**学生实验报告**

**专业 计算机科学与技术 姓名 严世军 学号 202226202046 日期 2024/10/23**

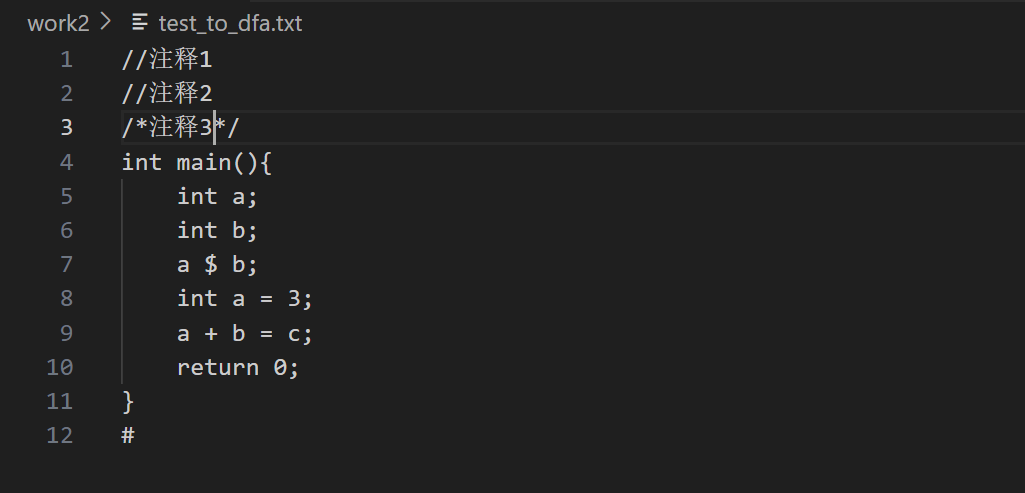
|  |  |  |  |
| --- | --- | --- | --- |
| 课程名称 | 编译原理 | 实验室名称 | X3408 |
| 实验名称 | 词法分析器 | | |
| 指导教师 | 漆志群 | 成绩 |  |
| 1. 实验目的 2. 熟悉词法分析、语法分析、语义分析和代码生成等编译过程。 3. 掌握编译器设计中的算法和数据结构。 4. 实现一个简单的编译器或其部分模块，了解程序从源代码到可执行代码的转化过程。 5. 通过将正则表达式转换为有限自动机，自动识别和提取词法单元。 6. 通过手动或工具生成词法分析器，实现从源代码输入到词法单元输出的完整过程。 | | | |
| 1. 实验原理和内容 2. 实验原理：   1.词法分析：将源代码转换为一系列符号（token），利用有限自动机和正则表达式识别代码中的词素（如标识符、关键字、运算符等）。  2.识别完成token，查看token是否存在错误  （2）实验内容  识别词法单元：通过词法分析器从输入的源代码中提取词法单元。词法单元包括标识符、关键字、数字、运算符、分隔符等。实现一个扫描器，依次扫描源代码字符，将其传递给DFA，识别并输出相应的词法单元。  词法分析错误处理：处理词法分析中的错误，如非法字符、未匹配的输入等。为词法分析器设计错误检测机制，遇到非法字符或无效输入时，提供合适的错误提示并处理。  对词法分析器进行功能测试，确保其能够正确识别各种词法单元。 | | | |
| 1. 实验步骤   设置一个名字表NameL和常数表ConstL，当遇到标识符时，将其字符串送入名字表NameL，并把其名字表地址作为标识符的语义Seman值。常数情形也一样，不要求翻译成二进制数。要求在NameL和ConstL表中没有相同元素，同时具有简单的错误处理功能（找出源程序中所有存在的词法错误并指出错误所在的行数）。试用高级程序设计语言编写一个针对下述单词集的词法分析器。  自动机能够识别的内容如下：  //关键字识别  "int", "char", "if", "else", "while", "return", "void", "for", "do", "switch", "auto", "double", "long", "enum", "register",  "case", "break", "continue", "default", "sizeof", "typedef", "struct", "union", "main", "extern", "const", "float", "short",  "unsigned", "signed", "goto", "volatile", "do", "static"  //符号识别   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **符号** | **class** |  | **符号** | **class** | | ~ | 31 |  | %= | 40 | | , | 46 |  | % | 17 | | [ | 47 |  | >= | 21 | | ] | 48 |  | > | 20 | | ( | 49 |  | >>= | 44 | | ) | 50 |  | >> | 34 | | { | 51 |  | <= | 23 | | } | 52 |  | < | 22 | | ; | 53 |  | <<= | 45 | | . | 54 |  | << | 33 | | : | 55 |  | != | 24 | | ++ | 18 |  | ! | 26 | | += | 36 |  | == | 25 | | + | 13 |  | = | 35 | | -- | 19 |  | &= | 41 | | -= | 37 |  | && | 27 | | - | 14 |  | & | 29 | | \*= | 38 |  | **|=** | 42 | | \* | 15 |  | **||** | 28 | | /= | 39 |  | **|** | 30 | | / | 16 |  | ^= | 43 | | ^ | 32 |  |  |  |   //数字常量识别 | | | |

