编译器专题实验报告

目录

实验一:词法部分

实验内容（必做）：

【问题描述】：请根据给定的文法设计并实现词法分析程序，从源程序中识别出单词，记录其单词类别和单词值，输入输出及处理要求如下：

1. 单词类别码 单词的字符/字符串形式(中间仅用一个空格间隔)
2. 单词的类别码请统一按如下形式定义：

词法分析类别码定义如下：

单词名称	类别码	单词名称	类别码	单词名称	类别码	单词名称	类别码
Ident	IDENFR	!	NOT	*	MULT	=	ASSIGN
IntConst	INTCON	&&	AND	/	DIV	;	SEMICN
FormatString	STRCON	||	OR	%	MOD	,	COMMA
main	MAINTK	while	WHILETK	<	LSS	(	LPARENT
const	CONSTTK	getint	GETINTTK	<=	LEQ	)	RPARENT
int	INTTK	printf	PRINTFTK	>	GRE	[	LBRACK
break	BREAKTK	return	RETURNTK	>=	GEQ	]	RBRACK
continue	CONTINUETK	+	PLUS	==	EQL	{	LBRACE
if	IFTK	-	MINU	!=	NEQ	}	RBRACE
else	ELSETK	void	VOIDTK				

【输入形式】testfile.txt 中的符合文法要求的测试程序。

【输出形式】要求将词法分析结果输出至 output.txt 中。

【特别提醒】

①　读取的字符串要原样保留着便于输出，特别是数字，这里输出的并不是真正的单词值，其实是读入的字符串，单词值需另行记录。（存储时直接复制字符串来保存）

②　本次作业只考核对正确程序的处理，但需要为今后可能出现的错误情况预留接口。（错误情况目前考虑了错误的标识符和不可识别的符号）

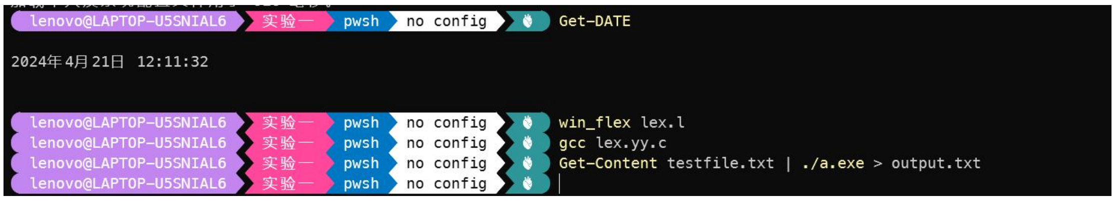
③　在今后的错误处理作业中，需要输出错误的行号，在词法分析的时候注意记录该信息。（行号用 line 来记录，设置了一个变量决定输出的时候

是否输出行号）

④ 单词的类别和单词值以及其他关注的信息，在词法分析阶段获取后，后续的分析阶段会使用，请注意记录；当前要求的输出只是为了便于评测，完成编译器中无需出现这些信息，请设计为方便打开/关闭这些输出的方案。（用一个链表来存储所有词法单元，在测试时选择用输出函数来显示，可以决定是否显示）

