词法分析实验报告

一.实验内容及要求

1. 可以识别出用 C 语言编写的源程序中的每个单词符号，并以记号的形式输出每个单词符号。
2. 可以识别并跳过源程序中的注释。
3. 可以统计源程序中的语句行数、各类单词的个数、以及字符总数，并输出统计结果。
4. 检查源程序中存在的词法错误，并报告错误所在的位置。
5. 对源程序中出现的错误进行适当的恢复，使词法分析可以继续进行，对源程序进行一次扫描，即可检查并报告源程序中存在的所有词法错误。

实现方法要求：分别用以下两种方法实现。

方法 1：采用 C/C++作为实现语言，手工编写词法分析程序。（必做）

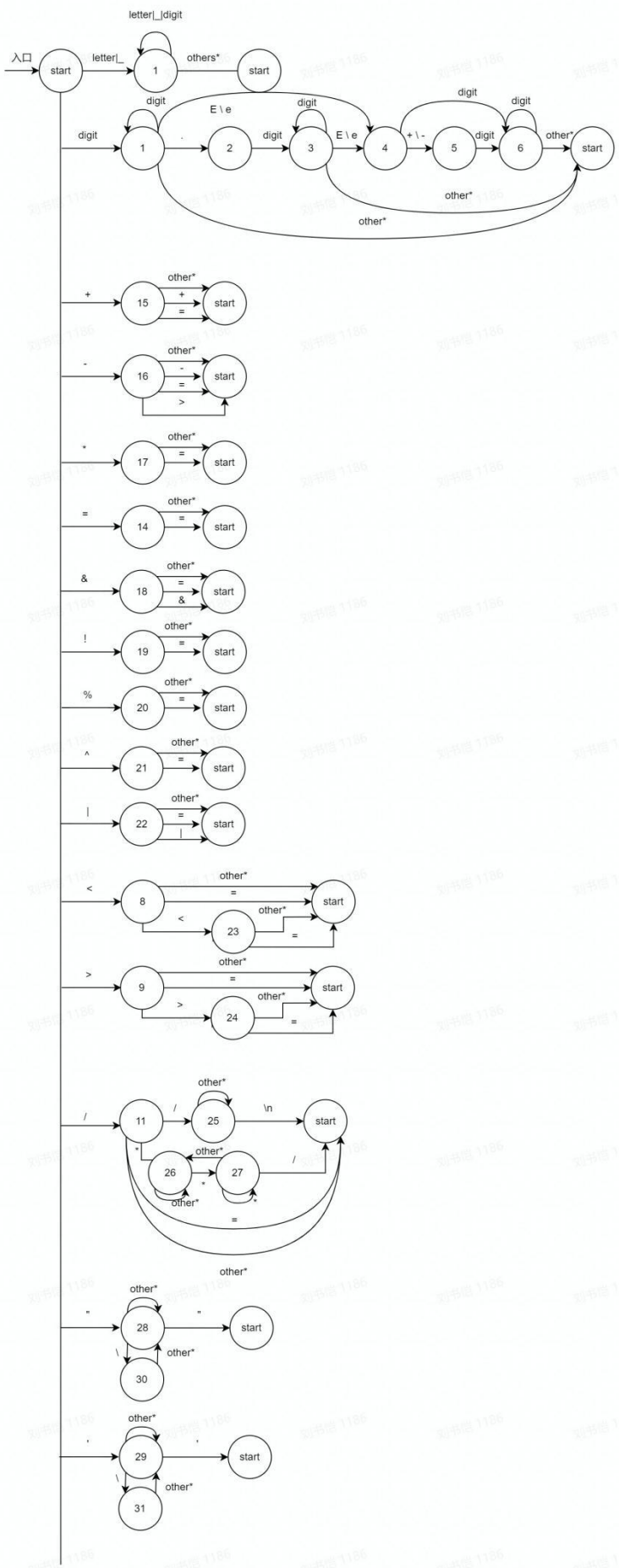
方法 2：编写 LEX 源程序，利用 LEX 编译程序自动生成词法分析程序。

二.程序设计说明

2.1 设计思路与状态转换图

词法分析的主体是识别各种各样的单词，识别单词是按照记号的模式进行的。为了设计词法分析程序,需要对模式给出规范、系统的精确说明。正规表达式和正规文法是描述模式的重要工具。但是 C/C++语言本身并不支持对正则文法的识别，所以我们需要将正则文法转化为状态转换图，并用编程语言对其进行描述，才可以实现词法分析的功能，下面我们将分析识别 C 语言各种记号的状态转换图应如何构建。

