

编译原理实验一

——词法分析扫描器——

19335253 葉琚明

目录

编译原理实验一

——词法分析扫描器——

一 实验目的

二 实验要求

三 实验设计

3.1 LEX工具

3.2 C语言的单词符号及种类枚举值

3.2.1 关键字

3.2.2 标识符

3.2.3 常数

3.2.4 运算符

3.2.5 界符

3.3 其他单词符号

3.3.1 单个字符、字符串

3.3.2 C语言中主函数main、常用输出函数printf和和输入函数scanf

3.3.3 C语言头文件和宏定义识别

3.3.4 注释识别

3.3.5 未定义字符的识别

四 实验代码

4.1 LEX文件中definition部分

4.2 LEX文件中rules部分

4.3 LEX文件中user code部分

4.4 编译步骤

五 实验结果

六 实验心得

附录A

A.1 完整的definition部分:

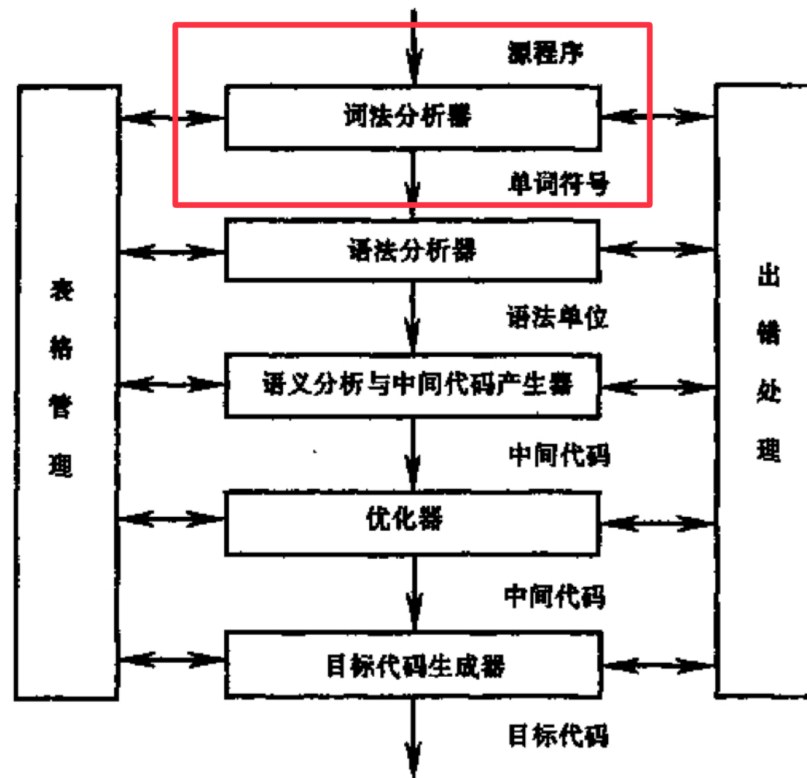
A.2 lex_ana.l代码:

A.3 demo.c文件

一 实验目的

- 设计C语言的词法分析器
- 实现词法分析器功能：输入源程序，输出单词符号
- 掌握词法规则，了解所选择编程语言单词符号及其种别值

编译程序的总体处理过程：红色框选部分为本次实验要求实现的部分。



二 实验要求

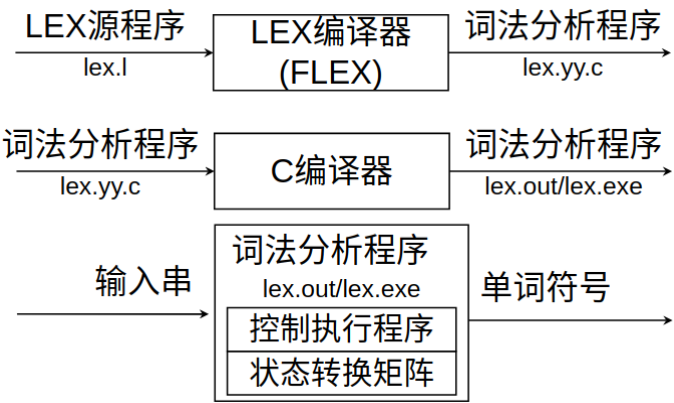
- 输入C语言源程序`demo.c`，输出一个文件`tokens.txt`，文件存有每个单词及其种类枚举值。

三 实验设计

使用LEX工具产生词法分析器

3.1 LEX工具

- LEX工具下词法分析器自动产生流程：



- LEX输入源程序构成：正规定义式、识别规则和辅助程序集或用户程序集。文件格式如下：

```
1  %{
2  /*definitions*/
3  %}
4  %%
5  /*rules*/
6  %%
7  /*user code*/
```