1. 程序及运行结果（或实验数据记录及分析）

/\* if: token.class=57 \*/  
/\* else: token.class=58 \*/  
/\* for: token.class=59 \*/  
/\* while: token.class=60 \*/  
/\* break: token.class=61 \*/  
/\* return: token.class=62 \*/  
/\* continue: token.class=63 \*/  
/\* float: token.class=64 \*/  
/\* int: token.class=65 \*/  
/\* char：token.class=66 \*/  
/\* 标识符：token.class=11 \*/  
/\* 正整数：token.class=12 \*/  
/\* 正实数：token.class=12 \*/  
/\* 零：token.class=12 \*/  
//算术运算符  
/\* "+"：token.class=13 \*/  
/\* "-"：token.class=14 \*/  
/\* "\*"：token.class=15 \*/  
/\* "/"：token.class=16 \*/  
/\* "%"：token.class=17 \*/  
/\* "++"：token.class=18 \*/  
/\* "--"：token.class=18 \*/  
//关系运算符  
/\* ">"：token.class=19 \*/  
/\* ">="：token.class=20 \*/  
/\* "<"：token.class=21 \*/  
/\* "<="：token.class=22 \*/  
/\* "!="：token.class=23 \*/  
/\* "=="：token.class=24 \*/  
//逻辑运算符  
/\* "!"：token.class=25 \*/  
/\* "&&"：token.class=26 \*/  
/\* "||"：token.class=27 \*/  
//位操作运算符  
/\* "&"：token.class=28 \*/  
/\* "|"：token.class=29 \*/  
/\* "~"：token.class=30 \*/  
/\* "^"：token.class=31 \*/  
/\* "<<"：token.class=32 \*/  
/\* ">>"：token.class=33 \*/  
//赋值运算符  
/\* "="：token.class=34 \*/  
/\* "+="：token.class=35 \*/  
/\* "-="：token.class=36 \*/  
/\* "\*="：token.class=37 \*/  
/\* "/="：token.class=38 \*/  
/\* "%="：token.class=39 \*/  
/\* "&="：token.class=40 \*/  
/\* "|="：token.class=41 \*/  
/\* "^="：token.class=42 \*/  
/\* ">>="：token.class=43 \*/  
/\* "<<="：token.class=44 \*/  
//界符  
/\* ","：token.class=45 \*/  
/\* "["：token.class=46 \*/  
/\* "]"：token.class=47 \*/  
/\* "("：token.class=48 \*/  
/\* ")"：token.class=49 \*/  
/\* "{"：token.class=50 \*/  
/\* "}"：token.class=51 \*/  
/\* ";"：token.class=52 \*/  
/\* "."：token.class=53 \*/  
/\* ":"：token.class=54 \*/  
/\* "换行符"：token.class=55 \*/  
/\* "文件结束"：token.class=56 \*/  
#include "stdio.h"  
#include "string.h"  
#include "stdlib.h"  
#define MAX 100  
#define KEYWORD\_COUNT (sizeof(keywords) / sizeof(keywords[0]))  
  
//关键字识别  
const char\* keywords[] = {  
 "int", "char", "if", "else", "while", "return", "void", "for", "do", "switch", "auto", "double", "long", "enum", "register",  
 "case", "break", "continue", "default", "sizeof", "typedef", "struct", "union", "main", "extern", "const", "float", "short",  
 "unsigned", "signed", "goto", "volatile", "do", "static"  
};  
  
// 定义结构体，用于保存正确的token  
struct a {  
 int class;  
 char seman[100];  
 int line\_number;  
 int const\_address; // 常数的地址，如果不是常数则为-1  
   