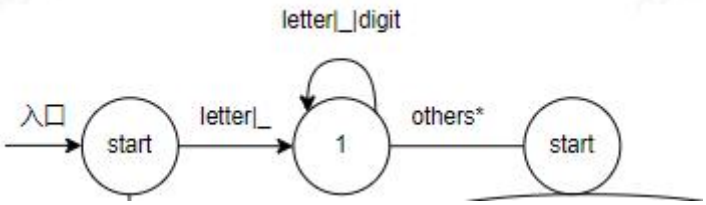
完整的状态转换图


入口
start
letter|_
1
letter|_|digit
others*
start
digit
E \ e
2
.
3
4
+ \ -
5
6
other*
other*
other*
15
16
17
14
18
19
20
21
22
8
23
9
24
11
25
26
27
28
30
29
31
\n
"
'
\

标识符

C 语言标识符由字母【A-Z,a-z】、数字【0-9】、下划线“_”组成，并且首字符不能是数字，但可以是字母或者下划线，构建的状态转换图如下所示

其中 letter 表示字母【A-Z,a-z】，digit 表示数字【0-9】



保留字

C 语言共包含 32 个保留字，展示如下表

auto	double	int	struct	break	else	static	long
switch	case	enum	register	typedef	char	extern	return
union	const	float	short	unsigned	continue	for	signed
void	default	goto	sizeof	volatile	do	if	while

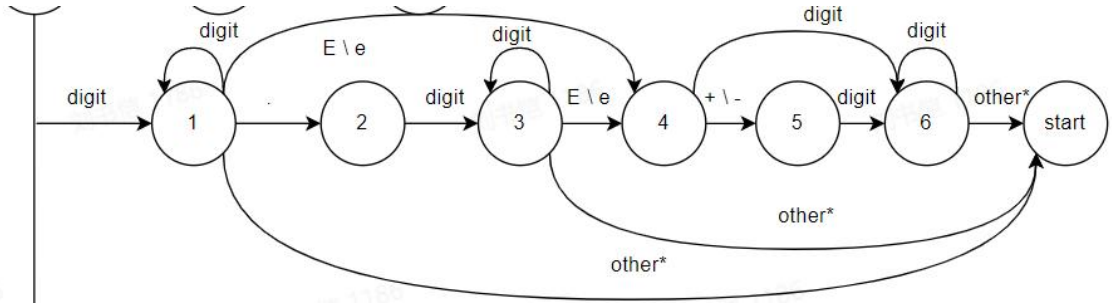
在程序里，我们将其存入了一个字符串的数组，每当我们的程序读入一个单词（{letter}+）之后，我们会先在字符串数组内查找这个单词，如果可以找到，那么这个单词就是一个保留字，反之则为一个标识符（因为全部为字母，符合标识符的定义）

无符号数

无符号数的语法规则基本与 Pascal 语言一致，这里我们给出其正则表达式（LEX 形式）

```
Python
{digit}+(\.{digit}+)?([Ee][+-]?{digit}+)?
```

