**C语言词法分析程序 实验报告**

#### 一、实验题目及要求

**1、C语言词法分析程序的设计与实现**

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。

**2、实现方式**

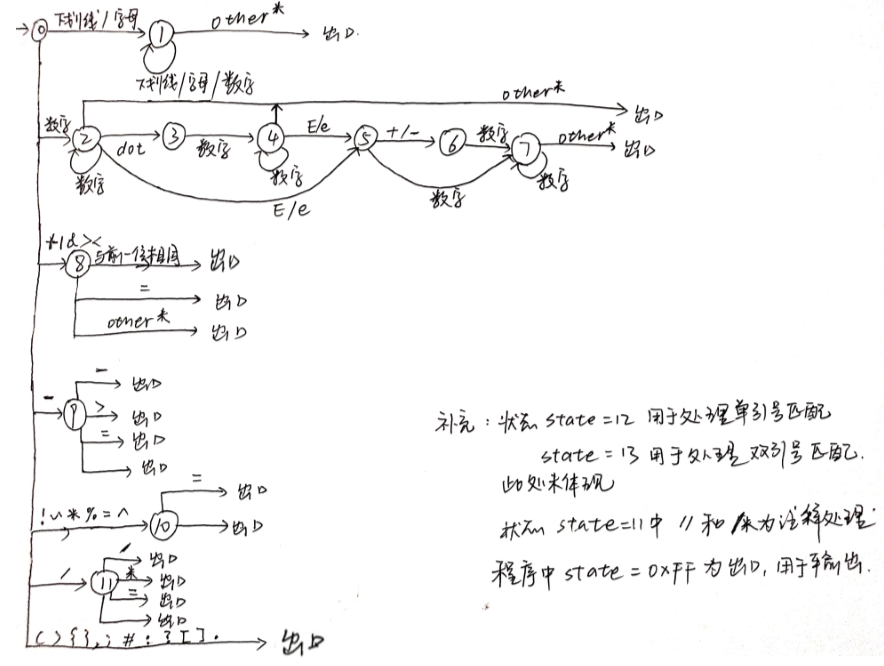
方法1：采用C/C++作为实现语言，手工编写词法分析程序。（必做）

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。

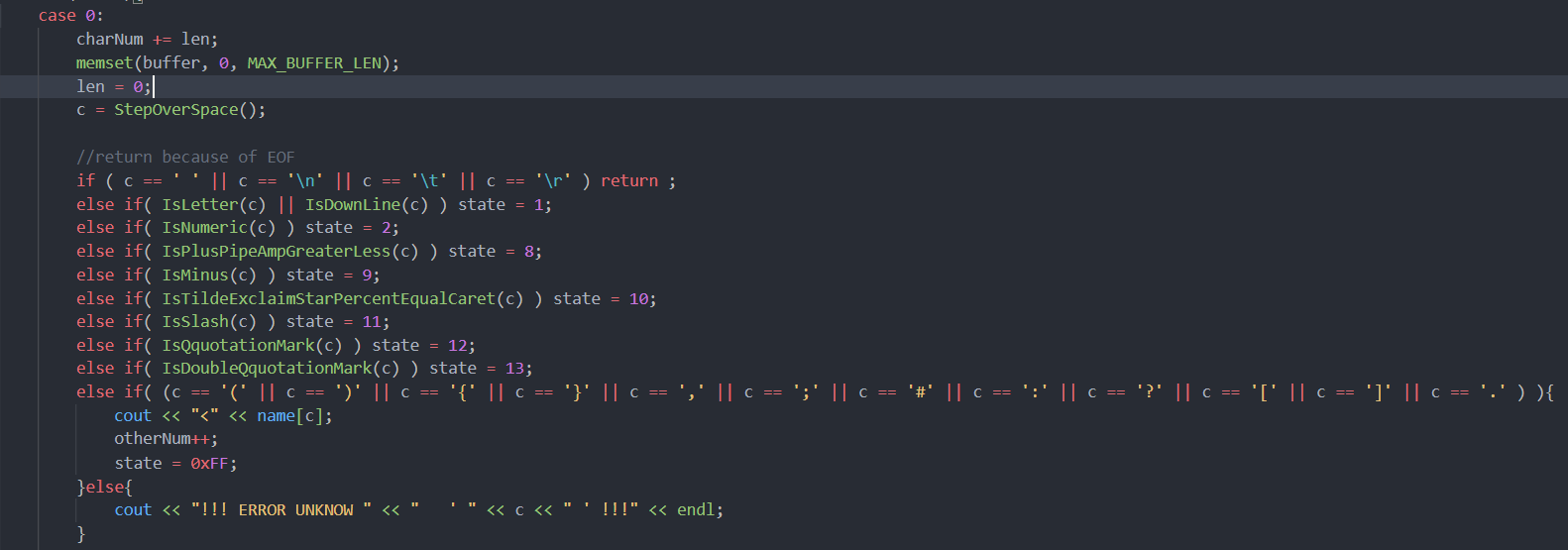
#### 二、C++分析程序设计说明

**1、程序执行状态转移设计：**

程序逐字符读入要进行词法分析的源码（由C语言编写），之后根据读入字符的ASCII码判断字符类型，然后根据类型设置不同的state值，依据此值进入case执行代码。状态转移图如下：