3.2 C语言的单词符号及种类枚举值

单词符号分为五类：关键字、标识符、常数、运算符和界符。

3.2.1 关键字

C语言有32个关键字：

- 基本数据类型（4个）： void、char、int、float、double

种类枚举值	C语言关键字
5	void
6	char
7	int
8	float
9	double

- 类型修饰关键字（4个）： short、long、unsigned、signed

种类枚举值	C语言关键字
10	short
11	long
12	unsigned
13	signed

- 复杂类型关键字（5个）： struct、union、enum、typedef、sizeof

种类枚举值	C语言关键字
14	struct
15	union
16	enum
17	typedef
18	sizeof

- 存储级别关键字（6个）： auto、static、register、extern、volatile、const

种类枚举值	C语言关键字
19	auto
20	static
21	register
22	extern
23	volatile
24	const

- 跳转结构（4个）： continue、break、goto、return

种类枚举值	C语言关键字
25	continue
26	break
27	goto
28	return

- 分支结构（5个）：if、else、switch、case、default

种类枚举值	C语言关键字
29	if
30	else
31	switch
32	case
33	default

- 循环结构（3个）：while、do、for

种类枚举值	C语言关键字
34	while
35	do
36	for

3.2.2 标识符

C语言标识符用于标识函数、变量，或用户自定义项目的名称。

一个标识符由字母A-Z或a-z或下划线（“_”）开始，后面接0个或以上的字母、数字或下划线。

则标识符id的集合可由以下正规式定义：

$$\begin{aligned} letter &\rightarrow A|B|\dots|Y|Z||a|b|\dots|y|z|_ \\ digit &\rightarrow 0|1|\dots|8|9 \\ id &\rightarrow letter(letter|digit)^* \end{aligned}$$

id的种类枚举值：

种类枚举值	C语言关键字
0	id

3.2.3 常数

考虑二进制、十进制、十六进制、浮点数和科学计数法。

二进制正规式定义： $int_bin \rightarrow 0[bB][01]^+$

十六进制正规式定义： $int_hex \rightarrow 0[xX]([A - Fa - f0 - 9])^+$

十进制、浮点数和科学计数法可以统一定义，

其正规式定义：

$$number \rightarrow [+ -]? \{digit\}^+ (\backslash. \{digit\}^+)? ([Ee][+ -]? \{digit\}^+)$$

常数的种类枚举值：

种类枚举值	C语言关键字
3	number
84	int_dec
85	ine_hex

3.2.4 运算符

C语言运算符分为算术运算符、关系运算符、逻辑运算符、赋值运算符、位运算符和其他运算符

- 算术运算符：+、-、*、/、%、++、--

种类枚举值	C语言关键字
37	+
38	-
39	*
40	/
41	%
42	++
43	--

- 关系运算符：==、!=、>、<、>=、<=

种类枚举值	C语言关键字
44	==
45	!=
46	>
47	<
48	>=
49	<=

- 逻辑运算符：&&、||、!

种类枚举值	C语言关键字
50	&&
51	
52	!

- 赋值运算符：=、+=、-=、*=、/=、%=、<<=、>>=、&=、^=、|=

种类枚举值	C语言关键字
53	=
54	+=
55	-=
56	*=
57	/=
58	%=
59	<<=
60	>>=
61	&=
62	^=
63	=

- 位运算符：&、|、^、~、<<、>>

种类枚举值	C语言关键字
64	&
65	
66	^
67	~
68	<<
69	>>

- 其他运算符：?、:、,、.、->

种类枚举值	C语言关键字
70	?
77	:
79	,
86	.
87	->

3.2.5 界符

C语言界符有：()、[]、{}、;、'、"

种类枚举值	C语言关键字
71	(
72)
73	[
74]
75	{
76	}
78	;
80	'
81	"

3.3 其他单词符号

3.3.1 单个字符、字符串

单个字符定义为' x '，有正规式定义： $cT \rightarrow \backslash'(.)\backslash'$

字符串定义为" xxx "，有正规式定义： $sT \rightarrow \backslash"(.) * \backslash"$

种类枚举值	C语言关键字
1	cT
2	sT

3.3.2 C语言中主函数main、常用输出函数printf和和输入函数scanf

main函数、printf函数、scanf函数可以按照关键字识别

种类枚举值	C语言关键字
4	main
82	printf
83	scanf

3.3.3 C语言头文件和宏定义识别

头文件的格式为：*#include < file >*，有正规式定义：*(#include)(.)**

宏定义的格式为：*#define XX xx*，有正规式定义：*(#define)(.)**

种类枚举值	C语言关键字
98	ifile
97	df

3.3.4 注释识别

C语言注释有两种：*//acomment*、*/* acomment */*

单行注释的正规式定义：*//(.)**

多行注释的正规式定义：*/*([*]*(([* /])⁺([/])^{*}))*/* */*，以注释符起始接0个或以上个“*”，再接除注释结束符外的任意字符和任意“/”，区别于单行注释，最后是注释结束符

种类枚举值	C语言关键字
96	comment

3.3.5 未定义字符的识别

除上述提及的单词符号外，剩下的归为未定义字符，以ERR标识。

其正规式定义为 ". "

种类枚举值	C语言关键字
99	ERR

四 实验代码

4.1 LEX文件中definition部分

在definition部分，定义C语言中的需要识别的单词符号的正规式：

```

1      %{
2          #include<stdio.h>
3          /*definitions of manifest constants*/
4          /* KEYWORD */
5          #define ID            0
6          #define cT            1
7          ...
8          /* OPERATION and DELIMITERS */
9          #define ADD            37
10         #define DEC            38
11         ...
12         /* OTHER */
13         #define PT            82
14         #define SCF            83
15         ...
16     %{
17
18     delim        [ \t\n]
19     ws           {delim}+
20     letter        [a-zA-Z_]
21     digit         [0-9]
22     id            {letter}({letter}|{digit})*
23     number         [+]?{digit}+(\.{digit}+)?([Ee][+-]?{digit}+)?
24     ifile         (#include)(.)*
25     df            (#define)(.)*
26     cT            \'(.)*\'
27     sT            \".)*\"
28     comment       (\"/\"|.*)|(\"/*"([*]*((^*/])+([/])*)*)*\"/*")
29     int_bin        0[bB][01]+
30     int_hex        0[xX]([A-Fa-f0-9])+

```

4.2 LEX文件中rules部分

在rules部分，按照匹配规则执行相应的动作，如：识别main成功后打印输出并返回MAIN。

```

1  %%
2  {ws}      {;}
3  main      {printf("%d      MAIN      -\n", MAIN); return MAIN;}
4  void      {printf("%d      VOID      -\n", VOID); return VOID;}
5  ...
6  "+"       {printf("%d      +          -\n", ADD); return ADD;}
7  "-"       {printf("%d      -          -\n", DEC); return DEC;}
8  ...
9  {id}      {printf("%d      ID          %s\n", ID, yytext); return ID;}
10 {cT}      {printf("%d      cT          %s\n", cT, yytext); return cT;}
11 ...

