#include<stdio.h>

#include<string.h>

#include<ctype.h>

#include<stdlib.h>

#define INT 01

#define CHAR 02

#define FLOAT 03

#define LONG 04

#define DOUBLE 05

#define STRUCT 06

#define PRINTF 07

#define SCANF 50

#define FOR 09

#define DO 10

#define WHILE 11

#define IF 12

#define ELSE 13

#define CONTINUE 14

#define BREAK 15

#define DEFAULT 16

#define CASE 17

#define GOTO 18

#define SWITCH 19

#define OB 20

#define CB 21

#define LP 22

#define RP 23

#define EQUALTO 24

#define LT 25

#define GT 26

#define PLUS 27

#define MINUS 28

#define MUL 29

#define DIV 30

#define COLON 31

#define SCOLON 32

#define TERNARY 33

#define COMMA 34

#define PLINE 35

#define CARAT 36

#define DIGIT 37

#define ID 38

#define OSB 39

#define CSB 40

#define HSH 41

#define PER 42

#define MAIN 43

#define VOID 44

#define NOTEQU 45

#define AND 46

#define BSH 47

#define DQ 48

#define SQ 49

#define FPC 51

#define DOT 52

#define SPACE 53

#define TAB 54

#define INCLUDE 55

#define CONIOH 56

#define STDIOH 57

#define CTYPEH 58

#define STRINGH 59

#define ARROW 60

#define NUM\_ENTRIES 61

int token\_no=0;

int sp=0,tb=0,delim=0,op=0,no=0,count=0,com=0,ident=0,kw=0,fn=0;

struct table

{

char name[25];

int token;

};

struct table ref\_tab[NUM\_ENTRIES] = {

{"int", INT},

{"char", CHAR},

{"float", FLOAT},

{"long", LONG},

{"double", DOUBLE},

{"struct", STRUCT},

{"printf", PRINTF},

{"scanf", SCANF},

{"do", DO},

{"while", WHILE},

{"if", IF},

{"else", ELSE},

{"continue", CONTINUE},

{"break", BREAK},

{"default", DEFAULT},

{"case", CASE},

{"goto", GOTO},

{"switch", SWITCH},

{"{", OB},

{"}", CB},

{"(", LP},

{")", RP},

{"=", EQUALTO},

{"<", LT},

{">", GT},

{"+", PLUS},

{"-", MINUS},

{"\*", MUL},

{"/", DIV},

{":", COLON},

{";", SCOLON},

{"?", TERNARY},

{",", COMMA},

{"|", PLINE},

{"^", CARAT},

{"[", OSB},

{"]", CSB},

{"#", HSH},

{"%", PER},

{"main", MAIN},

{"void", VOID},

{"!=", NOTEQU},

{"&", AND},

{"\\",BSH},

{"\"",DQ},

{"'", SQ},

{".", DOT},

{"include",INCLUDE},

{"conio.h",CONIOH},

{"stdio.h",STDIOH},

{"ctype.h",CTYPEH},

{"string.h",STRINGH},

{"->",ARROW},

};