状态0对应下图的开始状态，根据输入的字符类型进行状态分发。



**2、数据结构设计：**

使用C++设计LexAnalysis类，包含关键类成员变量和类成员函数。

**文本

描述已自动生成**

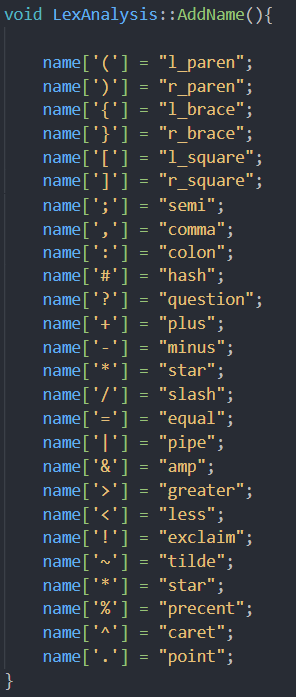
其中state为状态变量，Num结尾的变量为计数相关变量，cfile则为打开文件后的文件流。

全局变量：

文本

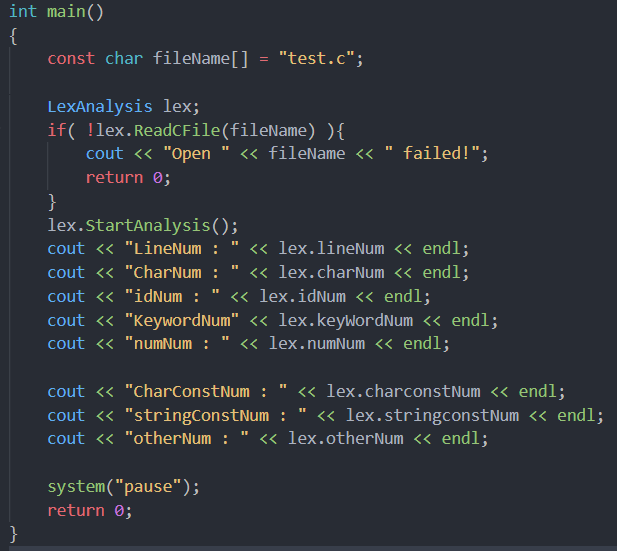
描述已自动生成

其中table结尾的为符号表，意欲与之后实验进行衔接，Keyword则为C语言关键字表。map类型变量name则为符号名称映射。



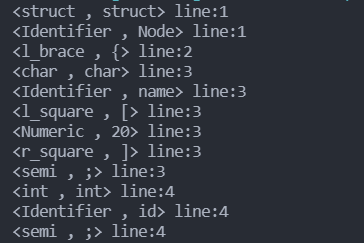
**3、主函数调用设计：**

主函数调用ReadCFile打开文件后执行StarAnalysis()开始词法分析。



**5、输出设计：**

以二元组<id,属性>的方式进行输出，后面再加上行号。

****

**6、其他函数设计：**

**文本

描述已自动生成**

Is开头函数均为判断字符类型函数，LexAnalysis则为构造函数、AddName则为名称映射函数、ReadCFile为打开文件函数、StartAnalysis则为主分析函数、StepOverSpace则为处理空格换行等符号的函数以及LexComment为注释处理函数。

