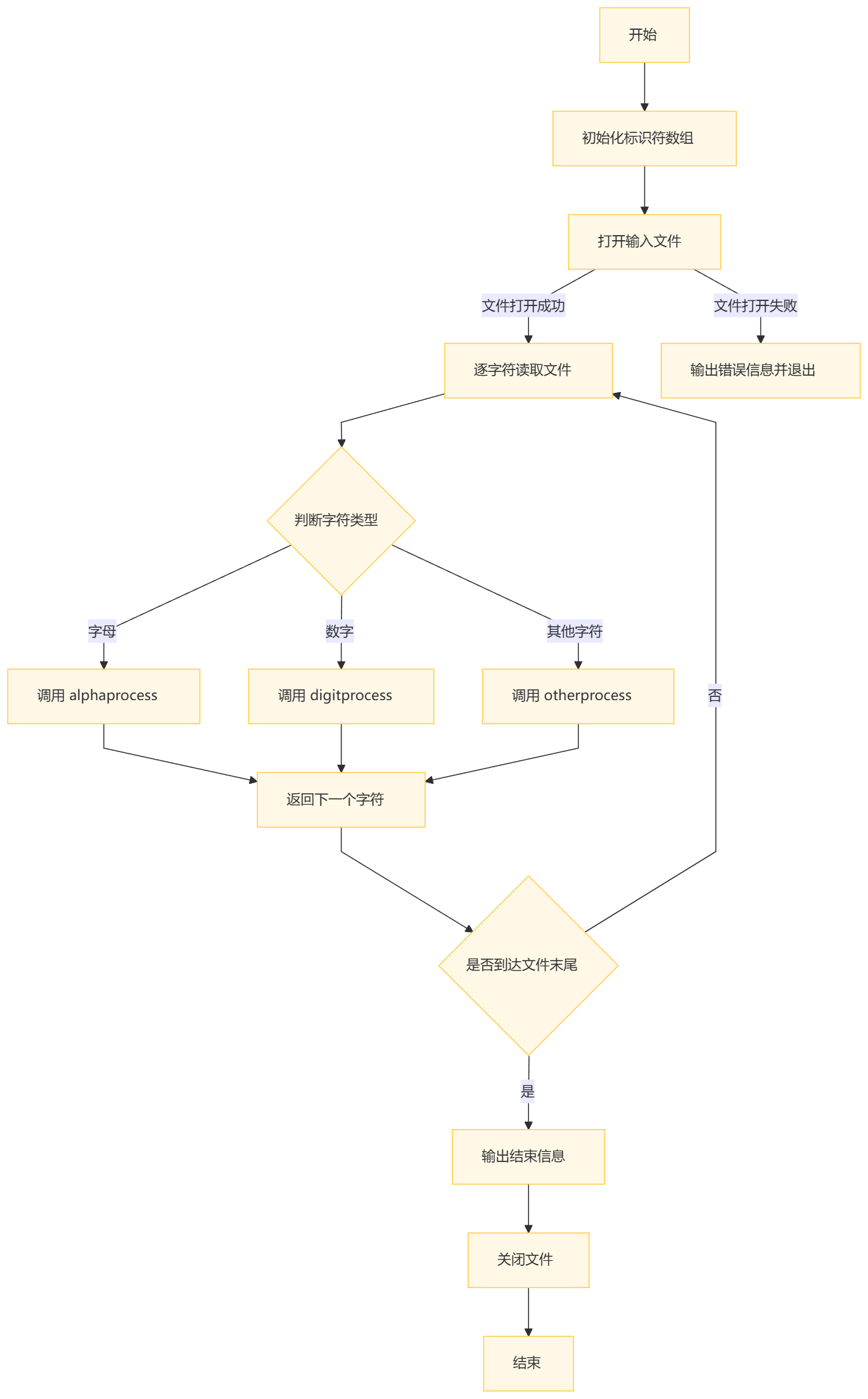
**编译原理实验报告**

**实验名称：词法分析**

**2208010421-王旭**



# 实验目的和要求：

编制一个读单词过程，从输入的源程序中，识别出各个具有独立意义的单词，即基本保留字、标识符、常数、运算符、分隔符五大类。

# 实验内容：

本实验的目标是实现一个词法分析器，能够从输入的源代码文件中识别并分类单词，具体包括以下几类：

****关键字****（如int、if、while等）。

****标识符****（如变量名、函数名等）。

****常数****（包括整数和浮点数）。

****运算符****（如+、-、\*、/等）。

****分隔符****（如{、}、;等）。

# 实验步骤：

****构造程序所需的模块****

主函数：负责读取输入文件并逐行调用扫描函数。

扫描函数：负责逐字符分析输入行，识别单词并分类。

辅助函数：用于判断单词类型（如是否为关键字、运算符等）。

****编写模块代码****

使用C语言实现词法分析器，通过fgetc逐字符读取输入文件。

根据字符类型（字母、数字、特殊字符等）调用不同的处理函数（如alphaprocess、digitprocess、otherprocess）。

识别关键字、标识符、数字、运算符和分隔符，并输出其类型和值。

****在主函数中调用模块****

打开输入文件并逐行读取。

对每一行调用扫描函数，识别并输出单词类型和值。

处理文件结束符EOF，完成词法分析。

****主函数****  
负责打开输入文件并逐字符读取，根据字符类型调用相应的处理函数。

****扫描函数****

alphaprocess：处理字母开头的单词，识别关键字和标识符。

digitprocess：处理数字，识别整数和浮点数。

otherprocess：处理特殊字符，识别运算符、分隔符和关系运算符。

****辅助函数****

search：用于查找单词是否为关键字、运算符、分隔符等。

is\_keyword、is\_operator、is\_delimiter：辅助函数，用于判断单词类型。

# 实验代码：

#include <stdio.h>

#include <ctype.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_TOKEN\_LENGTH 20

#define MAX\_LABELS 20

FILE \*fp;

char cbuffer;

char \*key[8] = {"if", "else", "for", "while", "do", "return", "break", "continue"};

char \*border[6] = {",", ";", "{", "}", "(", ")"};

char \*arithmetic[4] = {"+", "-", "\*", "/"};

char \*relation[6] = {"<", "<=", "=", ">", ">=", "<>"};

char \*consts[10] = {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9"};

char \*label[MAX\_LABELS];

int constnum = 0, labelnum = 0;

int search(char searchchar[], int wordtype) {