} token[800];  
  
//保存错误信息  
struct error\_info {  
 int line\_number;  
 char message[100];  
 char offending\_char;  
} errors[100];  
  
//保存注释信息  
struct comment\_info{  
 int start\_line;  
 int end\_line;  
 char content[200];  
} comments[100];  
  
  
  
int constants[100];// 常数表  
int constant\_count = 0;  
char identified\_identifiers[100][100]; // 保存识别出的标识符  
int identifier\_count = 0; // 标识符计数  
char input[MAX];  
int i = 0, k = 0, line = 1;  
int error\_count = 0; // 错误计数  
int comment\_count = 0; // 注释计数  
char identified\_keywords[100][100]; // 保存识别出的关键字  
int keyword\_count = 0; // 关键字计数  
  
int isletter(char x);  
int isdigit(char x);  
int is\_keyword(char\* name);  
void print\_token(int k, FILE\* output);  
void print\_keywords(FILE\* output); // 输出识别出的关键字  
void print\_constants(FILE\* output); // 输出识别出的常数表  
void error(int k, char offending\_char, const char \*message);  
void print\_errors(FILE\* output);  
void record\_comment(int start\_line, int end\_line, char\* content);  
void print\_comments(FILE\* output);  
int handle\_comment();  
int next\_token();  
  
// 判断是否为字母  
int isletter(char x) {  
 return (x >= 'a' && x <= 'z') || (x >= 'A' && x <= 'Z');  
}  
  
// 判断是否为数字  
int isdigit(char x) {  
 return (x >= '0' && x <= '9');  
}  
  
// 判断是否为关键字  
int is\_keyword(char\* name) {  
 for (int j = 0; j < KEYWORD\_COUNT; j++) {  
 if (strcmp(name, keywords[j]) == 0) {  
 return 1; // 是关键字  
 }  
 }  
 return 0; // 不是关键字  
}  
  
  
// 输出识别出的关键字列表  
void print\_keywords(FILE\* output) {  
 if (keyword\_count == 0) {  
 fprintf(output, "\n\nNo keywords identified.\n");  
 return;  
 }  
  
 fprintf(output, "\n\nIdentified Keywords: \n-----------------------------");  
 for (int i = 0; i < keyword\_count; i++) {  
 fprintf(output, "\n Keyword %d: %s", i + 1, identified\_keywords[i]);  
 }  
 fprintf(output, "\n\nTotal Keywords: %d\n", keyword\_count);  
}  
  
// 输出Token列表  
void print\_token(int k, FILE\* output) {  
 fprintf(output, "\n Token List: \n-----------------------------");  
 fprintf(output, "\n Number Class Seman Line Number");  
 for (int i = 0; i < k; i++) {  
 fprintf(output, "\n\t%-9d\t%-9d\t%-10s\tLine: %-10d", i + 1, token[i].class, token[i].seman, token[i].line\_number);  
 }  
}  
  
// 记录错误信息  
void error(int k, char offending\_char, const char \*message) {  
 errors[error\_count].line\_number = line;  
 errors[error\_count].offending\_char = offending\_char;  
 strcpy(errors[error\_count].message, message);  
 error\_count++;  
}  
  
// 输出错误列表  
void print\_errors(FILE\* output) {  
 if (error\_count == 0) {  
 fprintf(output, "\n\nNo errors detected.\n");  
 return;  
 }  
   
 fprintf(output, "\n\nError List: \n-----------------------------");  
 for (int i = 0; i < error\_count; i++) {  
 fprintf(output, "\n Error %d: Line %d: '%c' - %s", i + 1, errors[i].line\_number, errors[i].offending\_char, errors[i].message);  
 }  
 fprintf(output, "\n\nTotal Errors: %d\n", error\_count);  
}  
  
// 输出常数表  
void print\_constants(FILE\* output) {  
 if (constant\_count == 0) {  
 fprintf(output, "\n\nNo constants identified.\n");  
 return;  
 }  
  
 fprintf(output, "\n\nConstant Table: \n-----------------------------");  
 for (int i = 0; i < constant\_count; i++) {  
 fprintf(output, "\n Constant %d: %d", i + 1, constants[i]);  
 }  
 fprintf(output, "\n\nTotal Constants: %d\n", constant\_count);  
}  
  