**7、错误识别设计：**

词法分析程序可以识别部分词法分析错误，此处设计注释/\*\*/匹配报错并继续分析的处理、单双引号匹配报错并进行分析的处理以及不能识别的字符报错的处理。

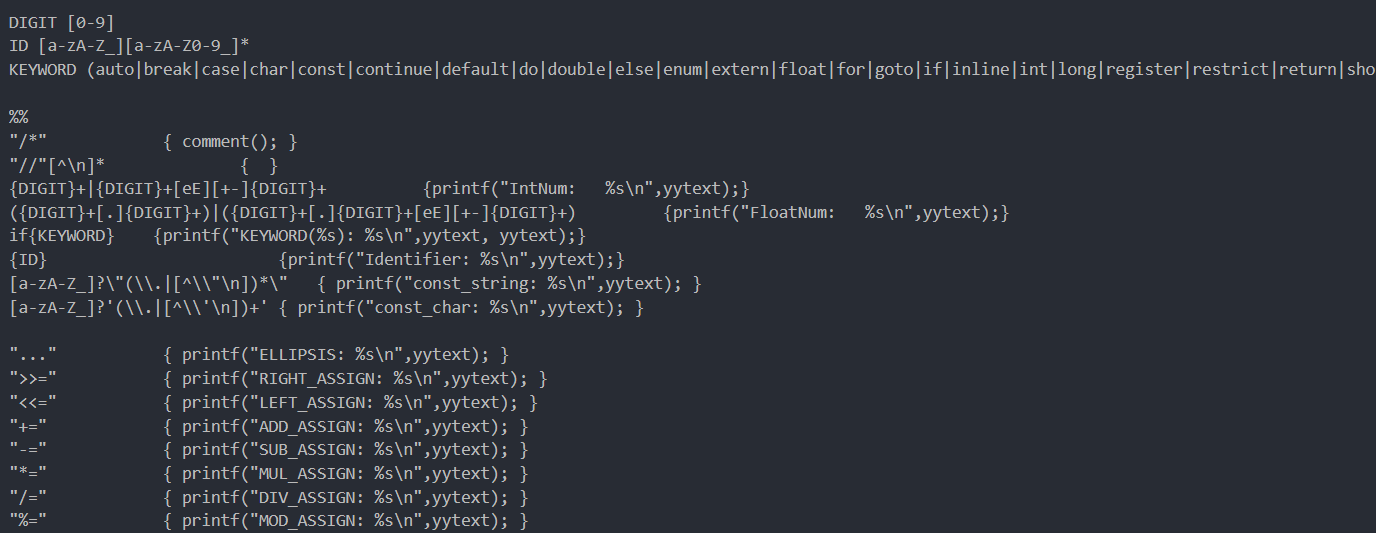
错误示范：

文本

描述已自动生成

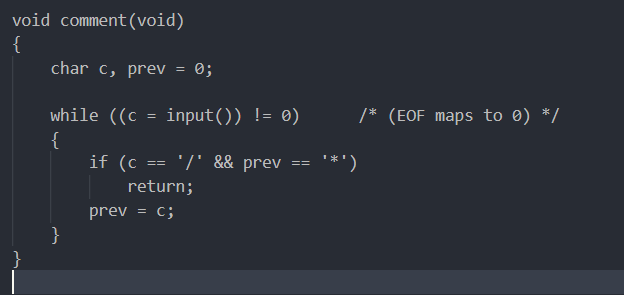
#### 三、Lex分析程序设计说明

在Linux下安装flex，编写lex.l源代码，如下图（截取部分代码）。

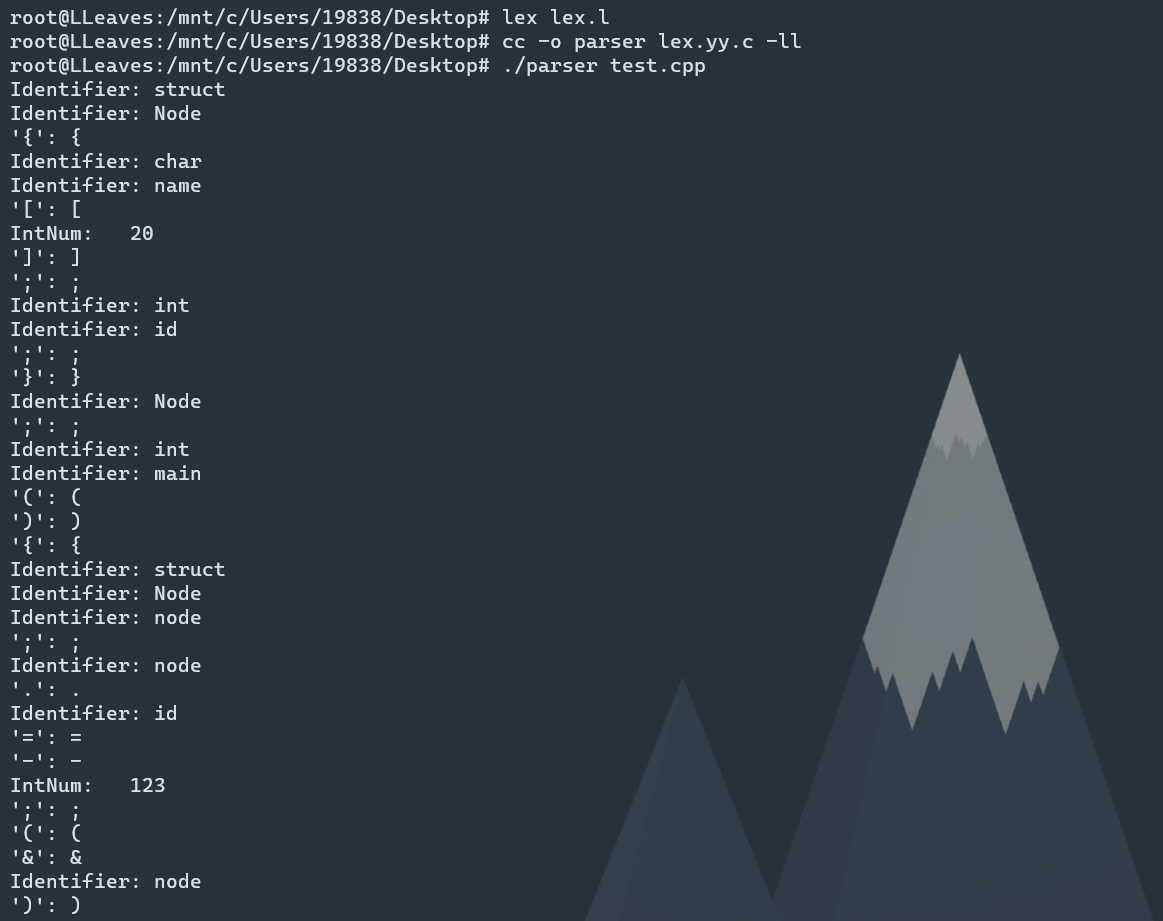


可以看出Lex源代码可以根据正则进行匹配，操作简单，无需状态转移。

需要编写的lex.l源码需要分为三部分。分别为定义部分、规则部分和用户子程序部分。最重要的是规则部分，其内容为词法分析规则。本次编程需要注释处理程序，需要在第一部分定义，第三部分声明（如下图所示）。