其状态转换图如下所示

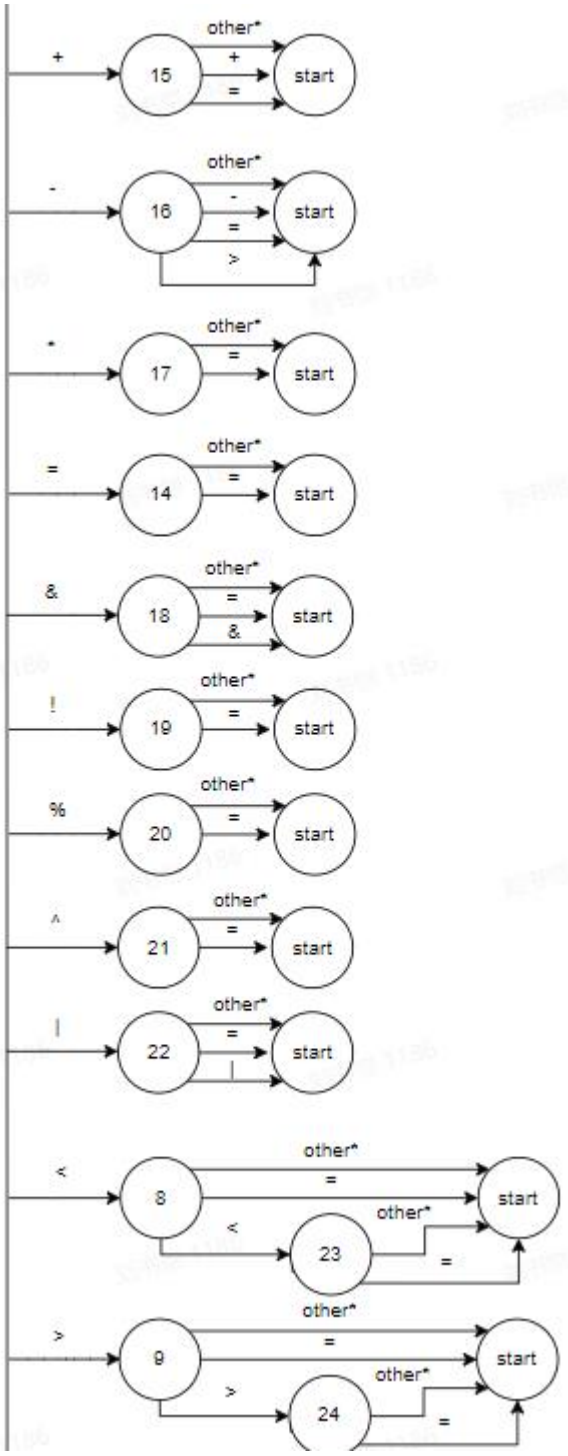


运算符

C 语言共涉及如下运算符

算术运算符	+	-	*	/	%	++	--
关系运算符	==	!=	>	<	>=	<=	
逻辑运算符	&&	||	!				
位运算符	&	|	^	~	<<	>>	
赋值运算符	=	+=	-=	*=	/=	%=	<<=
	>>=	&=	^=	|=			
指针运算符	.	->					

其中只有指针运算符.不需要超前读入，当我们读入'.'后可以理解做出判断，其他运算符均需要超前读入，下面将给出他们的状态转换图。



注：因为/还涉及到注释的识别，在这里先不作介绍，状态转换图将在注释识别部分给出。

分隔符

C 语言共涉及如下分隔符

;	,	:	.	#	[	]	{
}	(	)					

这些分隔符均不存在需要超前读入进行识别的情况，读入后即可直接识别，因此此处便不再给出状态转换图进行说明。

注释识别

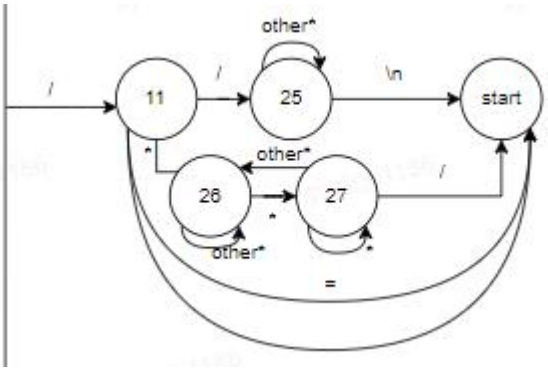
在 C 语言中存在两种不同的注释方法，一种为单行注释，形如

```
//this is a note
```

另一种为多行注释，形如

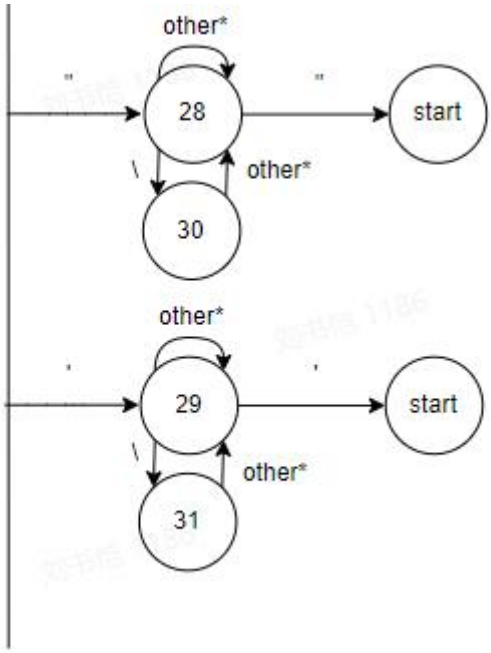
```
/*
this is also a note
*/
```

两种注释方法有一点共同之处：都是以'/'开头，而以'/'开头的单词还包括除法等一系列操作，于是我们对其稍作整理，便可得到以下状态转换图



字符串与字符常量的处理

在 C 语言中，字符串常量通常是通过配对的"进行标记，一般来说，我们只需要在识别到一个"之后，一直向后进行超前搜索，找到下一个"即可完成对字符串的识别。但是这种识别方式存在一种问题：字符串中如果存在\"的转移字符，会导致字符串错误地提前被截断。因此我们需要在识别到'\'符号时，越过其下一个符号的判断，才可以防止字符串错误地被提前截断。对字符变量的处理同理。下面给出两者的状态转换图