  
// 记录注释  
void record\_comment(int start\_line, int end\_line, char\* content) {  
 comments[comment\_count].start\_line = start\_line;  
 comments[comment\_count].end\_line = end\_line;  
 strcpy(comments[comment\_count].content, content);  
 comment\_count++;  
}  
  
// 输出注释列表  
void print\_comments(FILE\* output) {  
 if (comment\_count == 0) {  
 fprintf(output, "\n\nNo comments detected.\n");  
 return;  
 }  
   
 fprintf(output, "\n\nComment List: \n-----------------------------");  
 for (int i = 0; i < comment\_count; i++) {  
 fprintf(output, "\n Comment %d: Lines %d-%d: %s", i + 1, comments[i].start\_line, comments[i].start\_line, comments[i].content);  
 }  
}  
  
/\* 处理注释 \*/  
int handle\_comment() {  
 char ch = input[i];  
 char comment\_content[200];  
 int comment\_start\_line = line; // 记录注释开始的行号  
 int l = 0;  
  
 if (ch == '/') {  
 ch = input[++i];  
 if (ch == '/') { // 单行注释  
 comment\_content[l++] = '/';  
 while (input[i] != '\n' && input[i] != '#') {  
 comment\_content[l++] = input[i++];  
 }  
 comment\_content[l] = '\0';  
 record\_comment(comment\_start\_line, comment\_start\_line, comment\_content); // 单行注释结束，开始行号和结束行号相同  
 line++; // 因为是单行注释，换行处理  
 return 1; // 注释结束，返回继续分析下一个 token  
 } else if (ch == '\*') { // 多行注释  
 comment\_content[l++] = '/';  
 comment\_content[l++] = '\*'; // 添加注释的开头部分  
 i++;  
 while (!(input[i] == '\*' && input[i + 1] == '/')) {  
 if (input[i] == '\n' || input[i] == '\t') {  
 line++; // 多行注释时，遇到换行符更新行号  
 }  
 comment\_content[l++] = input[i++];  
   
 }  
 comment\_content[l++] = '\*';  
 comment\_content[l++] = '/';  
 comment\_content[l] = '\0';  
 record\_comment(comment\_start\_line, line, comment\_content); // 记录多行注释的开始行号和结束行号  
 i += 2; // 跳过 "\*/"  
 line++;  
 return 2; // 注释结束，返回继续分析下一个 token  
 }  
 }  
 return 0; // 不是注释，返回 0  
}  
  
// 输出识别出的标识符列表  
void print\_identifiers(FILE\* output) {  
 if (identifier\_count == 0) {  
 fprintf(output, "\n\nNo identifiers identified.\n");  
 return;  
 }  
  
 fprintf(output, "\n\nIdentified Identifiers: \n-----------------------------");  
 for (int i = 0; i < identifier\_count; i++) {  
 fprintf(output, "\n Identifier %d: %s", i + 1, identified\_identifiers[i]);  
 }  
 fprintf(output, "\n\nTotal Identifiers: %d\n", identifier\_count);  
}  
  