int i = 0;

switch (wordtype) {

case 1: // 关键字

for (i = 0; i < 8; i++) {

if (strcmp(key[i], searchchar) == 0)

return i + 1;

}

break;

case 2: // 分隔符

for (i = 0; i < 6; i++) {

if (strcmp(border[i], searchchar) == 0)

return i + 1;

}

break;

case 3: // 运算符

for (i = 0; i < 4; i++) {

if (strcmp(arithmetic[i], searchchar) == 0)

return i + 1;

}

break;

case 4: // 关系运算符

for (i = 0; i < 6; i++) {

if (strcmp(relation[i], searchchar) == 0)

return i + 1;

}

break;

case 5: // 数字

for (i = 0; i < 10; i++) {

if (strcmp(consts[i], searchchar) == 0)

return i + 1;

}

break;

case 6: // 标识符

for (i = 0; i < labelnum; i++) {

if (strcmp(label[i], searchchar) == 0)

return i + 1;

}

label[labelnum] = (char \*)malloc(strlen(searchchar) + 1);

strcpy(label[labelnum], searchchar);

labelnum++;

return i + 1;

}

return 0;

}

char alphaprocess(char buffer) {

int atype;

int i = -1;

char alphatp[MAX\_TOKEN\_LENGTH];

while ((isalpha(buffer)) || (isdigit(buffer))) {

alphatp[++i] = buffer;

buffer = fgetc(fp);

}

alphatp[i + 1] = '\0';

atype = search(alphatp, 1); // 检查是否为关键字

if (atype) {

printf("%s \t(1,%d)\n", alphatp, atype);

} else {

atype = search(alphatp, 6); // 检查是否为标识符

printf("%s \t(6,%d)\n", alphatp, atype);

}

return buffer;

}

char digitprocess(char buffer) {

int i = -1;

char digittp[MAX\_TOKEN\_LENGTH];

while (isdigit(buffer)) {

digittp[++i] = buffer;

buffer = fgetc(fp);

}

digittp[i + 1] = '\0';

printf("%s \t(5,%d)\n", digittp, atoi(digittp)); // 输出数字及其值

return buffer;