```

```

12 .          {printf("%d  ERR      %s\n", ERR, yytext); return ERR;}
13 <<EOF>>    {printf("END OF FILE.\n"); return -1;}
14 %%

```

4.3 LEX文件中user code部分

在user code部分，存放用户需要执行的C语言代码，主要编写main函数，读取C语言源文件，然后对该文件进行词法分析并输出tokens.txt文件。该部分所有代码会知已添加入到lex.yy.c文件的末尾。

如下代码，其中，函数yywrap()用于判断是否已经扫描完所有的文件，当所有的文件被扫描完时，返回1，词法分析器停止分析；函数textout()用于储存词法分析器输出的每个单词及其种类枚举值。

```

1  int yywrap (){
2      return 1;
3  }
4  void textout(int r){
5      switch(r){
6          case ID:      fprintf(yyout,"< %d  , %s >\n", ID, yytext); break;
7          case cT:      fprintf(yyout,"< %d  , %s >\n", cT, yytext); break;
8          ...
9          case IFILE:   fprintf(yyout,"< %d  , %s >\n", IFILE, yytext); break;
10         case ERR:     fprintf(yyout,"< %d  , %s >\n", ERR, yytext); break;
11         default: break;
12     }
13     return;
14 }
15 int main (int argc, char ** argv){
16     printf("TOKEN KEYWORD  TOKEN VALUE\n");
17     int c;
18     if (argc>=2){
19         if ((yyin = fopen(argv[1], "r")) == NULL){
20             printf("Can't open file %s\n", argv[1]);
21             return 1;
22         }
23         if (argc>=3){
24             yyout=fopen(argv[2], "w");
25         }
26     }
27     while (1){
28         c = yylex();
29         textout(c);
30         if(c == -1)
31             break;
32     }
33     if(argc>=2){
34         fclose(yyin);
35         if (argc>=3) fclose(yyout);
36     }
37     return 0;
38 }

```

4.4 编译步骤

```
1 $ flex -o test.yy.c test.l
2 $ gcc -o test test.yy.c -lfl
3 $ ./test demo.c tokens.txt
```

五 实验结果

测试文件为`demo.c`，其功能是简单的四则混合运算。

输入`demo.c`，词法分析器的终端输出比较详细的分析结果，`tokens.txt`存储每个单词和种类枚举值

取部分测试结果，实验结果为：

- 测试代码：

```

1  if(op_top > -1){
2      char op = ops[op_top];
3      if( (op == '*' || op == '/') && (ch == '+' || ch == '-') ){ // 乘/除优先于加/减
4          datas[++data_top].op = op;
5          --op_top;
6      }
7  }

```

- 终端输出和`tokens.txt`的输出结果:

```

C demo.c x
ip > Compilers > Code > C demo.c > mtoe(const char *)
push_num();
if(op_top > -1){
    char op = ops[op_top];
    if((op == '*' || op == '/') && (ch == '+' || ch == '-')) { /* 乘/除优先于加/减 */
        datas[++data_top].op = op;
        --op_top;
    }
}
ops[++op_top] = ch;

if(ch == ' '){
    push_num();
    while(ops[op_top] != '('){
        datas[++data_top].op = ops[op_top];
        --op_top;
    }
    --op_top;
}

nums[++num_top] = ch;

of while *tmp
a(); //最后的数据入栈
top > -1){ //最后的操作符入栈
    datas[++data_top].op = ops[op_top];
}
top;

ating(){
    top < 0) return 0;
    stack[CNT] = {0};
    i = -1;
    0;
    <= data_top){
        op = datas[i].op;
        if(op == 0){
            stack[++top] = datas[i].num;
        }
        if(a == stack[top-1];
        b = stack[top];
        op;
        if c = 0;
        op == '+') c = a + b;
        if(op == '-') c = a - b;
        if(op == '*') c = a * b;
        if(op == '/') c = a / b;
}

```

- 分析：逐个比较可以发现，词法分析器能够正确分析单词符号并输出。终端中第一列和 *tokens.txt* 的第一列表示种类枚举值，终端的第二列表示将辅助定义的标识，与种类枚举值一一

对应，终端的第三列和*tokens.txt*文件的第二列表示该种类枚举值的单词。另外，终端中输出了注释的具体内容，*tokens.txt*中没有展示具体的注释内容，注释内容对于机器来说没有处理的必要，所有也可以不进行存储操作。

六 实验心得

本次实验中，主要是实现词法分析器的功能，在此之前需要对与词法分析的相关部分如正规式，非确定自动机和确定自动机、DFA的最小化知识进行巩固并将其转化成实现代码。

本次实现借助LEX工具实现，难度不高，要求对C语言有完全的了解，比如关键字，运算符等，在实现的过程中需要不断测试不同的C语言代码，对rules部分进行完善。另外，对注释的处理是思考时间较长的一个点，对多行注释的处理，注释符含有的字符可以归纳为除注释符外的任意字符，即可把换行符等都考虑在内。