错误处理

本程序对出现的错误处理较为简单。当进入 error 状态后，程序会输出错误出现的位置，并且当前读到的字符 C 与字符串 token 进行输出，以便用户排查错误。最后回到 0 状态（即 start 状态），重新识别后面的单词

2.2 利用 C++实现

利用 C++进行词法分析的主体结构就是将上述所有状态转换图用 C++语言进行实现，主要依托于 switch 语句，与课本上给出参考程序结构基本一致，使用的全局变量与函数含义也基本一致。但课本上的参考程序采用的是类 C 语言描述，我们使用 C++语言进行实现必然会用到一些新的语法特性对程序结构进行重构，下面将讲述本程序的部分模块划分以及对象设计。

统计类的设计

程序设计要求我们可以统计源程序中的语句行数、各类单词的个数、以及字符总数，并输出统计结果，于是我们封装一个统计类，以便对各种信息进行统计

```C++
class Statistics
{
public:
        Statistics();
        ~Statistics();
        int my_line = 0;//程序的总行数
```

```
        long long num_char = 0;//程序中单词的总数
        int num_id = 0;//下面为各类单词数目的统计
        int num_num = 0;
        int num_integer = 0;
        int num_keyword = 0;
        int num_operater = 0;
        int num_single = 0;
        int num_my_string = 0;
        int num_my_char = 0;
        int num_error = 0;
        void print_num() ;
private:
};
```

二元组的设计

程序设计要求我们对识别出来的单词以二元组的形式进行输出,于是我们封装一个二元组的类，以便在输出时使用

```
C++
class Binary
{
public:
        Binary ();
        ~Binary ();
        Mark sign;//单词的记号（类别）
        long long Integer;//存储为整数
        double Unsigned_number;//存储为无符号数
        string word="";//存储为字符串
private:
};
```

其中 Mark 变量为定义的一个枚举类，包含了程序中所有可能出现的单词的记号

```
C++
enum Mark
{
        id,//标识符
        num,//无符号数
        integer,//整数
        keyword,//保留字
        operater,//运算符
```

```
        pound_sign,//分隔符
        my_string,//字符串常量
        my_char//字符常量
};
```

双缓冲区的设计

为了提高从磁盘中读取文件的效率，我们没有选择一次文件中读取一个字符的方法，而是选择一次从文件中读取 1kb 数据并利用双缓冲区进行存储。缓冲区为一个字符串数组 buffer，大小为 2048 字节。forward 为前向指针，具体处理逻辑如下:

```
C++
C = buffer[::forward];//取出当前字符
        if (C == '\n')
        {
                my_statis.my_line++;//行数+1
        }
        if (::forward == 1023)
        {
                if (!is_retract)/*这里为处理技巧 因为程序中存在将指针回退的操作
                                        如果在读取了文件后回退，会导致重复读入
                                        所以我们设计一个标志位来判断是否因为回退而到达了该位置*/
                {
                        myread.read_behind();//读取文件存储到后半缓冲区
                }
                ::forward++;
                is_retract = 0;
        }
        else if (::forward == 2047)
        {
                if (!is_retract)
                {
                        myread.read_front();//读取文件存储到前半缓冲区
                }
                ::forward=0;
                is_retract = 0;
        }
        else
```

```
        {
                ::forward++;
                is_retract = 0;
        }
```

2.3 利用 LEX 实现

利用 LEX 实现词法分析与利用 C++实现词法分析有所不同，C++需要我们依据状态转化图编写识别程序，而 LEX 只需要我们给出正则表达式即可自动生成识别其的程序，所以利用 LEX 进行词法分析的关键就是给出对应记号的正则表达式：

```
digit ([0-9])//数字

letter ([A-Za-z_])//字符与下划线

LINE_NOTES (\/\/[^\n]*)//单行注释

MUL_LINE_NOTES (\/\*([^\*]|(\*)*[^\*/])*(\*)*\*\/)//多行注释

STRING (\"([^\\\"\n]|\\.)*\")//字符串

CHAR (\'([^\'\\]|\\.)*\')//字符

KEYWORD
"auto"|"double"|"int"|"struct"|"break"|"else"|"long"|"switch"|"case"
|"enum"|"register"|"typedef"|
"char"|"extern"|"return"|"union"|"const"|"float"|"short"|"unsigned"|
"continue"|"for"|"signed"|"void"|"default"|"goto"|"sizeof"|"volatile
"|
"do"|"if"|"while"|"static"//保留字

IDENTIFIER ({letter}({letter}|{digit})*)//标识符

OPERATOR "+"|"."|"->"|"-"|"++"|"--"|"*"|"&"|"!"|"~"|"/"|"%"|"<<"|
">>"|">"|"<"|">="|"<="|"=="|"!="|"^"|"|"|"&&"|"||"|"="|"/="|"*="|
"%="|"+="|"-="|"<<="|">>="|"&="|"^="|"|="|":"|"?"//操作符

INTEGER ({digit}+)//整数

UNSIGNED_NUM ({digit}+(\.{digit}+)?([Ee][+-]?{digit}+)?)//无符号数

POUND_SIGN "("|")"|"["|"]"|"{"|"}"|"#"|";"|","//分隔符
```