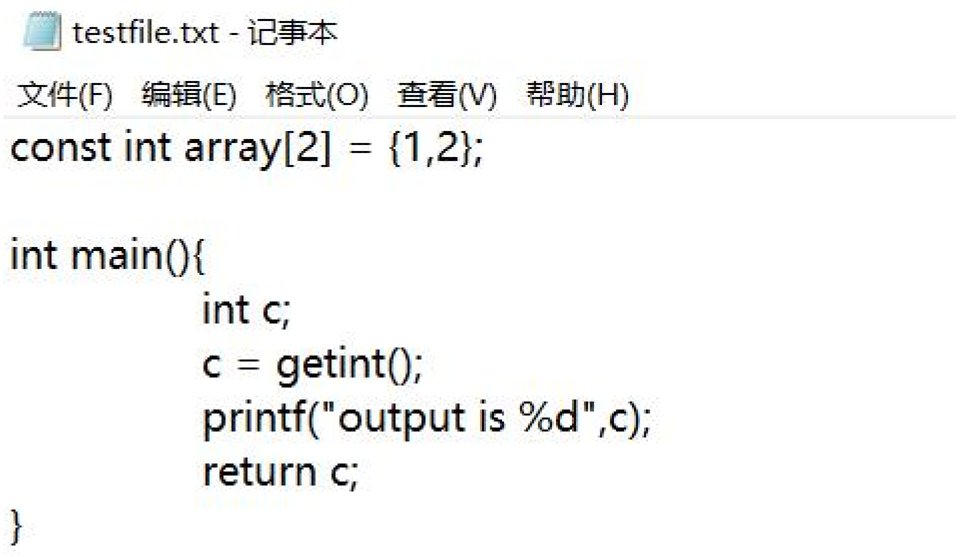
实验内容（选做）：

实验结果： （截图。实验结束截图提供实验完成时间，这一点也比较重要，截图中体现提示符，提示符揭示了自己是否独立完成，是合作模式还是独立模式等等。）：



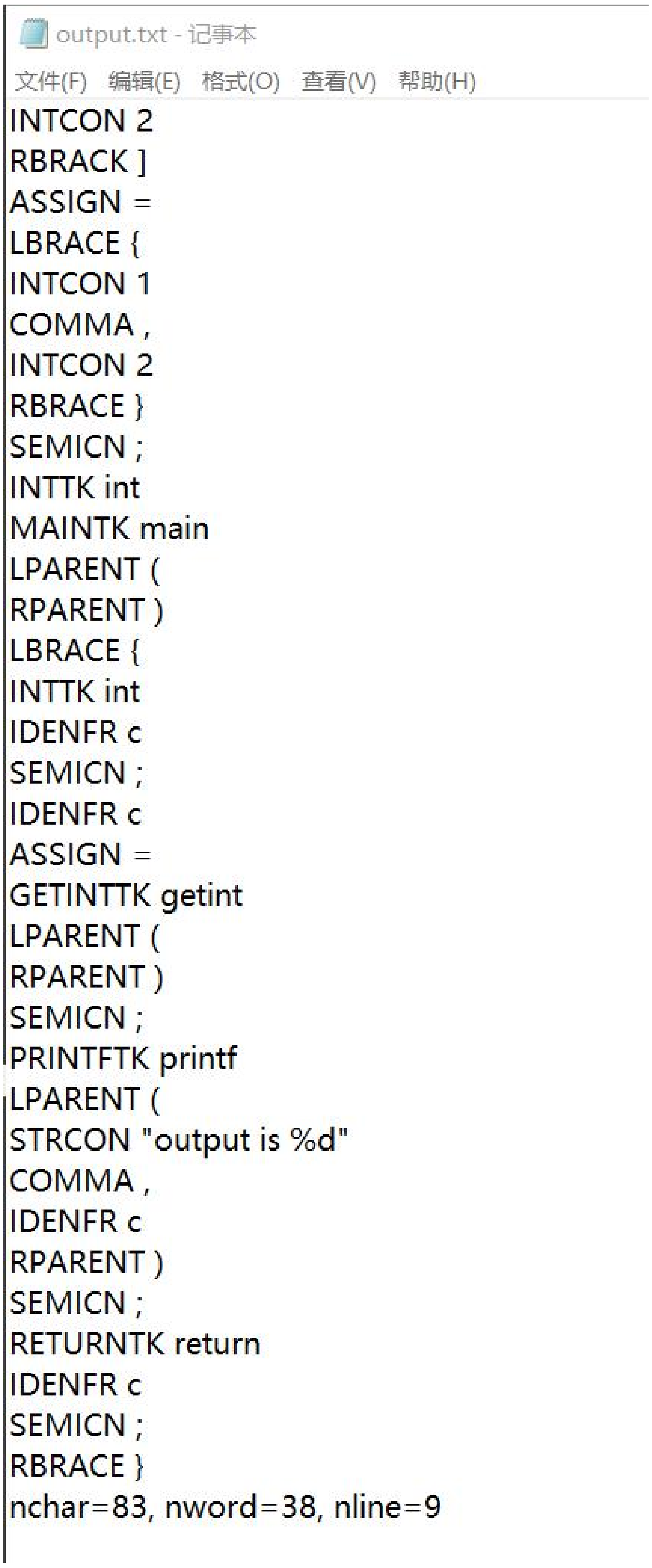
```
 lenovo@LAPTOP-U5SNIAL6  实验一  pwsh  no config   Get-DATE

2024年4月21日 12:11:32

 lenovo@LAPTOP-U5SNIAL6  实验一  pwsh  no config   win_flex lex.l
 lenovo@LAPTOP-U5SNIAL6  实验一  pwsh  no config   gcc lex.yy.c
 lenovo@LAPTOP-U5SNIAL6  实验一  pwsh  no config   Get-Content testfile.txt | ./a.exe > output.txt
 lenovo@LAPTOP-U5SNIAL6  实验一  pwsh  no config   |
```