}

char otherprocess(char buffer) {

int otype;

char othertp[3]; // 增加长度以支持多字符运算符

othertp[0] = buffer;

othertp[1] = '\0';

// 检查是否为运算符或关系运算符

otype = search(othertp, 3);

if (otype) {

printf("%s \t(3,%d)\n", othertp, otype);

buffer = fgetc(fp);

return buffer;

}

otype = search(othertp, 4);

if (otype) {

char nextChar = fgetc(fp);

othertp[1] = nextChar;

othertp[2] = '\0';

int otypetp = search(othertp, 4);

if (otypetp) {

printf("%s \t(4,%d)\n", othertp, otypetp);

} else {

othertp[1] = '\0';

printf("%s \t(4,%d)\n", othertp, otype);

ungetc(nextChar, fp); // 将下一个字符放回输入流

}

return fgetc(fp); // 继续读取下一个字符

}

// 检查是否为分隔符

otype = search(othertp, 2);

if (otype) {

printf("%s \t(2,%d)\n", othertp, otype);

buffer = fgetc(fp);

return buffer;

}

// 检查是否为赋值运算符 :=

if (buffer == ':') {

buffer = fgetc(fp);

if (buffer == '=') {

printf(":= \t(2,2)\n");

buffer = fgetc(fp);

return buffer;

} else {

ungetc(buffer, fp); // 将字符放回输入流

printf(": \t(2,1)\n");

buffer = fgetc(fp);

return buffer;

}

}

// 检查是否为单字符符号（如 '.', '' 等）

if (buffer == '.') {

printf(". \t(3,4)\n"); // 假设 '.' 是运算符，编号为 4

buffer = fgetc(fp);

return buffer;

}

if (buffer == '\'') {

char charLiteral[3];

charLiteral[0] = buffer;

buffer = fgetc(fp); // 读取字符

charLiteral[1] = buffer;

buffer = fgetc(fp); // 读取闭合的单引号

charLiteral[2] = buffer;

printf("'%c' \t(5,%d)\n", charLiteral[1], charLiteral[1]); // 假设字符常量的编号为 5

buffer = fgetc(fp);

return buffer;

}

// 新增对 '#'、'%'、'\' 的处理

if (buffer == '#') {

printf("# \t(3,5)\n"); // 假设 '#' 是运算符，编号为 5

buffer = fgetc(fp);

return buffer;

}

if (buffer == '%') {

printf("%% \t(3,6)\n"); // 假设 '%' 是运算符，编号为 6

buffer = fgetc(fp);

return buffer;

}

if (buffer == '\\') {

printf("\\ \t(3,7)\n"); // 假设 '\' 是运算符，编号为 7

buffer = fgetc(fp);

return buffer;

}

// 新增对双引号及其内部字符串的处理

if (buffer == '"') {

char stringLiteral[MAX\_TOKEN\_LENGTH];

int i = 0;

buffer = fgetc(fp); // 读取双引号内的第一个字符

while (buffer != '"' && buffer != EOF) {

stringLiteral[i++] = buffer;

buffer = fgetc(fp);

}

stringLiteral[i] = '\0';

if (buffer == '"') {

printf("\"%s\" \t(6,%d)\n", stringLiteral, i); // 假设字符串常量的编号为 6

} else {

printf("Error: Unterminated string literal\n");

}

buffer = fgetc(fp);

return buffer;

}

// 如果无法识别，输出错误信息

if ((buffer != '\n') && (buffer != ' ')) {

printf("%c error, not a word\n", buffer);

}

buffer = fgetc(fp);

return buffer;

}

int main() {

int i;

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

label[i] = NULL;

}

if ((fp = fopen("example.cpp", "r")) == NULL) {

printf("File open error\n");

return 1;

}

cbuffer = fgetc(fp);

while (cbuffer != EOF) {

if (isalpha(cbuffer)) {

cbuffer = alphaprocess(cbuffer);

} else if (isdigit(cbuffer)) {

cbuffer = digitprocess(cbuffer);

} else {

cbuffer = otherprocess(cbuffer);