  
// 获取下一个Token  
int next\_token() {  
 char ch, name[10];  
 int state, l;  
 ch = input[i];  
 while (ch == ' ' || ch == '\t' || ch == '\n' || ch == '\r') {  
 if (ch == '\n' || ch == '\r') line++;  
 ch = input[++i];  
 }  
 state = 0;  
 while(1)  
 {  
 switch(state)  
 {  
 case 0:  
 if(handle\_comment() > 0) return; //Check if it is a comment  
 if(isletter(ch)){ l=0; name[l++]=ch; state=1; token[k].line\_number = line;}  
 else if(isdigit(ch)) { l=0; name[l++]=ch; state=2; token[k].line\_number = line;}  
 else if(ch=='+') {state=3; token[k].line\_number = line;}  
 else if(ch=='-') {state=4; token[k].line\_number = line;}  
 else if(ch=='\*') {state=5; token[k].line\_number = line;}  
 else if(ch=='/') {state=6; token[k].line\_number = line;}  
 else if(ch=='%') {state=7; token[k].line\_number = line;}  
 else if(ch=='>') {state=8; token[k].line\_number = line;}  
 else if(ch=='<') {state=9; token[k].line\_number = line;}  
 else if(ch=='!') {state=10; token[k].line\_number = line;}  
 else if(ch=='=') {state=11; token[k].line\_number = line;}  
 else if(ch=='&') {state=12; token[k].line\_number = line;}  
 else if(ch=='|') {state=13; token[k].line\_number = line;}  
 else if(ch=='~') {state=14; token[k].line\_number = line;}  
 else if(ch=='^') {state=15; token[k].line\_number = line;}  
 else if(ch==',') {state=16; token[k].line\_number = line;}  
 else if(ch=='[') {state=17; token[k].line\_number = line;}  
 else if(ch==']') {state=18; token[k].line\_number = line;}  
 else if(ch=='(') {state=19; token[k].line\_number = line;}  
 else if(ch==')') {state=20; token[k].line\_number = line;}  
 else if(ch=='{') {state=21; token[k].line\_number = line;}  
 else if(ch=='}') {state=22; token[k].line\_number = line;}  
 else if(ch==';') {state=23; token[k].line\_number = line;}  
 else if(ch=='.') {state=24; token[k].line\_number = line;}  
 else if(ch==':') {state=25; token[k].line\_number = line;}  
 else if(ch=='#') {i--;return;}  
 else{  
 errors[error\_count].line\_number = line;  
 error(k, ch, "Illegal character!");  
 i++;  
 return;  
 }  
 break;  
 case 1:  
 ch = input[++i];  
 if (isletter(ch) || isdigit(ch)) {   
 name[l++] = ch;   
 state = 1;  
 } else {  
 name[l] = '\0'; // 终止字符串  
 if (is\_keyword(name)) {   
 token[k].class = 3; // 假设 3 表示关键字  
 strcpy(token[k].seman, name);  
   
 // 记录识别出的关键字  
 strcpy(identified\_keywords[keyword\_count], name);  
 keyword\_count++;  
 } else {  
 token[k].class = 1; // 表示标识符  
 strcpy(token[k].seman, name);  
 strcpy(identified\_identifiers[identifier\_count], name); // 记录识别出的标识符  
 identifier\_count++;  
 }  
 token[k].const\_address = -1; // 非常数，地址为-1  
 token[k].line\_number = line;  
 k++;  
 i--; // 回退一个字符，以便继续分析  
 return;  
 }  
 break;  
 case 2: // 数字常量  
 ch = input[++i];  
 while(isdigit(ch)) {  
 name[l++] = ch;  
 ch = input[++i];  
 }  
 name[l] = '\0'; // 终止字符串  
 // 将常数保存到常数表  
 constants[constant\_count] = atoi(name); // 转换为整数  
 token[k].class = 4; // 假设 2 表示常数  
 strcpy(token[k].seman, name);  
 token[k].const\_address = constant\_count; // 记录常数在常数表中的地址  
 constant\_count++;  
 token[k].line\_number = line;  
 k++;  
 i--; // 回退一个字符  
 return;break;  
 case 3:ch=input[++i];  
 if(ch=='+') state=26; else if(ch=='=') state=27; else state=28; break;  
 case 4:ch=input[++i];  
 if(ch=='-') state=29; else if(ch=='=') state=30; else state=31; break;  
 case 5:ch=input[++i];  
 if(ch=='=') state=32; else state=33; break;  
 case 6:ch=input[++i];  
 if(ch=='=') state=34; else state=35; break;  
 case 7:ch=input[++i];  
 if(ch=='=') state=36; else state=37; break;  
 case 8:ch=input[++i];  
 if(ch=='=') state=38; else if(ch=='>') state=39; else state=40; break;  
 case 9:ch=input[++i];  
 if(ch=='=') state=43; else if(ch=='<') state=44; else state=45; break;  
 case 10:ch=input[++i];  
 if(ch=='=') state=48; else state=49; break;  
  