上述部分全部位于 LEX 程序的第一部分。在第二部分中我们只要识别到了上述某一记号，就可以执行特定的代码段，以完成我们输出、统计需求，这部分与 C 语言差别不大，在此不作赘述，程序运行结果如下：

```
line146:(POUND_SIGN, })

------------------------------------------------------------

Total Word: 784
Total Line: 146
Total KEYWORD: 80
Total IDENTIFIER: 228
Total OPERATOR: 103
Total POUND_SIGN: 327
Total INTEGER: 27
Total UNSIGNED_NUM: 3
Total STRING: 15
Total CHAR: 1
```

三.测试报告

输入

```
C
#include<stdio.h>

int main(){
    printf("Hello world\n");//这是一个注释
    /*这是一个多行注释
      #include<stdio.h>
    */
    //整数与无符号数测试
    2.5E+1;
    1;
    2e1;
    2e-1;
    //运算符测试+分隔符测试
    (a+b);
    [a-b];
    {a<=b}
    a<b;
```

```
    a>b;
    a>=b
    a=b;
    a==b;
    a+=1;
    a*b;
    a*=b;
    a&b;
    a&&b;
    a&=b;
    !a
    a!=b;
    a%b;
    a%=b;
    a^b;
    a^=b;
    a|b;
    a||b;
    a,b;
    //字符串常量与字符常量测试
    "this is a \"string";
    '\'';
    //一段 C 语言程序测试
        struct trie_node *new_node(int level)
{
   struct trie_node *p = (struct trie_node *)malloc(sizeof(struct
trie_node));
   memset(p, 0, sizeof(struct trie_node));
   p->level = level + 1;
   return p;
}
// 增加节点
void trie_add(struct trie_node *node, unsigned char *pattern)
{
   unsigned int cnt = pattern[0];
   if (node->type[cnt] == TRIR_CHILD_TYPE_NULL)
   {
      if (cnt)
         set_child_str(node, cnt, pattern);
      else
         set_child_str(node, cnt, NULL);
      return;
   }
   else if (node->type[cnt] == TRIR_CHILD_TYPE_NODE)
```

```
   {
      trie_add((struct trie_node *)(node->child[cnt]), pattern + 1);
      return;
   }
   else if (node->type[cnt] == TRIR_CHILD_TYPE_LEAF) //分裂问题
   {
      if (node->child[cnt] == NULL)
      return;
      unsigned char *leaf = (unsigned char *)(node->child[cnt]);
      struct trie_node *child_node = new_node(node->level);
      add_child_node(node, cnt, child_node);
      trie_add(child_node, leaf + 1);
      trie_add(child_node, pattern + 1);
      return;
   }
}
BOOL trie_find(struct trie_node *node, unsigned char *p)
{
   int index = p[0];
   if (node->type[index] == TRIR_CHILD_TYPE_NULL)
   return FALSE;
   else if (node->type[index] == TRIR_CHILD_TYPE_LEAF)
   {
      if (node->child[index] == 0 && p[0] == 0)
         return TRUE;
      else if (node->child[index] != 0 && p[0] != 0) //都不为0，则判
断是否相同
         return strcmp((char *)(p), (char *)(node->child[index])) ==
0;
      else
         return FALSE;
   }
   return trie_find((node->child[index]), p + 1);
}
int main()
{
   FILE *f = fopen("pattern-gbk", "rt");
   if (f == NULL)
   {
      printf("can not open file 'pattern'\n");
      return 1;
   }
   struct trie_node *root = new_node(0);
   char buffer[256];
```

```
    while (fgets(buffer, sizeof(buffer), f) != NULL)
    {
        char *p = trim_str(buffer);
        if (*p == 0)
        continue;
        trie_add(root, (unsigned char *)strdup(p));
    }
    fclose(f);
    FILE *fData = fopen("input-gbk", "rt");
    if (fData == NULL)
    {
        printf("can not open file 'input'\n");
        return 1;
    }
    FILE *fwrite = fopen("yipipei.txt", "w");
    if (fwrite == NULL)
    {
        printf("can not open file 'pattern'\n");
        return 1;
    }
    int index = 0;
    int count = 0;
    while (fgets(buffer, sizeof(buffer), fData) != NULL)
    {
        index++;
        char *p = trim_str(buffer);
        if (*p == 0)
        continue;
        if (trie_find(root, (unsigned char *)p))
        {
            count++;
            printf("%d: %s yes\n", index, p);
            fputs(p,fwrite);
            fprintf(fwrite,"\n");
        }
        else
        printf("%d: %s no\n", index, p);
    }
    printf("read %d lines, found %d\n", index, count);
    fclose(fData);
    fclose(fwrite);
    return 0;
}
}
```