以上是命令行输入，显示了完成实验的时间



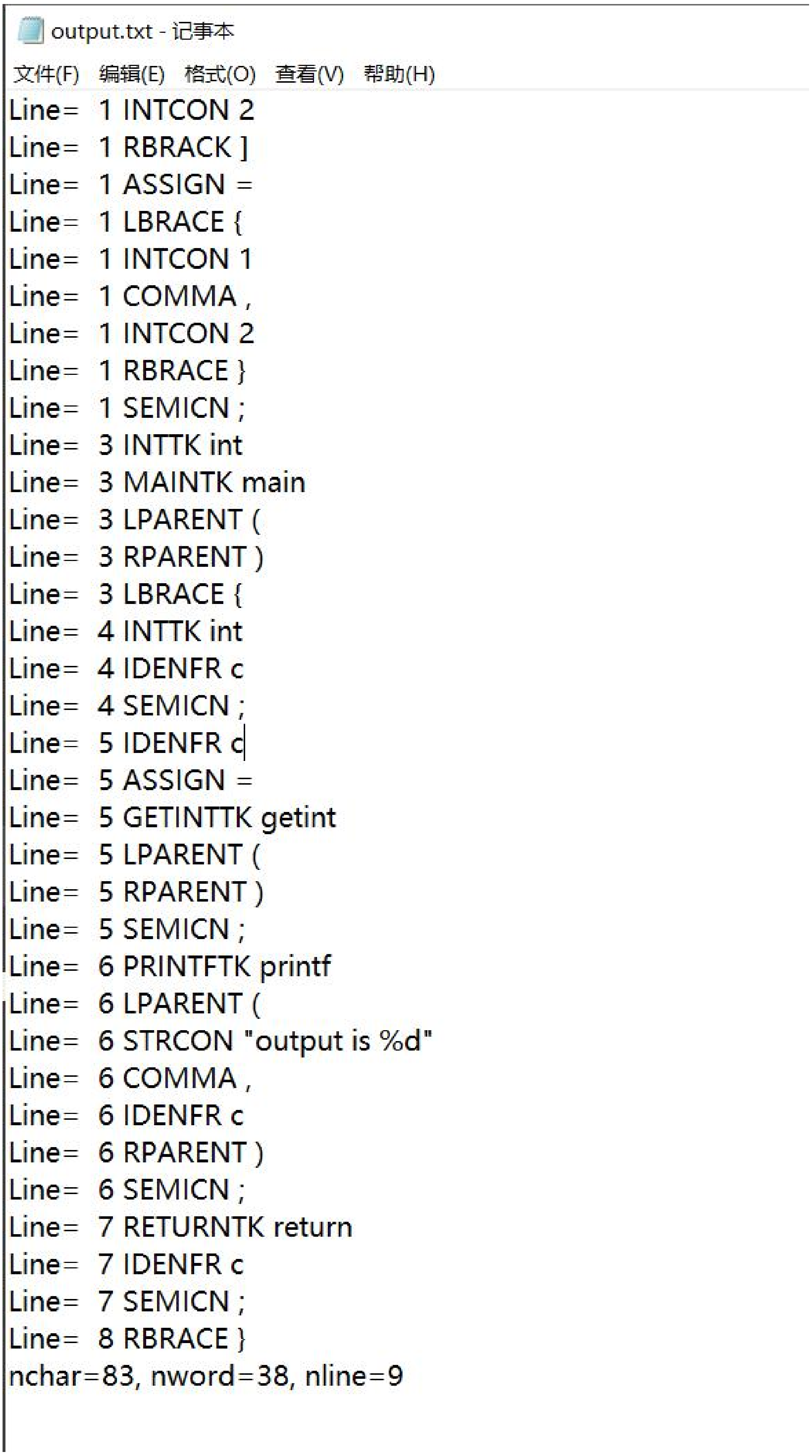
```
const int array[2] = {1,2};

int main(){
        int c;
        c = getint();
        printf("output is %d",c);
        return c;
}
```