Lex源代码的编译以及使用



#### 四、源代码

**C语言源码：**

|  |
| --- |
| #include <cstdio>  #include <cstdlib>  #include <iostream>  #include <cstring>  #include <fstream>  #include <string>  #include <map>  #include <iomanip>  #define MAX\_ID\_LEN 64  #define MAX\_KEYWORD\_LEN 31  #define MAX\_SYMBOL\_TABLE\_LEN 512  #define MAX\_NUMERIC\_TABLE\_LEN 512  #define KEYWORD\_TABLE\_LEN 34  #define MAX\_BUFFER\_LEN 256  char \*symbol\_table[MAX\_SYMBOL\_TABLE\_LEN];  char \*numeric\_table[MAX\_NUMERIC\_TABLE\_LEN];  const char keyword[KEYWORD\_TABLE\_LEN][MAX\_KEYWORD\_LEN] = {      "auto", "break", "case", "char", "const", "continue",      "default", "do", "double", "else", "enum", "extern",      "float", "for", "goto", "if", "inline", "int",      "long", "register", "restrict", "return", "short", "signed",      "sizeof", "static", "struct", "switch", "typedef", "union",      "unsigned", "void", "volatile", "while"};  using namespace std;  map<int, string> name;  map<string, int> num;  class LexAnalysis{  public:      int state;      int lineNum;      int charNum;      int idNum;      int numNum;      int keyWordNum;      int otherNum;      int charconstNum;      int stringconstNum;      ifstream cfile;      char identifier[MAX\_ID\_LEN];      LexAnalysis(){          state = 0;          AddName();      }      void AddName();      bool ReadCFile(const char \**fileName*);      void StartAnalysis();      char StepOverSpace();      bool IsLetter(const char *c*);      bool IsDownLine(const char *c*);      bool IsNumeric(const char *c*);      bool IsPlusPipeAmpGreaterLess(const char *c*);      bool IsMinus(const char *c*);      bool IsTildeExclaimStarPercentEqualCaret(const char *c*);      bool IsSlash(const char *c*);      bool IsDoubleQquotationMark(const char *c*);      bool IsQquotationMark(const char *c*);      bool IsKeyword(const char \**word*);      bool LexComment();  };  void LexAnalysis::AddName(){        name['('] = "l\_paren";      name[')'] = "r\_paren";      name['{'] = "l\_brace";      name['}'] = "r\_brace";      name['['] = "l\_square";      name[']'] = "r\_square";      name[';'] = "semi";      name[','] = "comma";      name[':'] = "colon";      name['#'] = "hash";      name['?'] = "question";      name['+'] = "plus";      name['-'] = "minus";      name['\*'] = "star";      name['/'] = "slash";      name['='] = "equal";      name['|'] = "pipe";      name['&'] = "amp";      name['>'] = "greater";      name['<'] = "less";      name['!'] = "exclaim";      name['~'] = "tilde";      name['\*'] = "star";      name['%'] = "precent";      name['^'] = "caret";      name['.'] = "point";  }  bool LexAnalysis::ReadCFile(const char \**fileName*){      bool is\_open = false;      cfile.open(*fileName*, ios::in | ios::binary);      if(cfile.is\_open()){          is\_open = true;      }      return is\_open;  }  bool LexAnalysis::IsLetter(const char *c*){      bool isLetter = false;      if( (*c* <= 'z' && *c* >= 'a') || (*c* <= 'Z' && *c* >= 'A') )          isLetter = true;      return isLetter;  }  bool LexAnalysis::IsDownLine(const char *c*){      bool isDownLine = false;      if( *c* == '\_' )          isDownLine = true;      return isDownLine;  }  bool LexAnalysis::IsPlusPipeAmpGreaterLess(const char *c*)  {      bool isPlusPipeAmp = false;      if (*c* == '+' || *c* == '|' || *c* == '&' || *c* == '<' || *c* == '>')          isPlusPipeAmp = true;      return isPlusPipeAmp;  }  bool LexAnalysis::IsMinus(const char *c*)  {      bool isMinus = false;      if (*c* == '-')          isMinus = true;      return isMinus;  }  char LexAnalysis::StepOverSpace()  {      char c;      cfile >> noskipws  >> c;      while ( cfile.peek() != EOF && (c == ' ' || c == '\n' || c == '\t' || c == '\r')){          if (c == '\n')              lineNum++;          cfile >> c;      }      return c;  }  bool LexAnalysis::IsNumeric(const char *c*)  {      bool isNumeric = false;      if( *c* <= '9' && *c* >= '0')          isNumeric = true;      return isNumeric;  }  bool LexAnalysis::IsKeyword(const char \**word*)  {      bool isKeyword = false;      int len = 0;      int wordLen = strlen(*word*);      for(len = 0;len < KEYWORD\_TABLE\_LEN;len++){          if(strlen(keyword[len]) == wordLen && strncmp(*word*,keyword[len],wordLen) == 0){              isKeyword = true;              break;          }      }      return isKeyword;  }  bool LexAnalysis::IsTildeExclaimStarPercentEqualCaret(const char *c*){      bool isTildeExclaimStarPercentEqualCaret = false;      if (*c* == '!' || *c* == '~' || *c* == '\*' || *c* == '%' || *c* == '=' || *c* == '^')          isTildeExclaimStarPercentEqualCaret = true;      return isTildeExclaimStarPercentEqualCaret;  }  bool LexAnalysis::IsQquotationMark(const char *c*){      bool isQquotationMark = false;      if (*c* == '\''){          isQquotationMark = true;      }      return isQquotationMark;  }  bool LexAnalysis::IsDoubleQquotationMark(const char *c*){      bool isDoubleQquotationMark = false;      if (*c* == '\"')      {          isDoubleQquotationMark = true;      }      return isDoubleQquotationMark;  }  bool LexAnalysis::IsSlash(const char *c*){      bool isSlash = false;      if( *c* == '/' )          isSlash = true;      return isSlash;  }  bool LexAnalysis::LexComment(){      int len = 0;      char c;      cfile >> noskipws >> c;      while (cfile.peek() != EOF ){          while (c != '\*' && cfile.peek() != EOF)          {              cfile >> noskipws >> c;              if (c == '\n')                  lineNum++;              len++;          }          cfile >> noskipws >> c;          len++;          if( c == '/' ){              break;          }          cfile >> noskipws >> c;          len++;      }      if( c != '/'){          cfile.seekg(-len,ios\_base::cur);          return 0;      }      return 1;  }  void LexAnalysis::StartAnalysis(){      char c;      int len = 0;      lineNum = 1;      charNum = 0;      idNum = 0;      numNum = 0;      keyWordNum = 0;      charconstNum = 0;      stringconstNum = 0;      otherNum = 0;      string before, now;      char buffer[MAX\_BUFFER\_LEN] = {'\0'};      