 case 11:ch=input[++i];  
 if(ch=='=') state=50; else state=51; break;  
 case 12:ch=input[++i];  
 if(ch=='=') state=52; else if(ch=='&') state=53; else state=54; break;  
 case 13:ch=input[++i];  
 if(ch=='=') state=55; else if(ch=='|') state=56; else state=57; break;  
 case 14:token[k].class=31; strcpy(token[k].seman,"~"); k++; return;/\*判断位操作符~ \*/  
 case 15:ch=input[++i];  
 if(ch=='=') state=58; else state=59; break;  
 case 16:token[k].class=46; strcpy(token[k].seman,","); k++; return;  
 case 17:token[k].class=47; strcpy(token[k].seman,"["); k++; return;  
 case 18:token[k].class=48; strcpy(token[k].seman,"]"); k++; return;  
 case 19:token[k].class=49; strcpy(token[k].seman,"("); k++; return;  
 case 20:token[k].class=50; strcpy(token[k].seman,")"); k++; return;  
 case 21:token[k].class=51; strcpy(token[k].seman,"{"); k++; return;  
 case 22:token[k].class=52; strcpy(token[k].seman,"}"); k++; return;  
 case 23:token[k].class=53; strcpy(token[k].seman,";"); k++; return;  
 case 24:token[k].class=54; strcpy(token[k].seman,"."); k++; return;  
 case 25:token[k].class=55; strcpy(token[k].seman,":"); k++; return;  
  
 case 26:token[k].class=18; strcpy(token[k].seman,"++"); k++; i--; return;/\*判断算术运算符++ \*/  
 case 27:token[k].class=36; strcpy(token[k].seman,"+="); k++; i--; return;/\*判断赋值运算符+= \*/  
 case 28:token[k].class=13; strcpy(token[k].seman,"+"); k++; i--; return;/\*判断算术运算符+ \*/  
  
 case 29:token[k].class=19; strcpy(token[k].seman,"--"); k++; i--; return;/\*判断算术运算符-- \*/  
 case 30:token[k].class=37; strcpy(token[k].seman,"-="); k++; i--; return;/\*判断赋值运算符-= \*/  
 case 31:token[k].class=14; strcpy(token[k].seman,"-"); k++; i--; return;/\*判断算术运算符- \*/  
  
 case 32:token[k].class=38; strcpy(token[k].seman,"\*="); k++; i--; return;/\*判断赋值运算符\*= \*/  
 case 33:token[k].class=15; strcpy(token[k].seman,"\*"); k++; i--; return;/\*判断算术运算符\* \*/  
  
 case 34:token[k].class=39; strcpy(token[k].seman,"/="); k++; i--; return;/\*判断赋值运算符/= \*/  
 case 35:token[k].class=16; strcpy(token[k].seman,"/"); k++; i--; return;/\*判断算术运算符/ \*/  
  
 case 36:token[k].class=40; strcpy(token[k].seman,"%="); k++; i--; return;/\*判断赋值运算符%= \*/  
 case 37:token[k].class=17; strcpy(token[k].seman,"%"); k++; i--; return;/\*判断算术运算符% \*/  
  
 case 38:token[k].class=21; strcpy(token[k].seman,">="); k++; i--; return;/\*判断关系运算符>= \*/  
 case 39:ch=input[++i];  
 if(ch=='=') state=41; else state=42; break;  
 case 40:token[k].class=20; strcpy(token[k].seman,">"); k++; i--; return;/\*判断关系运算符> \*/  
 case 41:token[k].class=44; strcpy(token[k].seman,">>="); k++; i--; return;/\*判断赋值运算符>>= \*/  
 case 42:token[k].class=34; strcpy(token[k].seman,">>"); k++; i--; return;/\*判断位操作运算符>> \*/  
  
 case 43:token[k].class=23; strcpy(token[k].seman,"<="); k++; return;/\*判断关系运算符<= \*/  
 case 44:ch=input[++i];  
 if(ch=='=') state=46; else state=47; break;  
 case 45:token[k].class=22; strcpy(token[k].seman,"<"); k++; i--; return;/\*判断关系运算符< \*/  
 case 46:token[k].class=45; strcpy(token[k].seman,"<<="); k++; i--; return;/\*判断赋值运算符<<= \*/  
 case 47:token[k].class=33; strcpy(token[k].seman,"<<"); k++; i--; return;/\*判断位操作运算符<< \*/  
  
 case 48:token[k].class=24; strcpy(token[k].seman,"!="); k++; i--; return;/\*判断赋值运算符!= \*/  
 case 49:token[k].class=26; strcpy(token[k].seman,"!"); k++; i--; return;/\*判断算术运算符! \*/  
  
 case 50:token[k].class=25; strcpy(token[k].seman,"=="); k++; i--; return;/\*判断赋值运算符== \*/  
 case 51:token[k].class=35; strcpy(token[k].seman,"="); k++; i--; return;/\*判断算术运算符= \*/  
  