输入用例说明:

输入用例尽可能遍历了所有可能出现的单词类型，以确保词法分析器的正确性。另外如字符串或字符变量，我们也测试了其中还有转义字符或者\"的情况，作了部分边界测试。此外在对所有可能出现单词类型进行测试后，我们在后面附加了一段 Trie 树的实现代码，一方面是为了测试实际情况下词法分析器是否可以得到预期的输出，另一方面由于输入量较大，也测试了我们双缓冲区机制是否可以正常工作。

输出

C++实现

```
          Type                    Value
line:1    pound_sign              #
line:1    id                      include
line:1    operator                <
line:1    id                      stdio
line:1    operator                .
line:1    id                      h
line:1    operator                >
line:3    keyword                 int
line:3    id                      main
line:3    pound_sign              (
line:3    pound_sign              )
line:3    pound_sign              {
line:4    id                      printf
line:4    pound_sign              (
line:4    string                  " Hello world\n"
line:4    pound_sign              )
line:4    pound_sign              ;
line:9    Unsigned number         2.5E+1
line:9    pound_sign              ;
line:10   integer                 1
line:10   pound_sign              ;
line:11   Unsigned number         2e1
line:11   pound_sign              ;
line:12   Unsigned number         2e-1
line:12   pound_sign              ;
line:14   pound_sign              (
line:14   id                      a
line:14   operator                +
line:14   id                      b
line:14   pound_sign              )
```

```
line:14   pound_sign             ;
line:15   pound_sign             [
line:15   id                     a
line:15   operator               -
line:15   id                     b
line:15   pound_sign             ]
line:15   pound_sign             ;
line:16   pound_sign             {
line:16   id                     a
line:16   operator               <=
line:16   id                     b
line:16   pound_sign             }
line:17   id                     a
line:17   operator               <
line:17   id                     b
line:17   pound_sign             ;
line:18   id                     a
line:18   operator               >
line:18   id                     b
line:18   pound_sign             ;
line:19   id                     a
line:19   operator               >=
line:19   id                     b
line:20   id                     a
line:20   operator               =
line:20   id                     b
line:20   pound_sign             ;
line:21   id                     a
line:21   operator               ==
line:21   id                     b
line:21   pound_sign             ;
line:22   id                     a
line:22   operator               +=
line:22   integer                1
line:22   pound_sign             ;
line:23   id                     a
line:23   operator               *
line:23   id                     b
line:23   pound_sign             ;
line:24   id                     a
line:24   operator               *=
line:24   id                     b
line:24   pound_sign             ;
line:25   id                     a
```

```
line:25   operator              &
line:25   id                    b
line:25   pound_sign            ;
line:26   id                    a
line:26   operator              &&
line:26   id                    b
line:26   pound_sign            ;
line:27   id                    a
line:27   operator              &=
line:27   id                    b
line:27   pound_sign            ;
line:28   operator              !
line:28   id                    a
line:29   id                    a
line:29   operator              !=
line:29   id                    b
line:29   pound_sign            ;
line:30   id                    a
line:30   operator              %
line:30   id                    b
line:30   pound_sign            ;
line:31   id                    a
line:31   operator              %=
line:31   id                    b
line:31   pound_sign            ;
line:32   id                    a
line:32   operator              ^
line:32   id                    b
line:32   pound_sign            ;
line:33   id                    a
line:33   operator              ^=
line:33   id                    b
line:33   pound_sign            ;
line:34   id                    a
line:34   operator              |
line:34   id                    b
line:34   pound_sign            ;
line:35   id                    a
line:35   operator              ||
line:35   id                    b
line:35   pound_sign            ;
line:36   id                    a
line:36   pound_sign            ,
line:36   id                    b
```

```
line:36   pound_sign             ;
line:38   string                 " this is a \"string"
line:38   pound_sign             ;
line:39   char                   ' \''
line:39   pound_sign             ;
line:41   keyword                struct
line:41   id                     trie_node
line:41   operator               *
line:41   id                     new_node
line:41   pound_sign             (
line:41   keyword                int
line:41   id                     level
line:41   pound_sign             )
line:42   pound_sign             {
line:43   keyword                struct
line:43   id                     trie_node
line:43   operator               *
line:43   id                     p
line:43   operator               =
line:43   pound_sign             (
line:43   keyword                struct
line:43   id                     trie_node
line:43   operator               *
line:43   pound_sign             )
line:43   id                     malloc
line:43   pound_sign             (
