# 代码

#define \_CRT\_SECURE\_NO\_WARNINGS

#include"node.h"

#define MAXKEY 1000

int hash(char\*key) {

int h = 0, g;

while (\*key) {

h = (h << 4) + \*key++;

g = h & 0xf0000000;

if (g)

h ^= g >> 24;

h &= ~g;

}

return h % MAXKEY;

}

void insertHashTable(pNode\_t\*hashTable, pNode\_t pnode) {

int hashValue = hash(pnode->pWord);

if (hashTable[hashValue] == NULL) {

hashTable[hashValue] = pnode;

}

else {

pnode->pNext = hashTable[hashValue];//pHead就是hashValue

hashTable[hashValue] = pnode;

}

}

void findHashTable(pNode\_t\*hashTable, pNode\_t pinsert) {

int hashValue = hash(pinsert->pWord);

if (hashTable[hashValue] == NULL) {

insertHashTable(hashTable, pinsert);//未发生冲突的时候

}

else {

pNode\_t pCur = hashTable[hashValue];//发生冲突后

while (pCur) {

if (strcmp(pCur->pWord, pinsert->pWord) == 0) {

pinsert->tokenCode = pCur->tokenCode;

break;

}

pCur = pCur->pNext;//继续遍历这个hashTable[hashValue]

}

if (pCur == NULL) {

insertHashTable(hashTable, pinsert);//pCur为NULL之后插入到该队列中

}

}

}

void wordTailInsert(pWordNode\_t \*ppHead, pWordNode\_t\*ppTail, pNode\_t ptokenNode) {

pWordNode\_t pNew = (pWordNode\_t)calloc(1, sizeof(WordNode\_t));

pNew->ptokenNode = ptokenNode;

if (\*ppHead == NULL) {

\*ppHead = pNew;

\*ppTail = pNew;

}

else {

(\*ppTail)->pnext = pNew;

\*ppTail = pNew;

}

}

void printWordList(pWordNode\_t pHead) {

HANDLE h = GetStdHandle(STD\_OUTPUT\_HANDLE);//打印字体彩色的函数

pWordNode\_t pCur = pHead;

while (pCur) {

if (pCur->ptokenNode->tokenCode <= TK\_EOF) {

SetConsoleTextAttribute(h, FOREGROUND\_RED | FOREGROUND\_INTENSITY);

}

else if (pCur->ptokenNode->tokenCode >= KW\_CHAR && pCur->ptokenNode->tokenCode <= KW\_STDCALL) {

SetConsoleTextAttribute(h, FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

}

else if (pCur->ptokenNode->tokenCode >= TK\_CINT && pCur->ptokenNode->tokenCode <= TK\_CSTR) {

SetConsoleTextAttribute(h, FOREGROUND\_RED | FOREGROUND\_GREEN);

}

else {

SetConsoleTextAttribute(h, FOREGROUND\_INTENSITY);

}

printf("%s", pCur->ptokenNode->pWord);

pCur = pCur->pnext;

}

}

void alp2word(FILE \*fp, pNode\_t pnode) {//字符变成单词

char ch = fgetc(fp);

++i;

char\*word = (char\*)calloc(30, 1);

int wordPos = 0;

pnode->pWord = word;//wor是内存地址不是内容，之后修改的是内存的内容，这样之后修改的内容也会修改

// + += 标识符 整型常量 空白 浮点型常量

pnode->tokenCode = TK\_IDENT;

if (ch == '+') {

word[wordPos++] = ch;//先传值再加加

ch = fgetc(fp);

++i;

if (ch == '=') {

word[wordPos] = ch;//wordPos现在为2

return;

}

--i;

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

fseek(fp, 0, SEEK\_CUR);

return;

pnode->tokenCode = TK\_CSTR;

}

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

word[wordPos++] = ch;//先传值再加加，现在为0

pnode->tokenCode = -1;

return;

pnode->tokenCode = TK\_CSTR;