 case 52:token[k].class=41; strcpy(token[k].seman,"&="); k++; i--; return;/\*判断赋值运算符&= \*/  
 case 53:token[k].class=27; strcpy(token[k].seman,"&&"); k++; i--; return;/\*判断逻辑运算符&& \*/  
 case 54:token[k].class=29; strcpy(token[k].seman,"&"); k++; i--; return;/\*判断位操作运算符& \*/  
  
 case 55:token[k].class=42; strcpy(token[k].seman,"|="); k++; i--; return;/\*判断赋值运算符|= \*/  
 case 56:token[k].class=28; strcpy(token[k].seman,"||"); k++; i--; return;/\*判断逻辑运算符|| \*/  
 case 57:token[k].class=30; strcpy(token[k].seman,"|"); k++; i--; return;/\*判断位操作运算符| \*/  
  
 case 58:token[k].class=43; strcpy(token[k].seman,"^="); k++; i--; return;/\*判断赋值运算符^= \*/  
 case 59:token[k].class=32; strcpy(token[k].seman,"^"); k++; i--; return;/\*判断位操作运算符^ \*/  
  
 case 60:token[k].class=1; name[l]='\0'; strcpy(token[k].seman, name); token[k].line\_number = line; k++; i--; return;  
 case 61:token[k].class=2; name[l]='\0'; strcpy(token[k].seman, name); token[k].line\_number = line; k++; i--; return;  
  
 default:   
 error(k,ch, "Unknown error occurred!");  
 return;  
 }  
 }  
}  
  
// 主函数  
int main() {  
 FILE \*input\_file, \*output\_file;  
 char ch;  
 int j = 0;  
  
 // 打开文件  
 input\_file = fopen("test\_to\_dfa.txt", "r");  
 output\_file = fopen("output1.txt", "w");  
  
 if (input\_file == NULL || output\_file == NULL) {  
 printf("File could not be opened.\n");  
 return 1;  
 }  
  
 // 从文件读取输入  
 while ((ch = fgetc(input\_file)) != EOF) {  
 input[j++] = ch;  
 }  
 input[j] = '#';  
  
 // 执行词法分析  
 i = 0;  
 while (input[i] != '#') {  
 next\_token();  
 i++;  
 }  
  
 if (input[i] == '#') {  
 token[k].class = 57;  
 strcpy(token[k].seman, "#");  
 token[k].line\_number = line;  
 k++;  
 fprintf(output\_file, "\n\nScanner completed successfully!");  
 print\_token(k, output\_file);  
 }  
 //打印常数表  
 print\_constants(output\_file);  
  
 // 打印识别出的关键字  
 print\_keywords(output\_file);  
  
 // 输出错误和注释  
 print\_errors(output\_file);  
 print\_comments(output\_file);  
  
 //打印标识符  
 print\_identifiers(output\_file);  
  