line:43   keyword                sizeof
line:43   pound_sign             (
line:43   keyword                struct
line:43   id                     trie_node
line:43   pound_sign             )
line:43   pound_sign             )
line:43   pound_sign             ;
line:44   id                     memset
line:44   pound_sign             (
line:44   id                     p
line:44   pound_sign             ,
line:44   integer                0
line:44   pound_sign             ,
line:44   keyword                sizeof
line:44   pound_sign             (
line:44   keyword                struct
line:44   id                     trie_node
line:44   pound_sign             )
```

```
line:44   pound_sign             )
line:44   pound_sign             ;
line:45   id                     p
line:45   operator               ->
line:45   id                     level
line:45   operator               =
line:45   id                     level
line:45   operator               +
line:45   integer                1
line:45   pound_sign             ;
line:46   keyword                return
line:46   id                     p
line:46   pound_sign             ;
line:47   pound_sign             }
line:49   keyword                void
line:49   id                     trie_add
line:49   pound_sign             (
line:49   keyword                struct
line:49   id                     trie_node
line:49   operator               *
line:49   id                     node
line:49   pound_sign             ,
line:49   keyword                unsigned
line:49   keyword                char
line:49   operator               *
line:49   id                     pattern
line:49   pound_sign             )
line:50   pound_sign             {
line:51   keyword                unsigned
line:51   keyword                int
line:51   id                     cnt
line:51   operator               =
line:51   id                     pattern
line:51   pound_sign             [
line:51   integer                0
line:51   pound_sign             ]
line:51   pound_sign             ;
line:52   keyword                if
line:52   pound_sign             (
line:52   id                     node
line:52   operator               ->
line:52   id                     type
line:52   pound_sign             [
line:52   id                     cnt
```

```
line:52   pound_sign             ]
line:52   operator               ==
line:52   id                     TRIR_CHILD_TYPE_NULL
line:52   pound_sign             )
line:53   pound_sign             {
line:54   keyword                if
line:54   pound_sign             (
line:54   id                     cnt
line:54   pound_sign             )
line:55   id                     set_child_str
line:55   pound_sign             (
line:55   id                     node
line:55   pound_sign             ,
line:55   id                     cnt
line:55   pound_sign             ,
line:55   id                     pattern
line:55   pound_sign             )
line:55   pound_sign             ;
line:56   keyword                else
line:57   id                     set_child_str
line:57   pound_sign             (
line:57   id                     node
line:57   pound_sign             ,
line:57   id                     cnt
line:57   pound_sign             ,
line:57   id                     NULL
line:57   pound_sign             )
line:57   pound_sign             ;
line:58   keyword                return
line:58   pound_sign             ;
line:59   pound_sign             }
line:60   keyword                else
line:60   keyword                if
line:60   pound_sign             (
line:60   id                     node
line:60   operator               ->
line:60   id                     type
line:60   pound_sign             [
line:60   id                     cnt
line:60   pound_sign             ]
line:60   operator               ==
line:60   id                     TRIR_CHILD_TYPE_NODE
line:60   pound_sign             )
line:61   pound_sign             {
```

```
line:62   id                    trie_add
line:62   pound_sign            (
line:62   pound_sign            (
line:62   keyword               struct
line:62   id                    trie_node
line:62   operator              *
line:62   pound_sign            )
line:62   pound_sign            (
line:62   id                    node
line:62   operator              ->
line:62   id                    child
line:62   pound_sign            [
line:62   id                    cnt
line:62   pound_sign            ]
line:62   pound_sign            )
line:62   pound_sign            ,
line:62   id                    pattern
line:62   operator              +
line:62   integer               1
line:62   pound_sign            )
line:62   pound_sign            ;
line:63   keyword               return
line:63   pound_sign            ;
line:64   pound_sign            }
line:65   keyword               else
line:65   keyword               if
line:65   pound_sign            (
line:65   id                    node
line:65   operator              ->
line:65   id                    type
line:65   pound_sign            [
line:65   id                    cnt
line:65   pound_sign            ]
line:65   operator              ==
line:65   id                    TRIR_CHILD_TYPE_LEAF
line:65   pound_sign            )
line:66   pound_sign            {
line:67   keyword               if
line:67   pound_sign            (
line:67   id                    node
line:67   operator              ->
line:67   id                    child