while ( 1 ){          switch (state){              case 0:                  charNum += len;                  memset(buffer, 0, MAX\_BUFFER\_LEN);                  len = 0;                  c = StepOverSpace();                  //return because of EOF                  if ( c == ' ' || c == '\n' || c == '\t' || c == '\r' ) return ;                  else if( IsLetter(c) || IsDownLine(c) ) state = 1;                  else if( IsNumeric(c) ) state = 2;                  else if( IsPlusPipeAmpGreaterLess(c) ) state = 8;                  else if( IsMinus(c) ) state = 9;                  else if( IsTildeExclaimStarPercentEqualCaret(c) ) state = 10;                  else if( IsSlash(c) ) state = 11;                  else if( IsQquotationMark(c) ) state = 12;                  else if( IsDoubleQquotationMark(c) ) state = 13;                  else if( (c == '(' || c == ')' || c == '{' || c == '}' || c == ',' || c == ';' || c == '#' || c == ':' || c == '?' || c == '[' || c == ']' || c == '.' ) ){                      cout << "<" << name[c];                      otherNum++;                      state = 0xFF;                  }else{                      cout << "!!! ERROR UNKNOW " << "   ' " << c << " ' !!!" << endl;                  }                  buffer[len++] = c;                  if( cfile.peek() == EOF){                      if( state != 0xFF ){                          cout << "<" << name[c];                          otherNum++;                      }                        cout << " , " << buffer << "> line:" << lineNum << endl;                      charNum += len;                      return;                  }                  break;              case 1:                  cfile >> noskipws >> c;                  if ( (IsLetter(c) || IsDownLine(c) || IsNumeric(c)) && !cfile.eof()){                      state = 1;                      buffer[len++] = c;                  }                  else{                      if( !IsKeyword(buffer) ) {                          cout << "<Identifier";                          idNum++;                      }                      else{                          cout << "<" << buffer;                          keyWordNum++;                      }                      cfile.seekg(-1,ios\_base::cur);                      state = 0xFF;                  }                  break;              case 2:                  cfile >> noskipws >> c;                  if( cfile.peek() != EOF){                      if( IsNumeric(c) )state = 2;                      else if( c == '.' ) state = 3;                      else if( c == 'E' || c == 'e') state = 5;                      else{                          cout << "<Numeric";                          numNum++;                          cfile.seekg(-1, ios\_base::cur);                          state = 0xFF;                      }                  }else{                      cout << "<Numeric";                      buffer[len++] = c;                      numNum++;                      state = 0xFF;                  }                    if(state != 0xFF)                      buffer[len++] = c;                  break;              case 3:                  cfile >> noskipws >> c;                  if( IsNumeric(c) ) state = 4;                  buffer[len++] = c;                  break;              case 4:                  cfile >> noskipws >> c;                  if( IsNumeric(c) ) state = 4;                  if( cfile.peek() == EOF){                      state = 0xFF;                      break;                  }                  else if( (c == 'E' || c == 'e')) state = 5;                  else{                      cfile.seekg(-1, ios\_base::cur);                      cout << "<Numeric";                      numNum++;                      state = 0xff;                  }                  if( state != 0xFF ) buffer[len++] = c;                  break;              case 5:                  cfile >> noskipws >> c;                  if( c == '+' || c == '-' ) state = 6;                  else if( IsNumeric(c) ) state = 7;                  else {                      cout << "!!! ERROR UNKNOW " << "   ' " << buffer << " ' !!!" << endl;                      state = 0;                  }                  if (cfile.peek() == EOF){                      charNum += len;                      return;                  }                  buffer[len++] = c;                  break;              case 6:                  cfile >> noskipws >> c;                  if( IsNumeric(c) ) state = 7;                  buffer[len++] = c;                  break;              case 7:                  cfile >> noskipws >> c;                  if( IsNumeric(c) ){                      state = 7;                      buffer[len++] = c;                  }else{                      state = 0xFF;                      cfile.seekg(-1, ios\_base::cur);                      cout << "<Numeric";                      numNum++;                  }                  break;              case 8:                  before = name[buffer[len - 1]];                  cfile >> noskipws >> c;                  if( buffer[len-1] == '+' && c == '+' ) buffer[len++] = c;                  else if( buffer[len-1] == '|' && c == '|') buffer[len++] = c;                  else if ( buffer[len-1] == '&' && c == '&'  ) buffer[len++] = c;                  else if (buffer[len - 1] == '<' && c == '<') buffer[len++] = c;                  else if (buffer[len - 1] == '>' && c == '>') buffer[len++] = c;                  else if (c == '=') buffer[len++] = c;                  else{                      cfile.seekg(-1, ios\_base::cur);                      cout << "<" << before << " , " << buffer << "> line:" << lineNum << endl;                      otherNum++;                  }                  if(len == 2){                      now = name[c];                      cout << "<" << before << now << " , " << buffer << "> line:" << lineNum << endl;                      otherNum++;                  }                  if (cfile.peek() == EOF){                      charNum += len;                      return;                  }                  state = 0;                  break;              case 9:                  before = "minus";                  cfile >> noskipws >> c;                  if( c == '-'  ){                      buffer[len++] = c;                      now = "minus";                  }                  else if(c == '>' ){                      buffer[len++] = c;                      now = "greater(arrow)";                  }                  else if(c == '='){                      buffer[len++] = c;                      now = "equal";                  }else{                      cfile.seekg(-1, ios\_base::cur);                  }                  if(len == 2){                      cout << "<" << before << now << " , " << buffer << "> line:" << lineNum << endl;                      otherNum++;                  }                  else{                      cout << "<" << before << " , " << buffer << "> line:" << lineNum << endl;                      otherNum++;                  }                    if (cfile.peek() == EOF){                      charNum += len;                      return;                  }                  state = 0;                  break;              case 10:                  before = name[buffer[len-1]];                  cfile >> noskipws >> c;                  if( c == '='  ){                      buffer[len++] = c;                  }else{                      cfile.seekg(-1,ios\_base::cur);                  }                  if( len == 2 ){                      cout << "<" << before << name['='] << " , " << buffer << "> line:" << lineNum << endl;                      otherNum++;                  }else{                      cout << "<" << before << " , " << buffer << "> line:" << lineNum << endl;                      otherNum++;                  }                    if (cfile.peek() == EOF)                      return;                  state = 0;                  break;              case 11:                  before = name[buffer[len - 1]];                  cfile >> noskipws >> c;                  if( c == '/' ){                      lineNum += 1;                      cfile >> noskipws >> c;                      if( cfile.peek() == EOF ) break;                      while ( c!= '\n' && cfile.peek() != EOF){                          cfile >> noskipws >> c;                      }                    }                  else if( c == '\*' ){                      if( !LexComment() ){                          cout << "!!! missing terminating \" \*/ \" character !!!" << endl;                          if (cfile.peek() == EOF)                              return;                      }                  }                  else if(c == '=' ){                      now = name[c];                      buffer[len++] = c;                      cout << "<" << before << name[c] << " , " << buffer << "> line:" << lineNum << endl;                      otherNum++;                  }else{                      cout << "<" << before << " , " << buffer << "> line:" << lineNum  << endl;                      otherNum++;                      cfile.seekg(-1, ios\_base::cur);                  }                  if (cfile.peek() == EOF)                      return;                  state = 0;                  break;              case 12:                  cfile >> noskipws >> c;                  while( c != '\r' && c != '\n' && cfile.peek() != EOF && c != '\'' ){                      buffer[len++] = c;                      cfile >> noskipws >> c;                  }                  if( c == '\'' ){                      buffer[len++] = c;                      cout << "<const\_character" << " , " << buffer << "> line:" << lineNum  << endl;                      charconstNum++;                  }else if( c == '\n' || c == '\r' || cfile.peek() == EOF ){                      cout << "!!! missing terminating \" \' \" character !!!" << endl;                      lineNum++;                      cout << "!!! ERROR UNKNOW " << "   ' " << buffer << " ' !!!" << endl;                  }                  if (cfile.peek() == EOF)                      return;                  state = 0;                  break;              case 13:                  cfile >> noskipws >> c;                  while( c != '\r' && c != '\n' && cfile.peek() != EOF && c != '\"'){                      buffer[len++] = c;                      cfile >> noskipws >> c;                  }                  if( c == '\"'){                      buffer[len++] = c;                      cout << "const\_string" << " , " << buffer << "> line:" << lineNum  << endl;                      stringconstNum++;                      if (cfile.peek() == EOF)                          return;                  }else if( c == '\n' || c == '\r' || cfile.peek() == EOF){                      cout << "missing terminating \" \" \" character" << endl;                      lineNum++;                      cout << "ERROR UNKNOW" << "   ' " << buffer << " ' !!! " << endl;                      if (cfile.peek() == EOF)                          return;                  }                  state = 0;                  break;              case 0xFF:                  cout << " , " << buffer << "> line:" << lineNum << endl;                  state = 0;                  if( cfile.peek() == EOF ){                      charNum += len;                      return ;                  }                  break;          }      }  }  int main()  {      const char fileName[] = "test.c";      LexAnalysis lex;      if( !lex.ReadCFile(fileName) ){          cout << "Open " << fileName << " failed!";          return 0;      }      lex.StartAnalysis();      cout << "LineNum : " << lex.lineNum << endl;      cout << "CharNum : " << lex.charNum << endl;      cout << "idNum : " << lex.idNum << endl;      cout << "KeywordNum" << lex.keyWordNum << endl;      cout << "numNum : " << lex.numNum << endl;      cout << "CharConstNum : " << lex.charconstNum << endl;      cout << "stringConstNum : " << lex.stringconstNum << endl;      cout << "otherNum : " << lex.otherNum << endl;      system("pause");      return 0;  } |