以上是 testfile.txt 文件的内容，为实验的样例输入。



```
INTCON 2
RBRACK ]
ASSIGN =
LBRACE {
INTCON 1
COMMA ,
INTCON 2
RBRACE }
SEMICN ;
INTTK int
MAINTK main
LPARENT (
RPARENT )
LBRACE {
INTTK int
IDENFR c
SEMICN ;
IDENFR c
ASSIGN =
GETINTTK getint
LPARENT (
RPARENT )
SEMICN ;
PRINTFTK printf
LPARENT (
STRCON "output is %d"
COMMA ,
IDENFR c
RPARENT )
SEMICN ;
RETURNTK return
IDENFR c
SEMICN ;
RBRACE }
nchar=83, nword=38, nline=9
```

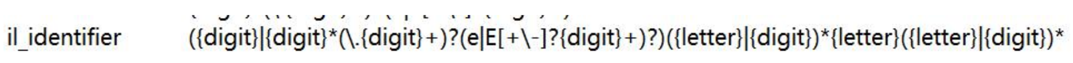
以上是实验的 output.txt 的内容，与实验样例输出完全一致。



```
Line=  1 INTCON 2
Line=  1 RBRACK ]
Line=  1 ASSIGN =
Line=  1 LBRACE {
Line=  1 INTCON 1
Line=  1 COMMA ,
Line=  1 INTCON 2
Line=  1 RBRACE }
Line=  1 SEMICN ;
Line=  3 INTTK int
Line=  3 MAINTK main
Line=  3 LPARENT (
Line=  3 RPARENT )
Line=  3 LBRACE {
Line=  4 INTTK int
Line=  4 IDENFR c
Line=  4 SEMICN ;
Line=  5 IDENFR c
Line=  5 ASSIGN =
Line=  5 GETINTTK getint
Line=  5 LPARENT (
Line=  5 RPARENT )
Line=  5 SEMICN ;
Line=  6 PRINTFTK printf
Line=  6 LPARENT (
Line=  6 STRCON "output is %d"
Line=  6 COMMA ,
Line=  6 IDENFR c
Line=  6 RPARENT )
Line=  6 SEMICN ;
Line=  7 RETURNTK return
Line=  7 IDENFR c
Line=  7 SEMICN ;
Line=  8 RBRACE }
nchar=83, nword=38, nline=9
```