附录A

A.1 完整的definition部分：

种类枚举值	C语言关键字
0	id
1	cT
2	sT
3	number
4	main
5	void
6	char
7	int
8	float
9	double
10	short
11	long
12	unsigned
13	signed
14	struct
15	union
16	enum
17	typedef
18	sizeof
19	auto
20	static
21	register
22	extern
23	volatile
24	const
25	continue

种类枚举值	C语言关键字
26	break
27	goto
28	return
29	if
30	else
31	switch
32	case
33	default
34	while
35	do
36	for
37	+
38	-
39	*
40	/
41	%
42	++
43	--
44	==
45	!=
46	>
47	<
48	>=
49	<=
50	&&
51	
52	!
53	=
54	+=
55	-=

种类枚举值	C语言关键字
56	* =
57	/ =
58	% =
59	<<=
60	>>=
61	& =
62	^ =
63	=
64	&
65	
66	^
67	~
68	<<
69	>>
70	?
71	(
72)
73	[
74]
75	{
76	}
78	;
79	,
80	'
81	"
82	printf
83	scanf
84	int_dec
85	ine_hex
86	PNT

种类枚举值	C语言关键字
87	ARROW
96	comment
97	df
98	ifile
99	ERR

A.2 lex_ana.l代码:

```

1  %{
2      #include<stdio.h>
3      /*definitions of manifest constants*/
4      /* KEYWORD */
5      #define ID          0
6      #define cT          1
7      #define sT          2
8      #define NUMBER     3
9
10     #define MAIN        4
11     #define VOID        5
12     #define CHAR        6
13     #define INT         7
14     #define FLOAT       8
15     #define DOUBLE      9
16
17     #define SHORT       10
18     #define LONG        11
19     #define UNSG        12
20     #define SG          13
21
22     #define STRUCT      14
23     #define UNION       15
24     #define ENUM        16
25     #define TYPEDEF     17
26     #define SIZEOF      18
27
28     #define AUTO        19
29     #define STATIC      20
30     #define REGISTER    21
31     #define EXTERN      22
32     #define VOLATILE    23
33     #define CONST       24
34
35     #define CONTINUE    25
36     #define BREAK       26
37     #define GOTO        27
38     #define RETURN      28
39
40     #define IF           29
41     #define ELSE        30
42     #define SWITCH      31

```

```
43      #define CASE          32
44      #define DEFAULT      33
45
46      #define WHILE        34
47      #define DO           35
48      #define FOR          36
49
50      /* OPERATION and DELIMITERS */
51      #define ADD           37
52      #define DEC           38
53      #define MUL           39
54      #define DIV           40
55      #define MOD           41
56      #define INCR          42
57      #define DECR          43
58
59      #define EQ            44
60      #define NEQ           45
61      #define GT            46
62      #define LT            47
63      #define GE            48
64      #define LE            49
65
66      #define LOG_AND       50
67      #define LOG_OR        51
68      #define LOG_NOT       52
69
70      #define ASM           53
71      #define ADD_ASM       54
72      #define DEC_ASM       55
73      #define MUL_ASM       56
74      #define DIV_ASM       57
75      #define MOD_ASM       58
76      #define LSF_ASM       59
77      #define RSF_ASM       60
78      #define BN_AND_ASM    61
79      #define BN_OR_ASM     62
80      #define BN_XOR_ASM    63
81
82      #define BN_AND        64
83      #define BN_OR         65
84      #define BN_XOR        66
85      #define BN_REVS       67
86      #define LSF           68
87      #define RSF           69
88      #define QM            70
89
90      #define LP            71
91      #define RP            72
92      #define LBK           73
93      #define RBK           74
94      #define LBC           75
95      #define RBC           76
96
97      #define COMMA         77
98      #define SMI           78
99      #define COLON         79
100     #define SQU           80
```

```

101     #define DQU          81
102
103     /* OTHER */
104     #define PT            82
105     #define SCF           83
106
107     #define INT_BIN       84
108     #define INT_HEX       85
109
110     #define PNT           86
111     #define ARROW         87
112
113     #define COMMENT       96
114     #define DF            97
115     #define IFILE         98
116     #define ERR           99
117
118 %}
119
120 delim      [ \t\n]
121 ws         {delim}+
122 letter     [a-zA-Z_]
123 digit      [0-9]
124 id         {letter}({letter}|{digit})*
125 number     [+~]?{digit}+(\.{digit}+)?([Ee][+~]?{digit}+)?
126 ifile      (#include)(.)*
127 df         (#define)(.)*
128 cT         \'(.)\''
129 sT         \"(.)*\"
130 comment    (\"/\".*)|(\"/*\"([*]*(([^*\/])+([/])*)*)\"*/\")
131
132 int_bin    0[bB][01]+
133 int_hex    0[xX]([A-Fa-f0-9])+
134
135
136 %%
137 {ws}      {;}
138 main      {printf(\"%d      MAIN      %s\n\", MAIN, yytext); return MAIN;}
139 void      {printf(\"%d      VOID      %s\n\", VOID, yytext); return VOID;}
140 char      {printf(\"%d      CHAR      %s\n\", CHAR, yytext); return CHAR;}
141 int       {printf(\"%d      INT       %s\n\", INT, yytext); return INT;}
142 float     {printf(\"%d      FLOAT     %s\n\", FLOAT, yytext); return FLOAT;}
143 double    {printf(\"%d      DOUBLE    %s\n\", DOUBLE, yytext); return DOUBLE;}
144
145 short     {printf(\"%d      SHORT     %s\n\", SHORT, yytext); return SHORT;}
146 long      {printf(\"%d      LONG      %s\n\", LONG, yytext); return LONG;}
147 unsigned  {printf(\"%d      UNSIGNED  %s\n\", UNSG, yytext); return UNSG;}
148 signed    {printf(\"%d      SIGNED    %s\n\", SG, yytext); return SG;}
149
150 struct    {printf(\"%d      STRUCT    %s\n\", STRUCT, yytext); return STRUCT;}
151 union     {printf(\"%d      UNION     %s\n\", UNION, yytext); return UNION;}
152 enum      {printf(\"%d      ENUM      %s\n\", ENUM, yytext); return ENUM;}
153 typedef   {printf(\"%d      TYPEDEF   %s\n\", TYPEDEF, yytext); return TYPEDEF;}
154 sizeof    {printf(\"%d      SIZEOF    %s\n\", SIZEOF, yytext); return SIZEOF;}
155
156 auto      {printf(\"%d      AUTO      %s\n\", AUTO, yytext); return AUTO;}
157 static    {printf(\"%d      STATIC    %s\n\", STATIC, yytext); return STATIC;}
158 register  {printf(\"%d      REGISTER  %s\n\", REGISTER, yytext); return REGISTER;}

