
            if (cur_c == '*') state = 18,output->comment += 2;
            else if (cur_c == '/') state = 20,output->comment += 2;
            else if (cur_c == '=') {
```

```
            ZERO;
            addResult(output,token,' ',TOKENS,Operator,line,col);
        }
        else {
            BACK;
            ZERO;
            addResult(output,token,' ',TOKENS,Operator,line,col);
        }
        break;
    }
    case 18: {
        if (cur_c != ' ' && cur_c != '\t' && cur_c != '\n' && cur_c != '\r')
            output->comment++;
        if (cur_c == '\n') line++;
        if (cur_c == '*') state = 19;
        break;
    }
    case 19: {
        if (cur_c != ' ' && cur_c != '\t' && cur_c != '\n' && cur_c != '\r')
            output->comment++;
        if (cur_c == '\n') line++;
        if (cur_c == '/') ZERO;
        else state = 18;
        break;
    }
    case 20: {
        if (cur_c != ' ' && cur_c != '\t' && cur_c != '\n' && cur_c != '\r')
            output->comment++;
        if (cur_c == '\n') {
            BACK;
            ZERO;
        }
        break;
    }
    case 26: {
        if (cur_c == '\\') state = 28;
        else if (cur_c == '\'') {
            ZERO;
            addResult(output,token,' ',TOKENS,String,line,col);
        }
        break;
    }
    case 27: {
        if (cur_c == '\\') state = 29;
        else if (cur_c == '\"') {
            ZERO;
            addResult(output,token,' ',TOKENS,String,line,col);
```

```
            }
            break;
        }
        case 28: {
            state = 26;
            break;
        }
        case 29: {
            state = 27;
            break;
        }
    }
  }
  return output;
}
```

附录 C　utils.h 辅助函数

```
#define CHARACTER 100
#define STRING 101
#define TOKENS 102
#define ERROR 103
#define Alphabet 104
#define Underline 105
#define Border 106
#define Operator 107
#define ID 108
#define Digit 109
#define String 110
#define Macro 111
#define Others 999
using namespace std;

typedef struct RESULT {
  vector<string> tokens;
  int line;
  int char_num;
  int comment;
}RESULT;

char borders[7] = {'{','}','(',')','[',']',';'};
char operators[20] = {'<','>','&','+','-','*','/','^','!','=','|','%','.','?',':',',','!'};
string keyword[32]={"auto","break","case","char","const",
                "continue","default","do","double","else","extern",
                "enum","float","for","goto","if","int","long","return",
```

```
                "register","static","short","signed","unsigned",
                "struct","switch","sizeof","typedef","union",
                "volatile","void","while"};

string id_table[256];
int id_num = 0;

int id_table_insert(string word) {
   for (int i = 0;i < id_num;i++) {
      if (id_table[i].compare(word) == 0) {
         return i+1;
      }
   }
   id_table[id_num++] = word;
   return id_num;
}

int isKey(string word) {
    for(int i = 0; i < 32; i++)
       if (keyword[i].compare(word)==0)
          return 1;
    return 0;
}

int isBorder(char c) {
   for (int i = 0; i < 7; i++)
      if (c == borders[i])
         return 1;
   return 0;
}

int isOperator(char c) {
   for (int i = 0; i < 17; i++)
      if (c == operators[i])
         return 1;
   return 0;
}

int getCharacterType(char c) {
   if (c >= '0' && c <= '9')
      return Digit;
   else if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
      return Alphabet;
   else if (c == '_')
      return Underline;
   else if (isBorder(c))
      return Border;
```

```
  else if (isOperator(c))
    return Operator;
  else
    return Others;
}

int nextChar(FILE *fp, int* col) {
  char c = fgetc(fp);
  int i = 0;
  while (c == ' ' || c == '\t') {
    i++;
    col++;
    c = fgetc(fp);
  }
  if (i > 0) {c = ' '; fseek(fp,-1L,1);col--;}
  return c;
}

int addResult(RESULT* output, string token, char cur_c, int output_type, int c_type, int line,
    int col) {
  char cons[1024]; col--;
  if (output_type == ERROR) {
    if (cur_c == ' ')
      sprintf(cons, "[Error!] Line %d, Col %d, %s\n",line,col,token.c_str());
    else
      sprintf(cons, "[Error!] Line %d, Col %d, %s,[%c]\n",line,col,token.c_str(),cur_c);
    output->line = line;
    output->char_num += token.size();
    printf("%s",cons);
  }
  else {
    if (c_type == Border) {
      sprintf(cons, "Line %d: < 边界符: %c >\n",line,cur_c);
      output->char_num += 1;
    }
    else if (c_type == Operator) {
      if (cur_c == ' ')
        sprintf(cons, "Line %d: < 操作符: %s >\n",line,token.c_str());
      else {
        sprintf(cons, "Line %d: < 操作符: %c >\n",line,cur_c);
        output->char_num += 1;
      }
    }
    else if (c_type == ID) {
      if (!isKey(token)) {
        int id = id_table_insert(token);
        sprintf(cons, "Line %d: < 标识符: %s , %d >\n",line,token.c_str(),id);
```

```
        }
        else {
            sprintf(cons, "Line %d: < 保留字: %s >\n",line,token.c_str());
        }
    }
    else if (c_type == Digit) {
        sprintf(cons, "Line %d: < 常数: %s >\n",line,token.c_str());
    }
    else if (c_type == String) {
        sprintf(cons, "Line %d: < 字符串: %s >\n",line,token.c_str());
    }
    else if (c_type == Macro) {
        sprintf(cons, "Line %d: < 预编译语句: %s >\n",line,token.c_str());
    }
    //printf("%s",cons);
    output->line = line;
    output->char_num += token.size();
    output->tokens.push_back(cons);
  }

  return 0;
}

int showTokenInfo(vector<string> tokens, int line, int char_num,int comment) {
  /*
  字符串: 0 头文件:  1 操作符:  2
  标识符: 3 保留字:  4 常数:  5
  边界符: 6 宏定义:  7
  */
  int info_num[7] = {0};
  string info_type[7] = {"字符串","预编译语句","操作符","标识符","保留字","常数","边界符"};
  for (int i = 0;i < tokens.size(); i++) {
    for (int j = 0;j<7;j++) {
      if (tokens[i].find(info_type[j]) != string::npos) {
        info_num[j]++;
        break;
      }
    }
  }
  printf("\n-----------程序信息统计--------------\n");
  printf("行数: %d, 非注释字符总数(不包含空白字符): %d,
      注释字符总数(不包含空包字符): %d\n各类单词个数如下: \n",line,char_num,comment);
  int j = 0;
  for (int i = 0; i < 7;i++) {
    if (info_num[i]) {
      j++;
      char temp[1024];
```

```
        sprintf(temp, "%s: %d", info_type[i].c_str(),info_num[i]);
        printf("%-20s",temp);
        if (j % 3 == 0) printf("\n");
      }
    }
    printf("\n\n-----------记号表--------------------\n");
    for (int i = 0;i < tokens.size();i++) {
      tokens[i].pop_back();
      printf("%-50s",tokens[i].c_str());
      if ((i+1) % 3 == 0) printf("\n");
    }
}
```