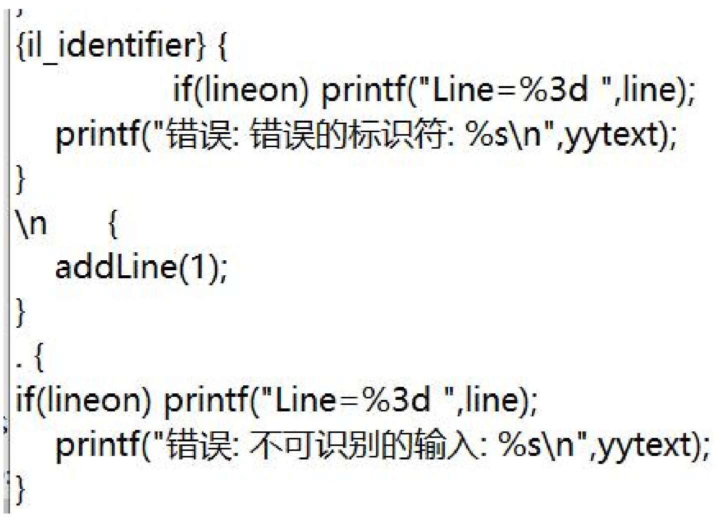
以上是输出行数的结果。

另外遇到的问题和解决思路（可选）：
解决思路：
1. 用链表来存储每一个词法单元，保存下来了具体的字符串（输入）的值。
2. 首先用正则式去定义每一个词法单元，之后再在模式匹配规则区域，去写匹配后的操作，存入链表（结构体）+调用输出函数+对 nchar、nword，nline 进行处理。
考虑的错误情况：
错误的标识符



```
il_identifier        ({digit}|{digit}*(\.{digit}+)?(e|E[+\-]?{digit}+)?)({letter}|{digit})*{letter}({letter}|{digit})*
```

错误处理情况



```
{il_identifier} {
                if(lineon) printf("Line=%3d ",line);
    printf("错误: 错误的标识符: %s\n",yytext);
}
\n      {
    addLine(1);
}
. {
if(lineon) printf("Line=%3d ",line);
    printf("错误: 不可识别的输入: %s\n",yytext);
}
```