```

```

159 extern      {printf("%d  EXTERN  %s\n", EXTERN, yytext); return EXTERN;}
160 volatile    {printf("%d  VOLATILE %s\n", VOLATILE, yytext); return VOLATILE;}
161 const       {printf("%d  CONST   %s\n", CONST, yytext); return CONST;}
162
163 continue    {printf("%d  CONTINUE %s\n", CONTINUE, yytext); return CONTINUE;}
164 break       {printf("%d  BREAK   %s\n", BREAK, yytext); return BREAK;}
165 goto        {printf("%d  GOTO    %s\n", GOTO, yytext); return GOTO;}
166 return      {printf("%d  RETURN  %s\n", RETURN, yytext); return RETURN;}
167
168 if          {printf("%d  IF      %s\n", IF, yytext); return IF;}
169 else        {printf("%d  ELSE    %s\n", ELSE, yytext); return ELSE;}
170 switch      {printf("%d  SWITCH  %s\n", SWITCH, yytext); return SWITCH;}
171 case        {printf("%d  CASE    %s\n", CASE, yytext); return CASE;}
172 default     {printf("%d  DEFAULT %s\n", DEFAULT, yytext); return DEFAULT;}
173
174 while       {printf("%d  WHILE   %s\n", WHILE, yytext); return WHILE;}
175 do          {printf("%d  DO      %s\n", DO, yytext); return DO;}
176 for         {printf("%d  FOR     %s\n", FOR, yytext); return FOR;}
177
178
179 "+"         {printf("%d  +      %s\n", ADD, yytext); return ADD;}
180 "-"         {printf("%d  -      %s\n", DEC, yytext); return DEC;}
181 "*"         {printf("%d  *      %s\n", MUL, yytext); return MUL;}
182 "/"         {printf("%d  /      %s\n", DIV, yytext); return DIV;}
183 "%"         {printf("%d  %%     %s\n", MOD, yytext); return MOD;}
184 "++"        {printf("%d  ++     %s\n", INCR, yytext); return INCR;}
185 "--"        {printf("%d  --     %s\n", DECR, yytext); return DECR;}
186
187 "=="        {printf("%d  ==     %s\n", EQ, yytext); return EQ;}
188 "!="        {printf("%d  !=     %s\n", NEQ, yytext); return NEQ;}
189 ">"         {printf("%d  >     %s\n", GT, yytext); return GT;}
190 "<"         {printf("%d  <     %s\n", LT, yytext); return LT;}
191 ">="        {printf("%d  >=    %s\n", GE, yytext); return GE;}
192 "<="        {printf("%d  <=    %s\n", LE, yytext); return LE;}
193
194 "&&"        {printf("%d  &&     %s\n", LOG_AND, yytext); return LOG_AND;}
195 "||"        {printf("%d  ||     %s\n", LOG_OR, yytext); return LOG_OR;}
196 "!"         {printf("%d  !      %s\n", LOG_NOT, yytext); return LOG_NOT;}
197
198 "="         {printf("%d  =      %s\n", ASM, yytext); return ASM;}
199 "+="        {printf("%d  +=     %s\n", ADD_ASM, yytext); return ADD_ASM;}
200 "-="        {printf("%d  -=     %s\n", DEC_ASM, yytext); return DEC_ASM;}
201 "*="        {printf("%d  *=     %s\n", MUL_ASM, yytext); return MUL_ASM;}
202 "/="        {printf("%d  /=     %s\n", DIV_ASM, yytext); return DIV_ASM;}
203 "%%"        {printf("%d  %%=    %s\n", MOD_ASM, yytext); return MOD_ASM;}
204 "<<="        {printf("%d  <<=   %s\n", LSF_ASM, yytext); return LSF_ASM;}
205 ">>="        {printf("%d  >>=   %s\n", RSF_ASM, yytext); return RSF_ASM;}
206 "&="         {printf("%d  &=     %s\n", BN_AND_ASM, yytext); return
BN_AND_ASM;}
207 "|="         {printf("%d  |=     %s\n", BN_OR_ASM, yytext); return
BN_OR_ASM;}
208 "^="         {printf("%d  ^=     %s\n", BN_XOR_ASM, yytext); return
BN_XOR_ASM;}
209
210 "&"         {printf("%d  &      %s\n", BN_AND, yytext); return BN_AND;}
211 "|"         {printf("%d  |      %s\n", BN_OR, yytext); return BN_OR;}
212 "^"         {printf("%d  ^      %s\n", BN_XOR, yytext); return BN_XOR;}
213 "~"         {printf("%d  ~      %s\n", BN_REVS, yytext); return BN_REVS;}