}

}

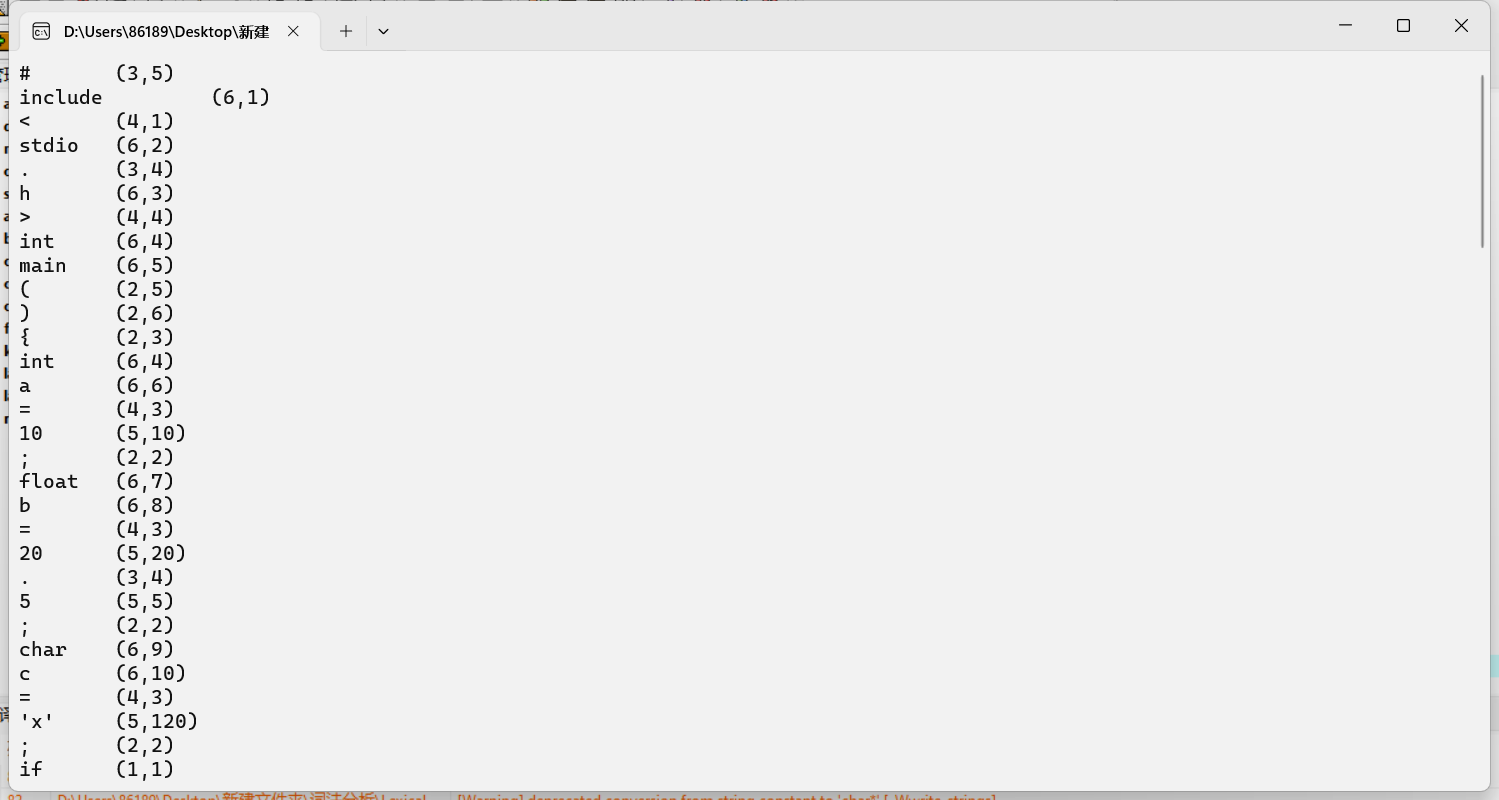
printf("Lexical analysis complete.\n");

fclose(fp);