遇到的问题：
1. 概念问题：在定义全局变量时只能定义静态变量。

代码很原创（ √ ）：
代码：

```
%{
    #include <stdio.h>
    #include <string.h>
    int lineon = 1;
    int line=1;
    int nchar,nword;
    char  token_name[50][20];
    typedef enum {
        head,
        NCHAR,
        letter,
        digit,
        IDENFR,
        INTCON,
        STRCON,
        MAINTK,
        CONSTTK,
        INTTK,
        BREAKTK,
        CONTINUETK,
        IFTK,
        ELSETK,
        WHILETK,
        GETINTTK,
        PRINTFTK,
        RETURNTK,
        PLUS,
        MINU,
        VOIDTK,
        NOT,
        AND,
        OR,
        MULT,
        DIV,
        MOD,
        LSS,
        LEQ,
        GRE,
        GEQ,
        EQL,
        NEQ,
        ASSIGN,
        SEMICN,
        COMMA,
        LPARENT,
        RPARENT,
        LBRACK,
        RBRACK,
        LBRACE,
        RBRACE,
        FLOAT,
        il_identifier
    } TokenType;

    typedef struct Token{
        TokenType type;
```

```
            char * value;
            int line;
            struct Token *next;
        } Token;
        Token * head_node;
        Token * last_node;

        Token* token_new(TokenType type, char* value, int line,Token* node);
        void token_free(Token* token);
        void token_print(Token* token);
        void addLine(int);
    %}

    line_comment    (\/\/.*\n)

    char    '[^']'
    letter  [A-Za-z_]
    digit   [0-9]
    IDENFR  {letter}({letter}|{digit})*
    INTCON  {digit}+
    STRCON   \"[^\"]*\"
    MAINTK  "main"
    CONSTTK "const"
    INTTK   "int"
    BREAKTK "break"
    CONTINUETK "continue"
    IFTK    "if"
    ELSETK  "else"
    WHILETK "while"
    GETINTTK    "getint"
    PRINTFTK    "printf"
    RETURNTK "return"
    PLUS    "+"
    MINU    "-"
    VOIDTK  "void"
    NOT "!"
    AND "&&"
    OR  "||"
    MULT    "*"
    DIV "/"
    MOD "%"
    LSS "<"
    LEQ "<="
    GRE ">"
```

```
GEQ ">="
EQL "=="
NEQ "!="
ASSIGN  "="
SEMICN  ";"
COMMA   ","
LPARENT "("
RPARENT ")"
LBRACK  "["
RBRACK  "]"
LBRACE  "{"
RBRACE  "}"
FLOAT                   {digit}*(\.{digit}+)?(e|E[+\-]?{digit}+)?
il_identifier          ({digit}|{digit}*(\.{digit}+)?(e|E[+\-]?{digit}+)?)({letter}|{digit})*{letter}({letter}|{digit})*


%%

{line_comment}  {
    if(lineon) printf("Line=%3d ",line);
    printf("line_comment %s",yytext);
    addLine(1);
}
{char}  {
    Token* Tnode =token_new(NCHAR,yytext,line,NULL);
    last_node->next = Tnode;
    last_node = Tnode;
    token_print(Tnode);
    nchar+=yyleng;
    nword++;
}
{STRCON}  {
    Token* Tnode =token_new(STRCON,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{MAINTK}   {
    Token* Tnode =token_new(MAINTK,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{CONSTTK}   {
    Token* Tnode =token_new(CONSTTK,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{INTTK}   {
    Token* Tnode =token_new(INTTK,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{BREAKTK}   {
    Token* Tnode =token_new(BREAKTK,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{CONTINUETK}   {
    Token* Tnode =token_new(CONTINUETK,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{IFTK}   {
    Token* Tnode =token_new(IFTK,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
```

```
{ELSETK}   {
    Token* Tnode =token_new(ELSETK,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{WHILETK}   {
    Token* Tnode =token_new(WHILETK,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{GETINTTK}   {
    Token* Tnode =token_new(GETINTTK,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{PRINTFTK}   {
    Token* Tnode =token_new(PRINTFTK,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{RETURNTK}   {
    Token* Tnode =token_new(RETURNTK,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{VOIDTK}   {
    Token* Tnode =token_new(VOIDTK,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{IDENFR}   {
    Token* Tnode =token_new(IDENFR,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}

{INTCON}  {
    Token* Tnode =token_new(INTCON,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{FLOAT}  {
    Token* Tnode =token_new(FLOAT,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{NOT}  {
    Token* Tnode =token_new(NOT,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{AND}  {
    Token* Tnode =token_new(AND,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
```

```
}
{OR}   {
    Token* Tnode =token_new(OR,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{PLUS}   {
    Token* Tnode =token_new(PLUS,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{MINU}   {
    Token* Tnode =token_new(MINU,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{MULT}   {