```

```

214 "<<"      {printf("%d  <<      %s\n", LSF, yytext); return LSF;}
215 ">>"      {printf("%d  >>      %s\n", RSF, yytext); return RSF;}
216 "?"       {printf("%d  ?       %s\n", QM, yytext); return QM;}
217
218 "("       {printf("%d  (       %s\n", LP, yytext); return LP;}
219 ")"       {printf("%d  )       %s\n", RP, yytext); return RP;}
220 "["       {printf("%d  [       %s\n", LBK, yytext); return LBK;}
221 "]"       {printf("%d  ]       %s\n", RBK, yytext); return RBK;}
222 "{"       {printf("%d  {       %s\n", LBC, yytext); return LBC;}
223 "}"       {printf("%d  }       %s\n", RBC, yytext); return RBC;}
224
225 ","       {printf("%d  ,       %s\n", COMMA, yytext); return COMMA;}
226 ";"       {printf("%d  ;       %s\n", SMI, yytext); return SMI;}
227 ":"       {printf("%d  :       %s\n", COLON, yytext); return COLON;}
228 "'"       {printf("%d  '       %s\n", SQU, yytext); return SQU;}
229 "\"       {printf("%d  \"      %s\n", DQU, yytext); return DQU;}
230
231 "->"      {printf("%d  ->      %s\n", ARROW, yytext); return ARROW;}
232 "."       {printf("%d  .       %s\n", PNT, yytext); return PNT;}
233
234 printf     {printf("%d  PRINTF   %s\n", PT, yytext); return PT;}
235 scanf     {printf("%d  SCANF    %s\n", SCF, yytext); return SCF;}
236
237 {id}      {printf("%d  ID       %s\n", ID, yytext); return ID;}
238 {cT}      {printf("%d  cT      %s\n", cT, yytext); return cT;}
239 {sT}      {printf("%d  sT      %s\n", sT, yytext); return sT;}
240 {number}  {printf("%d  NUMBER  %s\n", NUMBER, yytext); return NUMBER;}
241
242 {int_bin} {printf("%d  INT_BIN  %s\n", INT_BIN, yytext); return INT_BIN;}
243 {int_hex} {printf("%d  INT_HEX  %s\n", INT_HEX, yytext); return INT_HEX;}
244
245 {comment} {printf("%d  COMMENT %s\n", COMMENT, yytext); return COMMENT;}
246 {df}      {printf("%d  DF       %s\n", DF, yytext); return DF;}
247 {ifile}   {printf("%d  FILE     %s\n", IFILE, yytext); return IFILE;}
248
249 .         {printf("%d  ERR      %s\n", ERR, yytext); return ERR;}
250
251 <<EOF>>   {printf("END OF FILE.\n"); return -1;}
252
253 %%
254
255
256 int yywrap (){
257     return 1;
258 }
259
260 void textout(int r){
261     switch(r){
262         case ID:      fprintf(yyout,"< %d  , %s >\n", ID, yytext); break;
263         case cT:      fprintf(yyout,"< %d  , %s >\n", cT, yytext); break;
264         case sT:      fprintf(yyout,"< %d  , %s >\n", sT, yytext); break;
265         case NUMBER:  fprintf(yyout,"< %d  , %s >\n", NUMBER, yytext); break;
266
267         case MAIN:    fprintf(yyout,"< %d  , %s >\n", MAIN, yytext); break;
268         case VOID:    fprintf(yyout,"< %d  , %s >\n", VOID, yytext); break;
269         case CHAR:    fprintf(yyout,"< %d  , %s >\n", CHAR, yytext); break;
270         case INT:     fprintf(yyout,"< %d  , %s >\n", INT, yytext); break;
271         case FLOAT:   fprintf(yyout,"< %d  , %s >\n", FLOAT, yytext); break;