**Lex源代码**

|  |
| --- |
| %{  #include<math.h>  #include<stdlib.h>  #include<stdio.h>  void comment(void);  %}  DIGIT [0-9]  ID [a-zA-Z\_][a-zA-Z0-9\_]\*  KEYWORD (auto|break|case|char|const|continue|default|do|double|else|enum|extern|float|for|goto|if|inline|int|long|register|restrict|return|short|signed|sizeof|static|struct|switch|typedef|union|unsigned|void|volatile|while)  %%  "/\*" { comment(); }  "//"[^\n]\* { }  {DIGIT}+|{DIGIT}+[eE][+-]{DIGIT}+ {printf("IntNum: %s\n",yytext);}  ({DIGIT}+[.]{DIGIT}+)|({DIGIT}+[.]{DIGIT}+[eE][+-]{DIGIT}+) {printf("FloatNum: %s\n",yytext);}  if{KEYWORD} {printf("KEYWORD(%s): %s\n",yytext, yytext);}  {ID} {printf("Identifier: %s\n",yytext);}  [a-zA-Z\_]?\"(\\.|[^\\"\n])\*\" { printf("const\_string: %s\n",yytext); }  [a-zA-Z\_]?'(\\.|[^\\'\n])+' { printf("const\_char: %s\n",yytext); }  "..." { printf("ELLIPSIS: %s\n",yytext); }  ">>=" { printf("RIGHT\_ASSIGN: %s\n",yytext); }  "<<=" { printf("LEFT\_ASSIGN: %s\n",yytext); }  "+=" { printf("ADD\_ASSIGN: %s\n",yytext); }  "-=" { printf("SUB\_ASSIGN: %s\n",yytext); }  "\*=" { printf("MUL\_ASSIGN: %s\n",yytext); }  "/=" { printf("DIV\_ASSIGN: %s\n",yytext); }  "%=" { printf("MOD\_ASSIGN: %s\n",yytext); }  "&=" { printf("AND\_ASSIGN: %s\n",yytext); }  "^=" { printf("XOR\_ASSIGN: %s\n",yytext); }  "|=" { printf("OR\_ASSIGN: %s\n",yytext); }  ">>" { printf("RIGHT\_OP: %s\n",yytext); }  "<<" { printf("LEFT\_OP: %s\n",yytext); }  "++" { printf("INC\_OP: %s\n",yytext); }  "--" { printf("DEC\_OP: %s\n",yytext); }  "->" { printf("PTR\_OP: %s\n",yytext); }  "&&" { printf("AND\_OP: %s\n",yytext); }  "||" { printf("OR\_OP: %s\n",yytext); }  "<=" { printf("LE\_OP: %s\n",yytext); }  ">=" { printf("GE\_OP: %s\n",yytext); }  "==" { printf("EQ\_OP: %s\n",yytext); }  "!=" { printf("NE\_OP: %s\n",yytext); }  ";" { printf("';': %s\n",yytext); }  ("{"|"<%") { printf("'{': %s\n",yytext); }  ("}"|"%>") { printf("'}': %s\n",yytext); }  "," { printf("',': %s\n",yytext); }  ":" { printf("':': %s\n",yytext); }  "=" { printf("'=': %s\n",yytext); }  "(" { printf("'(': %s\n",yytext); }  ")" { printf("')': %s\n",yytext); }  ("["|"<:") { printf("'[': %s\n",yytext); }  ("]"|":>") { printf("']': %s\n",yytext); }  "." { printf("'.': %s\n",yytext); }  "&" { printf("'&': %s\n",yytext); }  "!" { printf("'!': %s\n",yytext); }  "~" { printf("'~': %s\n",yytext); }  "-" { printf("'-': %s\n",yytext); }  "+" { printf("'+': %s\n",yytext); }  "\*" { printf("'\*': %s\n",yytext); }  "/" { printf("'/': %s\n",yytext); }  "%" { printf("percent: %s\n",yytext); }  "<" { printf("'<': %s\n",yytext); }  ">" { printf("'>': %s\n",yytext); }  "^" { printf("'^': %s\n",yytext); }  "|" { printf("'|': %s\n",yytext); }  "?" { printf("'?': %s\n",yytext); }  [ \t\v\n\f]  . {printf("ERROR:%s\n",yytext);}  %%  int main(int argc,char \*\*argv)  {  if(argc>1) yyin=fopen(argv[1],"r");  else printf("error:\n command: lexC filename");  yylex();  return 0;  }  int yywrap()  {  return 1;  }  void comment(void)  {  char c, prev = 0;    while ((c = input()) != 0) /\* (EOF maps to 0) \*/  {  if (c == '/' && prev == '\*')  return;  prev = c;  }  } |