char out\_token[NUM\_ENTRIES+1][40]=

{" ",

"KEY WORD",

"KEY WORD",

"KEY WORD",

"KEY WORD",

"KEY WORD",

"KEY WORD",

"FUNCTION",

"FUNCTION",

"KEY WORD",

"KEY WORD",

"KEY WORD",

"KEY WORD",

"KEY WORD",

"KEY WORD",

"KEY WORD",

"KEY WORD",

"KEY WORD",

"KEY WORD",

"KEY WORD",

"SPECIAL SYMBOL",

"SPECIAL SYMBOL",

"SPECIAL SYMBOL",

"SPECIAL SYMBOL",

"SPECIAL SYMBOL",

"SPECIAL SYMBOL",

"SPECIAL SYMBOL",

"1OPERATOR",

"1OPERATOR",

"1OPERATOR",

"1OPERATOR",

"SPECIAL SYMBOL",

"SPECIAL SYMBOL",

"SPECIAL SYMBOL",

"SPECIAL SYMBOL",

"SPECIAL SYMBOL",

"SPECIAL SYMBOL",

"NUMERIC CONSTANT",

"IDENTIFIER",

"SPECIAL SYMBOL",

"SPECIAL SYMBOL",

"SPECIAL SYMBOL",

"SPECIAL SYMBOL",

"FUNCTION",

"KEY WORD",

"SPECIAL SYMBOL",

"SPECIAL SYMBOL",

"SPECIAL SYMBOL",

"SPECIAL SYMBOL",

"SPECIAL SYMBOL",

"FUNCTION",

"FLOATING PIONT NUMBER",

"SPECIAL SYMBOL",

"KEY WORD",

"KEY WORD",

"KEY WORD",

"HEADER FILE",

"HEADER FILE",

"HEADER FILE",

"OPERATOR",

"OPERATOR",

};

int search(char lexeme[])

{

int i;

for ( i = 0; i < NUM\_ENTRIES; i++)

{

if (strcmp(lexeme,ref\_tab[i].name) == 0)

return ref\_tab[i].token;

}

ident++;

return ID;

}

void output(int num,char b[])

{

if(num==1||num==2||num==3||num==4||num==5||num==6||num==9||num==10||num==11||num==12||num==13||num==14||num==15||num==16||num==17||num==18||num==19||num==44||num==55)

kw++;

if(num==43||num==7||num==8)

fn++;

printf("\n%d\t\t%d\t\t%s is %s\n",++token\_no,num,b,out\_token[num]);

}

void lexical(char store[], int store\_len)

{

int i,j,k,line=2;

char ch,lexeme[10],next;

char a[100];

for (i = 0; i < store\_len;)

{

ch = store[i];

for(k=0;k<10;k++)

a[k]='\0';

k=0;

a[k++]=store[i];

a[k]='\0';

switch(ch)

{

case' ':

sp++;

i++;

printf("\n\tSPACE REMOVED\n");

break;

case'\t':

tb++;

i++;

printf("\n\tTAB REMOVED\n");

break;

case '.':

i++;

output(DOT,a);

break;

case '{':

delim++;

i++;

output(OB,a);

break;

case '}':

delim++;

i++;

output(CB,a);

break;

case '(':

delim++;

i++;

output(LP,a);

break;

case ')':

delim++;

i++;

output(RP,a);

break;

case '=':

delim++;

i++;

output(EQUALTO,a);

break;

case '<':

j=0;

i++;

output(LT,a);

// lexeme[j++]= store[i];

while((store[i])!='>')

{

lexeme[j++]= store[i];

i++;

// output(search(lexeme),lexeme);

}

lexeme[j++]='\0';

output(search(lexeme),lexeme);

// delim++;

// output(LT,a);

break;

case '>':

i++;

output(GT,a);

break;

case '+':

op++;

i++;

output(PLUS,a);

break;

case '-':

if(store[i+1]=='>')

{ i++;

a[k++]=store[i];

op++;

i++;

output(ARROW,a);

break;

}

op++;

i++;

output(MINUS,a);

break;

case '\*':

op++;

i++;

output(MUL,a);

break;

case '/':

next=store[++i];

if(next=='\*')

{

i++;

while(store[i]!='\*')i++;

i++;

if(store[i]=='/')

printf("\nComments removed\n");

com++;

i++;

break;

}

else if(next=='/')

{

i++;

while(store[i]!='\n')i++;

com++;

printf("\nComments removed\n");

break;

}

else

{

op++;

output(DIV,a);

break;