```

```

272         case DOUBLE:    fprintf(yyout,"< %d , %s >\n", DOUBLE, yytext); break;
273
274         case SHORT:     fprintf(yyout,"< %d , %s >\n", SHORT, yytext); break;
275         case LONG:      fprintf(yyout,"< %d , %s >\n", LONG, yytext); break;
276         case UNSG:      fprintf(yyout,"< %d , %s >\n", UNSG, yytext); break;
277         case SG:        fprintf(yyout,"< %d , %s >\n", SG, yytext); break;
278
279         case STRUCT:    fprintf(yyout,"< %d , %s >\n", STRUCT, yytext); break;
280         case UNION:     fprintf(yyout,"< %d , %s >\n", UNION, yytext); break;
281         case ENUM:      fprintf(yyout,"< %d , %s >\n", ENUM, yytext); break;
282         case TYPEDEF:   fprintf(yyout,"< %d , %s >\n", TYPEDEF, yytext); break;
283         case SIZEOF:    fprintf(yyout,"< %d , %s >\n", SIZEOF, yytext); break;
284
285         case AUTO:      fprintf(yyout,"< %d , %s >\n", AUTO, yytext); break;
286         case STATIC:    fprintf(yyout,"< %d , %s >\n", STATIC, yytext); break;
287         case REGISTER:  fprintf(yyout,"< %d , %s >\n", REGISTER, yytext);
break;
288         case EXTERN:    fprintf(yyout,"< %d , %s >\n", EXTERN, yytext); break;
289         case VOLATILE:  fprintf(yyout,"< %d , %s >\n", VOLATILE, yytext);
break;
290         case CONST:     fprintf(yyout,"< %d , %s >\n", CONST, yytext); break;
291
292         case CONTINUE:  fprintf(yyout,"< %d , %s >\n", CONTINUE, yytext);
break;
293         case BREAK:     fprintf(yyout,"< %d , %s >\n", BREAK, yytext); break;
294         case GOTO:      fprintf(yyout,"< %d , %s >\n", GOTO, yytext); break;
295         case RETURN:    fprintf(yyout,"< %d , %s >\n", RETURN, yytext); break;
296
297         case IF:        fprintf(yyout,"< %d , %s >\n", IF, yytext); break;
298         case ELSE:      fprintf(yyout,"< %d , %s >\n", ELSE, yytext); break;
299         case SWITCH:    fprintf(yyout,"< %d , %s >\n", SWITCH, yytext); break;
300         case CASE:      fprintf(yyout,"< %d , %s >\n", CASE, yytext); break;
301         case DEFAULT:   fprintf(yyout,"< %d , %s >\n", DEFAULT, yytext); break;
302
303         case WHILE:     fprintf(yyout,"< %d , %s >\n", WHILE, yytext); break;
304         case DO:        fprintf(yyout,"< %d , %s >\n", DO, yytext); break;
305         case FOR:       fprintf(yyout,"< %d , %s >\n", FOR, yytext); break;
306
307         case ADD:       fprintf(yyout,"< %d , %s >\n", ADD, yytext); break;
308         case DEC:       fprintf(yyout,"< %d , %s >\n", DEC, yytext); break;
309         case MUL:       fprintf(yyout,"< %d , %s >\n", MUL, yytext); break;
310         case DIV:       fprintf(yyout,"< %d , %s >\n", DIV, yytext); break;
311         case MOD:       fprintf(yyout,"< %d , %s >\n", MOD, yytext); break;
312         case INCR:      fprintf(yyout,"< %d , %s >\n", INCR, yytext); break;
313         case DECR:      fprintf(yyout,"< %d , %s >\n", DECR, yytext); break;
314
315         case EQ:        fprintf(yyout,"< %d , %s >\n", EQ, yytext); break;
316         case NEQ:       fprintf(yyout,"< %d , %s >\n", NEQ, yytext); break;
317         case GT:        fprintf(yyout,"< %d , %s >\n", GT, yytext); break;
318         case LT:        fprintf(yyout,"< %d , %s >\n", LT, yytext); break;
319         case GE:        fprintf(yyout,"< %d , %s >\n", GE, yytext); break;
320         case LE:        fprintf(yyout,"< %d , %s >\n", LE, yytext); break;
321
322         case LOG_AND:   fprintf(yyout,"< %d , %s >\n", LOG_AND, yytext); break;
323         case LOG_OR:    fprintf(yyout,"< %d , %s >\n", LOG_OR, yytext); break;
324         case LOG_NOT:   fprintf(yyout,"< %d , %s >\n", LOG_NOT, yytext); break;
325
326         case ASM:       fprintf(yyout,"< %d , %s >\n", ASM, yytext); break;

```

```

327         case ADD_ASM:    fprintf(yyout,"< %d , %s >\n", ADD_ASM, yytext); break;
328         case DEC_ASM:    fprintf(yyout,"< %d , %s >\n", DEC_ASM, yytext); break;
329         case MUL_ASM:    fprintf(yyout,"< %d , %s >\n", MUL_ASM, yytext); break;
330         case DIV_ASM:    fprintf(yyout,"< %d , %s >\n", DIV_ASM, yytext); break;
331         case MOD_ASM:    fprintf(yyout,"< %d , %s >\n", MOD_ASM, yytext); break;
332         case LSF_ASM:    fprintf(yyout,"< %d , %s >\n", LSF_ASM, yytext); break;
333         case RSF_ASM:    fprintf(yyout,"< %d , %s >\n", RSF_ASM, yytext); break;
334         case BN_AND_ASM:fprintf(yyout,"< %d , %s >\n", BN_AND_ASM, yytext);
break;
335         case BN_OR_ASM:  fprintf(yyout,"< %d , %s >\n", BN_OR_ASM, yytext);
break;
336         case BN_XOR_ASM:fprintf(yyout,"< %d , %s >\n", BN_XOR_ASM, yytext);
break;
337
338         case BN_AND:     fprintf(yyout,"< %d , %s >\n", BN_AND, yytext); break;
339         case BN_OR:      fprintf(yyout,"< %d , %s >\n", BN_OR, yytext); break;
340         case BN_XOR:     fprintf(yyout,"< %d , %s >\n", BN_XOR, yytext); break;
341         case BN_REVS:    fprintf(yyout,"< %d , %s >\n", BN_REVS, yytext); break;
342         case LSF:        fprintf(yyout,"< %d , %s >\n", LSF, yytext); break;
343         case RSF:        fprintf(yyout,"< %d , %s >\n", RSF, yytext); break;
344         case QM:         fprintf(yyout,"< %d , %s >\n", QM, yytext); break;
345
346         case LP:         fprintf(yyout,"< %d , %s >\n", LP, yytext); break;
347         case RP:         fprintf(yyout,"< %d , %s >\n", RP, yytext); break;
348         case LBK:        fprintf(yyout,"< %d , %s >\n", LBK, yytext); break;
349         case RBK:        fprintf(yyout,"< %d , %s >\n", RBK, yytext); break;
350         case LBC:        fprintf(yyout,"< %d , %s >\n", LBC, yytext); break;
351         case RBC:        fprintf(yyout,"< %d , %s >\n", RBC, yytext); break;
352
353         case COMMA:      fprintf(yyout,"< %d , %s >\n", COMMA, yytext); break;
354         case SMI:        fprintf(yyout,"< %d , %s >\n", SMI, yytext); break;
355         case COLON:      fprintf(yyout,"< %d , %s >\n", COLON, yytext); break;
356         case SQU:        fprintf(yyout,"< %d , %s >\n", SQU, yytext); break;
357         case DQU:        fprintf(yyout,"< %d , %s >\n", DQU, yytext); break;
358
359         case PT:         fprintf(yyout,"< %d , %s >\n", PT, yytext); break;
360         case SCF:        fprintf(yyout,"< %d , %s >\n", SCF, yytext); break;
361
362         case INT_BIN:    fprintf(yyout,"< %d , %s >\n", INT_BIN, yytext); break;
363         case INT_HEX:    fprintf(yyout,"< %d , %s >\n", INT_HEX, yytext); break;
364
365         case PNT:        fprintf(yyout,"< %d , %s >\n", PNT, yytext); break;
366         case ARROW:      fprintf(yyout,"< %d , %s >\n", ARROW, yytext); break;
367
368         case COMMENT:    fprintf(yyout,"< %d , COMMENT >\n", COMMENT); break;
369         case DF:         fprintf(yyout,"< %d , %s >\n", DF, yytext); break;
370         case IFILE:      fprintf(yyout,"< %d , %s >\n", IFILE, yytext); break;
371         case ERR:        fprintf(yyout,"< %d , %s >\n", ERR, yytext); break;
372
373         default: break;
374     }
375     return;
376 }
377
378 int main (int argc, char ** argv){
379
380     printf("TOKEN KEYWORD    TOKEN VALUE\n");
381     int c;