}

else if (ch == '\_' || ch >= 'a'&&ch <= 'z' || ch >= 'A'&&ch <= 'Z') {

word[wordPos++] = ch;

ch = fgetc(fp);

++i;

while (ch == "\_" || ch >= 'a'&&ch <= 'z' || ch >= 'A'&&ch <= 'Z' || ch >= '0'&&ch <= '9') {

word[wordPos++] = ch;

ch = fgetc(fp);

++i;

}

--i;

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

fseek(fp, 0, SEEK\_CUR);

return;

pnode->tokenCode = TK\_CSTR;

}

else if (ch >= '0'&&ch <= '9') {

word[wordPos++] = ch;

ch = fgetc(fp);

++i;

while (ch >= '0'&&ch <= '9') {

word[wordPos++] = ch;

ch = fgetc(fp);

++i;

}

if (ch == '.') {

word[wordPos++] = ch;

ch = fgetc(fp);

++i;

while (ch >= '0'&&ch <= '9') {

word[wordPos++] = ch;

ch = fgetc(fp);

++i;

}

}

--i;

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

fseek(fp, 0, SEEK\_CUR);

pnode->tokenCode = TK\_CSTR;

return;

}

else if (ch == '"') {

word[wordPos++] = ch;//先传值再加加

ch = fgetc(fp);

++i;

while ((ch == '\*') || (ch >= 'a' && ch <= 'z') || (ch == '\*') || (ch >= 'A') && (ch <= 'Z')) {

word[wordPos++] = ch;

ch = fgetc(fp);

++i;

}

--i;

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

fseek(fp, 0, SEEK\_CUR);

pnode->tokenCode = TK\_CSTR;

return;

}

else if (ch == '(') {

word[wordPos++] = ch;

pnode->tokenCode = TK\_CSTR;

return;

}

else if (ch == ')') {

word[wordPos++] = ch;

return;

}

else if (ch == '<') {

word[wordPos++] = ch;

ch = fgetc(fp);

++i;

if (ch == '=') {

word[wordPos] = ch;

return;

}

--i;

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

fseek(fp, 0, SEEK\_CUR);

pnode->tokenCode = TK\_CSTR;

return;

}

else if (ch == '>') {

word[wordPos++] = ch;

ch = fgetc(fp);

++i;

if (ch == '=') {

word[wordPos] = ch;

return;

}

--i;

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

fseek(fp, 0, SEEK\_CUR);

pnode->tokenCode = TK\_CSTR;

return;

}

else if (ch == '{') {

word[wordPos++] = ch;

pnode->tokenCode = TK\_CSTR;

return;

}

else if (ch == '}') {

word[wordPos++] = ch;

pnode->tokenCode = TK\_CSTR;

return;

}

else if (ch == '[') {

word[wordPos++] = ch;

pnode->tokenCode = TK\_CSTR;

return;

}

else if (ch == ']') {

word[wordPos++] = ch;

pnode->tokenCode = TK\_CSTR;

return;

}

else if (ch == ';') {

word[wordPos++] = ch;

pnode->tokenCode = TK\_CSTR;

return;

}

else if (ch == '.') {

word[wordPos++] = ch;

pnode->tokenCode = TK\_CSTR;

return;

}

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

word[wordPos++] = ch;

pnode->tokenCode = TK\_CSTR;

return;

}

else if (ch == '：') {

word[wordPos++] = ch;

pnode->tokenCode = TK\_CSTR;

return;

}

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

word[wordPos++] = ch;

pnode->tokenCode = TK\_CSTR;

return;

}

else if (ch == '/') {

word[wordPos++] = ch;

pnode->tokenCode = TK\_CSTR;

return;

}

else if (ch == '%') {

word[wordPos++] = ch;

pnode->tokenCode = TK\_CSTR;

return;

}

else if (ch == '！') {

word[wordPos++] = ch;

ch = fgetc(fp);