#### 五、测试报告

**测试样例：**

|  |
| --- |
| struct Node  {      char name[20];      int id;  } Node;  //test  int main()  {      struct Node node;      node.id = -123;      (&node)->id = 456;      int a = '2';      int num1, \_num2;      float b = 5.3e-2;      /\*calu a + b\*/      printf("%d + %f = %f", a, b, a + b);      return 0;  } |

**C++分析程序测试结果：**

|  |
| --- |
| <struct , struct> line:1  <Identifier , Node> line:1  <l\_brace , {> line:2  <char , char> line:3  <Identifier , name> line:3  <l\_square , [> line:3  <Numeric , 20> line:3  <r\_square , ]> line:3  <semi , ;> line:3  <int , int> line:4  <Identifier , id> line:4  <semi , ;> line:4  <r\_brace , }> line:5  <Identifier , Node> line:5  <semi , ;> line:5  <int , int> line:7  <Identifier , main> line:7  <l\_paren , (> line:7  <r\_paren , )> line:7  <l\_brace , {> line:8  <struct , struct> line:9  <Identifier , Node> line:9  <Identifier , node> line:9  <semi , ;> line:9  <Identifier , node> line:10  <point , .> line:10  <Identifier , id> line:10  <equal , => line:10  <minus , -> line:10  <Numeric , 123> line:10  <semi , ;> line:10  <l\_paren , (> line:11  <amp , &> line:11  <Identifier , node> line:11  <r\_paren , )> line:11  <minusgreater(arrow) , ->> line:11  <Identifier , id> line:11  <equal , => line:11  <Numeric , 456> line:11  <semi , ;> line:11  <int , int> line:12  <Identifier , a> line:12  <equal , => line:12  <const\_character , '2'> line:12  <semi , ;> line:12  <int , int> line:13  <Identifier , num1> line:13  <comma , ,> line:13  <Identifier , \_num2> line:13  <semi , ;> line:13  <float , float> line:14  <Identifier , b> line:14  <equal , => line:14  <Numeric , 5.3e-2> line:14  <semi , ;> line:14  <Identifier , printf> line:16  <l\_paren , (> line:16  const\_string , "%d + %f = %f"> line:16  <comma , ,> line:16  <Identifier , a> line:16  <comma , ,> line:16  <Identifier , b> line:16  <comma , ,> line:16  <Identifier , a> line:16  <plus , +> line:16  <Identifier , b> line:16  <r\_paren , )> line:16  <semi , ;> line:16  <return , return> line:17  <Numeric , 0> line:17  <semi , ;> line:17  <r\_brace , }> line:18  LineNum : 18  CharNum : 169  idNum : 20  KeywordNum9  numNum : 5  CharConstNum : 1  stringConstNum : 1  otherNum : 36 |

**Lex分析程序测试结果：**

|  |
| --- |
| Identifier: struct  Identifier: Node  '{': {  Identifier: char  Identifier: name  '[': [  IntNum: 20  ']': ]  ';': ;  Identifier: int  Identifier: id  ';': ;  '}': }  Identifier: Node  ';': ;  Identifier: int  Identifier: main  '(': (  ')': )  '{': {  Identifier: struct  Identifier: Node  Identifier: node  ';': ;  Identifier: node  '.': .  Identifier: id  '=': =  '-': -  IntNum: 123  ';': ;  '(': (  '&': &  Identifier: node  ')': )  PTR\_OP: ->  Identifier: id  '=': =  IntNum: 456  ';': ;  Identifier: int  Identifier: a  '=': =  const\_char: '2'  ';': ;  Identifier: int  Identifier: num1  ',': ,  Identifier: \_num2  ';': ;  Identifier: float  Identifier: b  '=': =  FloatNum: 5.3e-2  ';': ;  Identifier: printf  '(': (  const\_string: "%d + %f = %f"  ',': ,  Identifier: a  ',': ,  Identifier: b  ',': ,  Identifier: a  '+': +  Identifier: b  ')': )  ';': ;  Identifier: return  IntNum: 0  ';': ;  '}': } |

**分析说明：**

能够正确的分析处关键字和数字，并且统计个数，能够正确的处理换行和注释，测试其他特殊样例能够正确的报错。