```

```

382     if (argc>=2){
383         if ((yyin = fopen(argv[1], "r")) == NULL){
384             printf("Can't open file %s\n", argv[1]);
385             return 1;
386         }
387         if (argc>=3){
388             yyout=fopen(argv[2], "w");
389         }
390     }
391
392     while (1){
393         c = yylex();
394         textout(c);
395         if(c == -1)
396             break;
397     }
398
399     if(argc>=2){
400         fclose(yyin);
401         if (argc>=3) fclose(yyout);
402     }
403     return 0;
404
405 }
406

```

A.3 demo.c文件

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define CNT 1000
6
7  struct node{
8      float num;
9      char op;
10 };
11
12 struct node datas[CNT]; //后缀表达式栈
13 int data_top = -1;      //栈顶索引
14
15 char ops[CNT];          //操作符栈
16 int op_top = -1;        //栈顶索引
17
18 char nums[100];         //数字字符串
19 int num_top = -1;       //栈顶索引
20 //数字入栈
21 void push_num(){
22     if(num_top > -1){
23         datas[++data_top].num = atof(nums);
24         datas[data_top].op = 0;
25         num_top = -1;
26         memset(nums, 0, sizeof(nums));
27     }
28 }

```



```

29 //中缀转后缀
30 void mtoe(const char* str){
31     char *tmp;
32     tmp = (char*)str;
33     while(*tmp != '\0'){
34         char ch = *tmp;
35         ++tmp;
36         if(ch == ' ') continue;
37
38         if(ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '('){
39             push_num();
40             if(op_top > -1){
41                 char op = ops[op_top];
42                 if( (op == '*' || op == '/') && (ch == '+' || ch == '-') ){// 乘/除
                    优先于加/减
43                     datas[++data_top].op = op;
44                     --op_top;
45                 }
46             }
47             ops[++op_top] = ch;
48         }
49         else if(ch == ')'){
50             push_num();
51             while(ops[op_top] != '('){
52                 datas[++data_top].op = ops[op_top];
53                 --op_top;
54             }
55             --op_top;
56         }
57         else{
58             nums[++num_top] = ch;
59         }
60     } // end of while *tmp
61     push_num();//最后的数据入栈
62     while(op_top > -1){ //最后的操作符入栈
63         datas[++data_top].op = ops[op_top];
64         --op_top;
65     }
66
67 }
68 //计算值
69 float calculating(){
70     if(data_top < 0) return 0;
71     float stack[CNT] = {0};
72     int top = -1;
73     int i = 0;
74     while(i <= data_top){
75         char op = datas[i].op;
76         if(op == 0){
77             stack[++top] = datas[i].num;
78         }else{
79             float a = stack[top - 1];
80             float b = stack[top];
81             --top;
82             float c = 0;
83             if(op == '+') c = a + b;
84             else if(op == '-') c = a - b;
85             else if(op == '*') c = a * b;

```

```
86         else if(op == '/') c = a / b;
87         stack[top] = c;
88     }
89     ++i;
90 }
91 if(top < 0) return 0;
92 else return stack[top];
93 }
94
95 int main(int argc, char *argv[]){
96     char *parms = "20.5+(100-(3+2)*8)/(8-5) - 10";
97     memset(datas, 0, sizeof(datas));
98     memset(ops, 0, sizeof(ops));
99     data_top = -1;
100    op_top = -1;
101    num_top = -1;
102    mtoe(parms);
103    printf("%s = ",parms);
104    printf("%f\n",calculating());
105    int i = 0;
106    for(i = 0; i <= data_top; i++){
107        if(datas[i].op) printf("%c ", datas[i].op);
108        else printf("%f ",datas[i].num);
109    }
110    printf("\n");
111    return 0;
112 }
```