 // 关闭文件  
 fclose(input\_file);  
 fclose(output\_file);  
 return 0;  
}

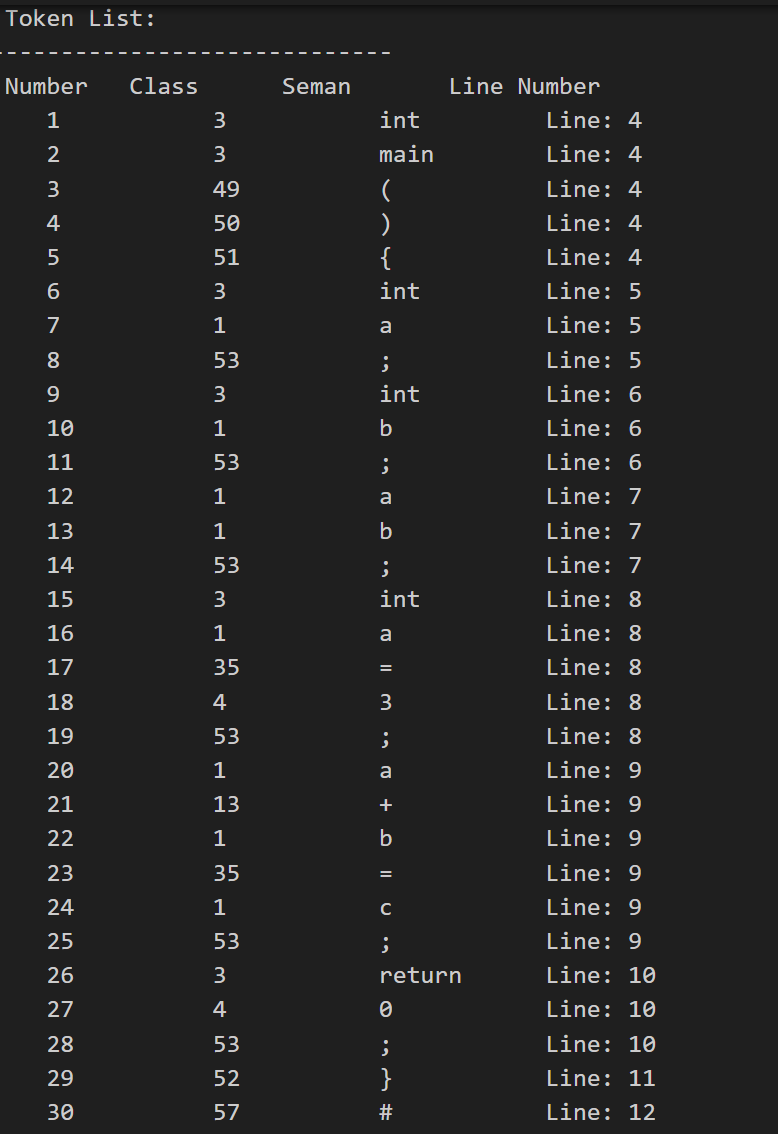
执行结果

输入文件内容：

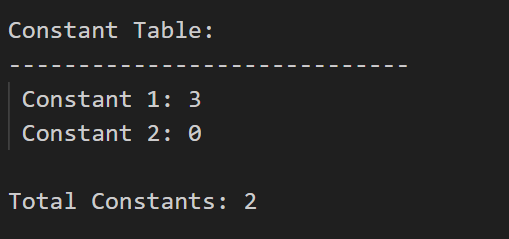


输出文件内容

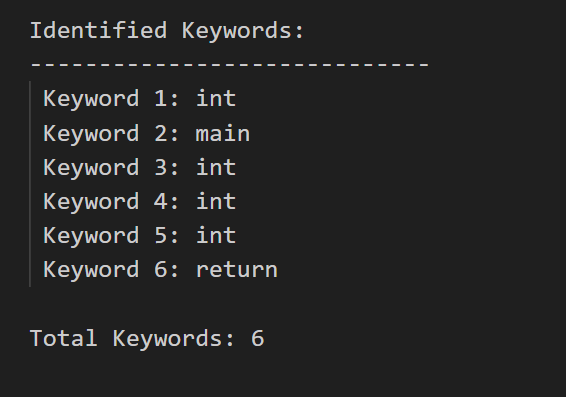
1.token表



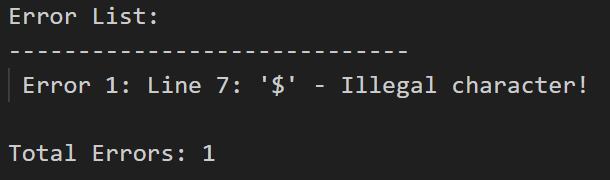
2.常量表



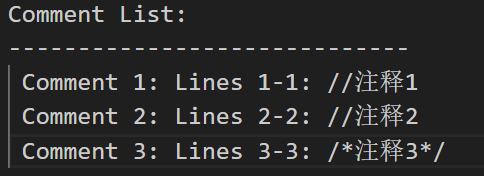
3.关键字表



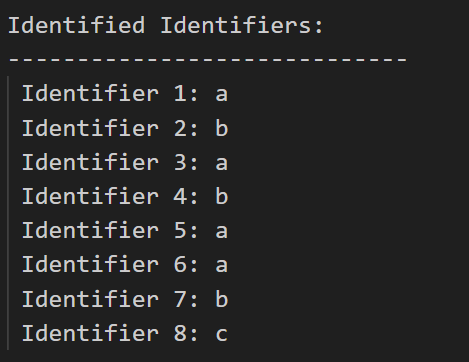
4.错误信息列表



5.注释列表



6.用户自定义标识符列表



所能够实现的功能：

①使用文件输入、输出；

②可以查找多个错误；

③每个错误有错误提示，包括错误所在行、错误信息；

④错误提示信息区分类别；

⑤同名标识符或常数重复存放检查；

⑥正确建立单词表和名字表、常量表的链接；

⑦处理注释。