    Token* Tnode =token_new(MULT,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{DIV}   {
    Token* Tnode =token_new(DIV,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{MOD}   {
    Token* Tnode =token_new(MOD,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{LSS}   {
    Token* Tnode =token_new(LSS,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{LEQ}   {
    Token* Tnode =token_new(LEQ,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{GRE}   {
    Token* Tnode =token_new(GRE,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{GEQ}   {
    Token* Tnode =token_new(GEQ,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{EQL}   {
    Token* Tnode =token_new(EQL,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
```

```
}
{NEQ}   {
    Token* Tnode =token_new(NEQ,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{ASSIGN}   {
    Token* Tnode =token_new(ASSIGN,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{SEMICN}   {
    Token* Tnode =token_new(SEMICN,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{COMMA}   {
    Token* Tnode =token_new(COMMA,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{LPARENT}   {
    Token* Tnode =token_new(LPARENT,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{RPARENT}   {
    Token* Tnode =token_new(RPARENT,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{LBRACK}   {
    Token* Tnode =token_new(LBRACK,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{RBRACK}   {
    Token* Tnode =token_new(RBRACK,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{LBRACE}   {
    Token* Tnode =token_new(LBRACE,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{RBRACE}   {
    Token* Tnode =token_new(RBRACE,yytext,line,NULL);
    token_print(Tnode);
    last_node->next = Tnode;
    last_node = Tnode;
    nchar+=yyleng;
    nword++;
}
{il_identifier} {
    if(lineon) printf("Line=%3d ",line);
    printf("错误: 错误的标识符: %s\n",yytext);
}
\n      {
    addLine(1);
}
```

```
. {
if(lineon) printf("Line=%3d ",line);
    printf("错误: 不可识别的输入: %s\n",yytext);
}


%%
int main(void)
{
    head_node = token_new(head,"begin",0,NULL);
    last_node = head_node;

    strcpy(token_name[0], "head");
    strcpy(token_name[1], "NCHAR");
    strcpy(token_name[2], "letter");
    strcpy(token_name[3], "digit");
    strcpy(token_name[4], "IDENFR");
    strcpy(token_name[5], "INTCON");
    strcpy(token_name[6], "STRCON");
    strcpy(token_name[7], "MAINTK");
    strcpy(token_name[8], "CONSTTK");
    strcpy(token_name[9], "INTTK");
    strcpy(token_name[10], "BREAKTK");
    strcpy(token_name[11], "CONTINUETK");
    strcpy(token_name[12], "IFTK");
    strcpy(token_name[13], "ELSETK");
    strcpy(token_name[14], "WHILETK");
    strcpy(token_name[15], "GETINTTK");
    strcpy(token_name[16], "PRINTFTK");
    strcpy(token_name[17], "RETURNTK");
    strcpy(token_name[18], "PLUS");
    strcpy(token_name[19], "MINU");
    strcpy(token_name[20], "VOIDTK");
    strcpy(token_name[21], "NOT");
    strcpy(token_name[22], "AND");
    strcpy(token_name[23], "OR");
    strcpy(token_name[24], "MULT");
    strcpy(token_name[25], "DIV");
    strcpy(token_name[26], "MOD");
    strcpy(token_name[27], "LSS");
    strcpy(token_name[28], "LEQ");
    strcpy(token_name[29], "GRE");
    strcpy(token_name[30], "GEQ");
    strcpy(token_name[31], "EQL");
```

```c
    strcpy(token_name[32], "NEQ");
    strcpy(token_name[33], "ASSIGN");
    strcpy(token_name[34], "SEMICN");
    strcpy(token_name[35], "COMMA");
    strcpy(token_name[36], "LPARENT");
    strcpy(token_name[37], "RPARENT");
    strcpy(token_name[38], "LBRACK");
    strcpy(token_name[39], "RBRACK");
    strcpy(token_name[40], "LBRACE");
    strcpy(token_name[41], "RBRACE");
    strcpy(token_name[42], "FLOAT");
    strcpy(token_name[43], "il_identifier");

    yylex();
    printf("nchar=%d, nword=%d, nline=%d\n",nchar, nword, line);
    return 0;
}
int yywrap()    {
    return 1;
}

void addLine(int cnt)   {
    line += cnt;
}

Token* token_new(TokenType type, char* value, int line,Token* node) {
    Token* token = (Token*)malloc(sizeof(Token));
    token->type = type;
    token->value = strdup(value); // 使用 strdup 分配内存并复制字符串
    token->line = line;
    token->next = node;
    return token;
}

void token_free(Token* token) {
    if (token != NULL) {
        free(token->value);
        free(token);
    }
}

void token_print(Token* token) {
    if (token != NULL) {
        if(lineon) printf("Line=%3d ",token->line);
        printf("%s %s\n",
                token_name[token->type], token->value);
    }
}
```