++i;

if (ch == '=') {

word[wordPos] = ch;

return;

}

--i;

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

fseek(fp, 0, SEEK\_CUR);

pnode->tokenCode = TK\_CSTR;

return;

}

else if (ch == '=') {

word[wordPos++] = ch;

ch = fgetc(fp);

++i;

if (ch == '=') {

word[wordPos] = ch;

return;

}

--i;

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

fseek(fp, 0, SEEK\_CUR);

pnode->tokenCode = TK\_CSTR;

return;

}

else if (ch == '-') {

word[wordPos++] = ch;

ch = fgetc(fp);

++i;

if (ch == '>') {

word[wordPos] = ch;

return;

}

--i;

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

fseek(fp, 0, SEEK\_CUR);

pnode->tokenCode = TK\_CSTR;

return;

}

else if (ch == '&') {

word[wordPos++] = ch;

pnode->tokenCode = TK\_CSTR;

return;

}

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

word[wordPos++] = ch;

pnode->tokenCode = TK\_CSTR;

return;

}

else if (ch == '#') {

word[wordPos++] = ch;

pnode->tokenCode = TK\_CSTR;

return;

}

}

int main() {

pNode\_t hashTable[MAXKEY] = { 0 };//初始化hashtable，hashtable是一个存储指向Node\_t的指针的数组

Node\_t preinsert[] = {//preinsert是一个存储Node\_t的数组

{TK\_PLUS,NULL,"+"},

{TK\_MINUS,NULL,"-"},

{TK\_STAR,NULL,"\*"},

{TK\_DIVIDE,NULL,"/"},

{TK\_MOD,NULL,"%"},

{TK\_EQ,NULL,"=="},

{TK\_NEQ,NULL,"!="},

{TK\_LT,NULL,"<"},

{TK\_LEQ,NULL,"<="},

{TK\_GT,NULL,">"},

{TK\_GEQ,NULL,">="},

{TK\_ASSIGN,NULL,"="},

{TK\_POINTSTO,NULL,"->"},

{TK\_DOT,NULL,"."},

{TK\_AND,NULL,"&"},

{TK\_OPENPA,NULL,"("},

{TK\_CLOSEPA,NULL,")"},

{TK\_OPENBR,NULL,"["},

{TK\_CLOSEBR,NULL,"]"},

{TK\_BEGIN,NULL,"{"},

{TK\_END,NULL,"}"},

{TK\_SEMICOLON,NULL,";"},

{TK\_COMMA,NULL,","},

{TK\_ELLIPSIS,NULL,"..."},

{TK\_EOF,NULL,"End\_Of\_File"},

{TK\_CINT,NULL,"整型常量"},

{TK\_CCHAR,NULL,"字符常量"},

{TK\_CSTR,NULL,"字符串常量"},

{KW\_CHAR,NULL,"char"},

{KW\_SHORT,NULL,"short"},

{KW\_INT,NULL,"int"},

{KW\_VOID,NULL,"void"},

{KW\_STRUCT,NULL,"struct"},

{KW\_IF,NULL,"if"},

{KW\_ELSE,NULL,"else"},

{KW\_FOR,NULL,"for"},

{KW\_CONTINUE,NULL,"continue"},

{KW\_BREAK,NULL,"break"},

{KW\_RETURN,NULL,"return"},

{KW\_SIZEOF,NULL,"sizeof"},

{KW\_ALIGN,NULL,"\_\_align"},

{KW\_CDECL,NULL,"\_\_cdecl"},

{KW\_STDCALL,NULL,"\_\_stdcall"}

};

for (int i = 0; i < sizeof(preinsert) / sizeof(Node\_t); ++i) {

insertHashTable(hashTable, &preinsert[i]);

}//初始化哈希表

pWordNode\_t pHead = NULL;

pWordNode\_t pTail = NULL;

//Node\_t node1 = { TK\_IDENT,NULL,"int" };

//findHashTable(hashTable, &node1);