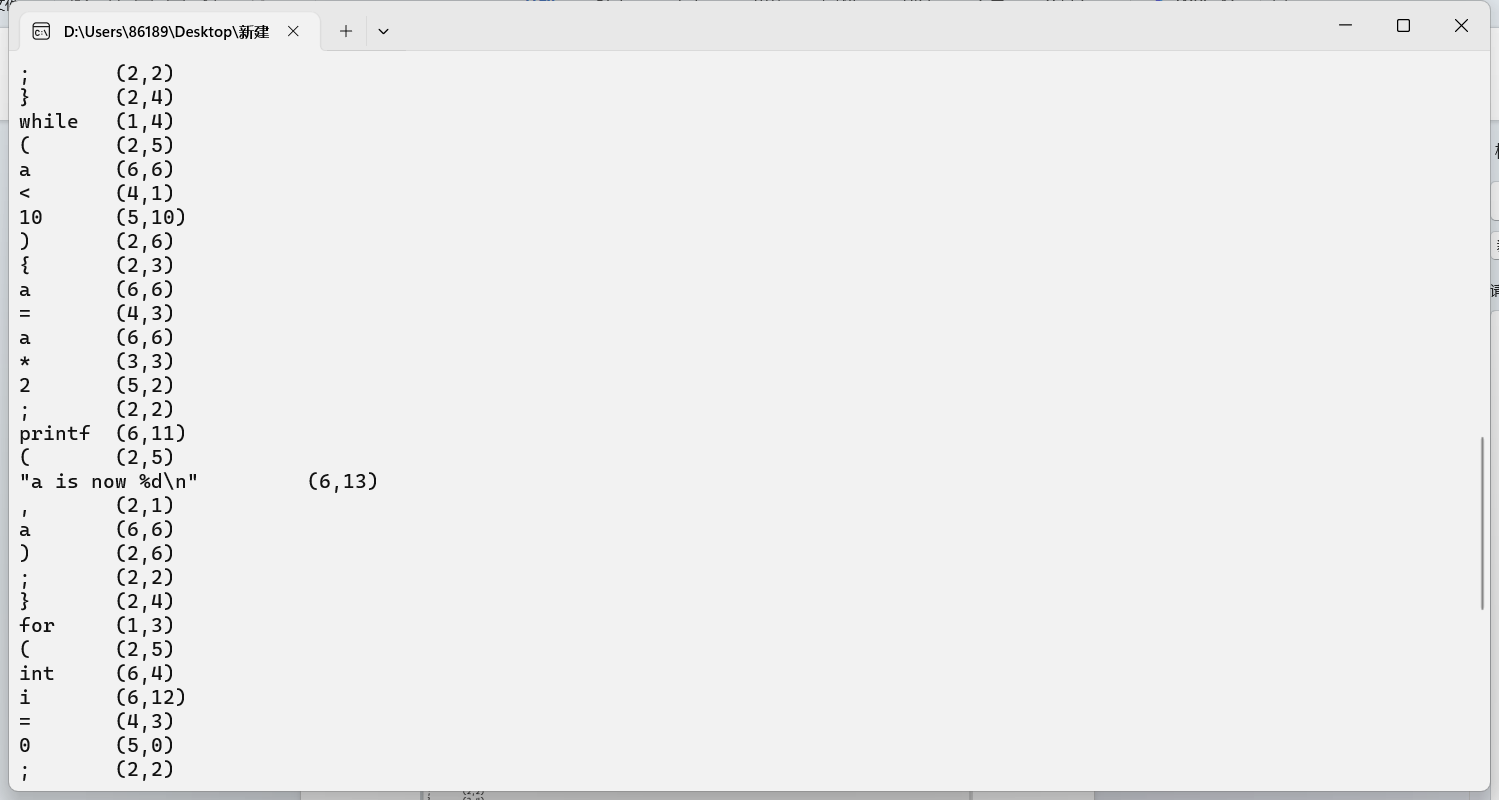
return 0;