}

case ':':

delim++;

i++;

output(COLON,a);

break;

case ';':

delim++;

i++;

output(SCOLON,a);

break;

case '?':

delim++;

i++;

output(TERNARY,a);

break;

case '\"':

delim++;

i++;

output(DQ,a);

break;

case '\'':

delim++;

i++;

output(SQ,a);

break;

case ',':

delim++;

i++;

output(COMMA,a);

break;

case '|':

delim++;

i++;

output(PLINE,a);

break;

case '^':

delim++;

i++;

output(CARAT,a);

break;

case '[':

delim++;

i++;

output(OSB,a);

break;

case ']':

delim++;

i++;

output(CSB,a);

break;

case '#':

delim++;

i++;

output(HSH,a);

break;

case '%':

delim++;

i++;

output(PER,a);

break;

case '!':

delim++;

i++;

output(NOTEQU,a);

break;

case '&':

delim++;

i++;

output(AND,a);

break;

default:

if (isalpha(store[i]))

{

j = 0;

while((isalpha(store[i]))||(isdigit(store[i])))

lexeme[j++] = store[i++];

lexeme[j] = '\0';

output(search(lexeme),lexeme);

break;

}

else if(isdigit(store[i]))

{

if(isalpha(store[i+1]))

{

printf("\ninvalid identifier");

count++;

}

j = 0;

while (isdigit(store[i]))

lexeme[j++] = store[i++];

if(store[i]!='.')

{

lexeme[j] = '\0';

no++;

output(DIGIT,lexeme);

break;

}

else if(store[i]=='.'&&isdigit(store[i+1]))

{

int flag=0;

lexeme[j++]='.';

i++;

while (isdigit(store[i]))

lexeme[j++] = store[i++];

if((store[i]=='e')||(store[i]=='E'))

{

lexeme[j++]=store[i++];

if((store[i]=='+')||(store[i]=='-'))

{ if(((store[i+1])!='+')&&((store[i+1])!='-'))

{ lexeme[j++]=store[i++];

}

else { i+=2; printf("\t\t\t INVALID FLOATING POINT NO");flag=1; }

}

}

while (isdigit(store[i]))

{if(flag==0) lexeme[j++] = store[i];

i++;}

if(flag==1) break;

lexeme[j] = '\0';

no++;

output(FPC,lexeme);

break;

}

}

else if (store[i]=='\n')

{

i++;

if(store[i+1]!='\n')

{

printf("\n\nLine=%d\n",line++);

printf("\_\_\_\_\n");

}

}

else if (store[i]=='\t' || store[i]==' ')

i++;

else

i++;// printf("Invalid symbol\n");

}

}

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

a[i]='\0';

}

int main()

{

FILE \*fp1; /\* Source file pointer \*/

int ch,i; /\* used to store character read \*/

char store[4000];

int store\_len;

char file[15];

printf("\n\n\t\t\t\tLEXICAL ANALYZER\n\n");

printf("\nEnter file name with .c as the extension\n");

scanf("%s",file);

fp1 = fopen(file, "r");

if (fp1 == NULL)

{

printf("The source file cannot be opened for reading\n");

exit(0);

}

/\* read till end of file is encountered \*/

i = 0;

while ( (ch = getc(fp1)) != EOF)

{

store[i++] = ch;

}

store\_len = i;

/\* close the input file \*/

fclose(fp1);

printf("\nNUMBER\t\tTOKEN ID\tDESCRIPTION\n");

printf("\_\_\_\_\_\_\t\t\_\_\_\_\_\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n");

printf("Line=1\n");

printf("\n\_\_\_\_\n");

lexical(store, store\_len);

/\* printf("--------------------");

printf("\n\nTOTAL NO OF : \n");

printf("\nSPECIAL SYMBOLS = %d\n\nOPERATORS = %d\n\nNUMBERS = %d\n\nIDENTIFIERS = %d\n\nKEYWORDS = %d\n\nFUNCTIONS = %d\n\nSPACES REMOVED = %d\n\nTABS REMOVED = %d\n\nCOMMENTS REMOVED = %d\n",delim,op,no, ident,kw,fn,sp,tb,com);

\*/

getch();

return 0;

}