//Node\_t node2 = { TK\_IDENT,NULL,"main" };

//findHashTable(hashTable, &node2);

//Node\_t node3 = { TK\_IDENT,NULL,"(" };

//findHashTable(hashTable, &node3);

//Node\_t node4 = { TK\_IDENT,NULL,")" };

//findHashTable(hashTable, &node4);

/\*char file[] = "int a += 1.2 +123";\*/

//char str1[][20] = { 0 };

//printf("请输入文件名:");

//while (scanf("%s", str1) != EOF) {}//等待要词法分析的文件名输入

///\*pNode\_t pNew1 = str1;\*/

////pNew1->pNext = NULL;

FILE \*fp;

fp = fopen("2.txt", "r+");

fseek(fp, 0, SEEK\_END);

fseek(fp, 0, SEEK\_CUR);

int len = ftell(fp);

fseek(fp, 0, SEEK\_SET);

while (i < len) {

pNode\_t pnode = (pNode\_t)calloc(1, sizeof(Node\_t));

alp2word(fp, pnode);

findHashTable(hashTable, pnode);

wordTailInsert(&pHead, &pTail, pnode);

}

//wordTailInsert(&pHead, &pTail, &node1);

//wordTailInsert(&pHead, &pTail, &node2);

//wordTailInsert(&pHead, &pTail, &node3);

//wordTailInsert(&pHead, &pTail, &node4);

printWordList(pHead);

}

# 头文件

#pragma once

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<Windows.h>

typedef struct node {

int tokenCode;

struct node \*pNext;

char \*pWord;

}Node\_t,\*pNode\_t;

/\* 单词编码 \*/

enum e\_TokenCode

{

/\* 运算符及分隔符 \*/

TK\_PLUS, // + 加号

TK\_MINUS, // - 减号

TK\_STAR, // \* 星号

TK\_DIVIDE, // / 除号

TK\_MOD, // % 求余运算符

TK\_EQ, // == 等于号

TK\_NEQ, // != 不等于号

TK\_LT, // < 小于号

TK\_LEQ, // <= 小于等于号

TK\_GT, // > 大于号

TK\_GEQ, // >= 大于等于号

TK\_ASSIGN, // = 赋值运算符

TK\_POINTSTO, // -> 指向结构体成员运算符

TK\_DOT, // . 结构体成员运算符

TK\_AND, // & 地址与运算符

TK\_OPENPA, // ( 左圆括号

TK\_CLOSEPA, // ) 右圆括号

TK\_OPENBR, // [ 左中括号

TK\_CLOSEBR, // ] 右圆括号

TK\_BEGIN, // { 左大括号

TK\_END, // } 右大括号

TK\_SEMICOLON, // ; 分号

TK\_COMMA, // , 逗号

TK\_ELLIPSIS, // ... 省略号

TK\_EOF, // 文件结束符

/\* 常量 \*/

TK\_CINT, // 整型常量

TK\_CCHAR, // 字符常量

TK\_CSTR, // 字符串常量

/\* 关键字 \*/

KW\_CHAR, // char关键字

KW\_SHORT, // short关键字

KW\_INT, // int关键字

KW\_VOID, // void关键字

KW\_STRUCT, // struct关键字

KW\_IF, // if关键字

KW\_ELSE, // else关键字

KW\_FOR, // for关键字

KW\_CONTINUE, // continue关键字

KW\_BREAK, // break关键字

KW\_RETURN, // return关键字

KW\_SIZEOF, // sizeof关键字

KW\_ALIGN, // \_\_align关键字

KW\_CDECL, // \_\_cdecl关键字 standard c call

KW\_STDCALL, // \_\_stdcall关键字 pascal c call

/\* 标识符 \*/

TK\_IDENT //函数

};

typedef struct wordNode {

pNode\_t ptokenNode;

struct wordNode\*pnext;

}WordNode\_t,\*pWordNode\_t;

int i = 0;