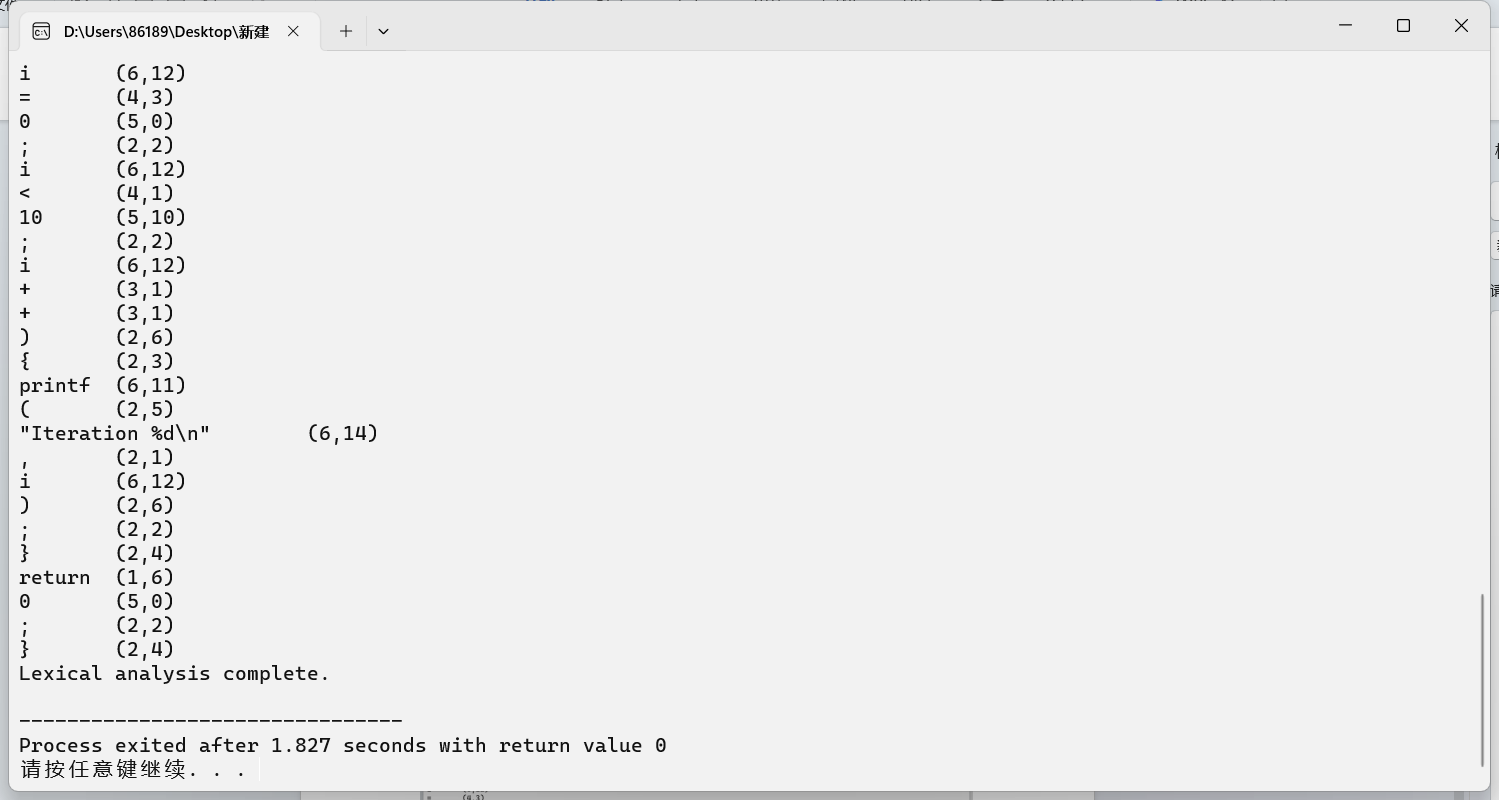
}

# 运行截图：









# 实验总结：

****实验结果****  
本实验成功实现了词法分析器的功能，能够正确识别关键字、标识符、常数、运算符和分隔符。通过输入文件example.c，程序能够逐行读取并分析单词，输出其类型和值。

****改进之处****

当前程序对某些特殊字符（如#、.、Tab键等）的处理还不够完善，需要进一步扩展字符集以支持更多符号。

在处理多字符运算符（如<=、>=）时，程序逻辑可以进一步优化，以提高效率和可读性。

当前程序对输入文件的格式有一定要求（如不能包含注释或特殊符号），后续可以增加对注释的处理能力。

****学习体会****  
通过本次实验，我加深了对词法分析的理解，掌握了如何通过程序实现单词的分类和识别。实验过程中，我复习了C语言的基本语法和文件操作，同时也学习了如何设计和实现一个简单的词法分析器。未来，我将继续学习编译原理的其他模块，如语法分析和语义分析，以进一步完善编译器的设计。