line:67   pound_sign            [
line:67   id                    cnt
```

```
line:67   pound_sign             ]
line:67   operator               ==
line:67   id                     NULL
line:67   pound_sign             )
line:68   keyword                return
line:68   pound_sign             ;
line:69   keyword                unsigned
line:69   keyword                char
line:69   operator               *
line:69   id                     leaf
line:69   operator               =
line:69   pound_sign             (
line:69   keyword                unsigned
line:69   keyword                char
line:69   operator               *
line:69   pound_sign             )
line:69   pound_sign             (
line:69   id                     node
line:69   operator               ->
line:69   id                     child
line:69   pound_sign             [
line:69   id                     cnt
line:69   pound_sign             ]
line:69   pound_sign             )
line:69   pound_sign             ;
line:70   keyword                struct
line:70   id                     trie_node
line:70   operator               *
line:70   id                     child_node
line:70   operator               =
line:70   id                     new_node
line:70   pound_sign             (
line:70   id                     node
line:70   operator               ->
line:70   id                     level
line:70   pound_sign             )
line:70   pound_sign             ;
line:71   id                     add_child_node
line:71   pound_sign             (
line:71   id                     node
line:71   pound_sign             ,
line:71   id                     cnt
line:71   pound_sign             ,
line:71   id                     child_node
```

```
line:71   pound_sign             )
line:71   pound_sign             ;
line:72   id                     trie_add
line:72   pound_sign             (
line:72   id                     child_node
line:72   pound_sign             ,
line:72   id                     leaf
line:72   operator               +
line:72   integer                1
line:72   pound_sign             )
line:72   pound_sign             ;
line:73   id                     trie_add
line:73   pound_sign             (
line:73   id                     child_node
line:73   pound_sign             ,
line:73   id                     pattern
line:73   operator               +
line:73   integer                1
line:73   pound_sign             )
line:73   pound_sign             ;
line:74   keyword                return
line:74   pound_sign             ;
line:75   pound_sign             }
line:76   pound_sign             }
line:77   id                     BOOL
line:77   id                     trie_find
line:77   pound_sign             (
line:77   keyword                struct
line:77   id                     trie_node
line:77   operator               *
line:77   id                     node
line:77   pound_sign             ,
line:77   keyword                unsigned
line:77   keyword                char
line:77   operator               *
line:77   id                     p
line:77   pound_sign             )
line:78   pound_sign             {
line:79   keyword                int
line:79   id                     index
line:79   operator               =
line:79   id                     p
line:79   pound_sign             [
line:79   integer                0
```

```
line:79   pound_sign             ]
line:79   pound_sign             ;
line:80   keyword                if
line:80   pound_sign             (
line:80   id                     node
line:80   operator               ->
line:80   id                     type
line:80   pound_sign             [
line:80   id                     index
line:80   pound_sign             ]
line:80   operator               ==
line:80   id                     TRIR_CHILD_TYPE_NULL
line:80   pound_sign             )
line:81   keyword                return
line:81   id                     FALSE
line:81   pound_sign             ;
line:82   keyword                else
line:82   keyword                if
line:82   pound_sign             (
line:82   id                     node
line:82   operator               ->
line:82   id                     type
line:82   pound_sign             [
line:82   id                     index
line:82   pound_sign             ]
line:82   operator               ==
line:82   id                     TRIR_CHILD_TYPE_LEAF
line:82   pound_sign             )
line:83   pound_sign             {
line:84   keyword                if
line:84   pound_sign             (
line:84   id                     node
line:84   operator               ->
line:84   id                     child
line:84   pound_sign             [
line:84   id                     index
line:84   pound_sign             ]
line:84   operator               ==
line:84   integer                0
line:84   operator               &&
line:84   id                     p
line:84   pound_sign             [
line:84   integer                0
line:84   pound_sign             ]
```

```
line:84   operator                ==
line:84   integer                 0
line:84   pound_sign              )
line:85   keyword                 return
line:85   id                      TRUE
line:85   pound_sign              ;
line:86   keyword                 else
line:86   keyword                 if
line:86   pound_sign              (
line:86   id                      node
line:86   operator                ->
line:86   id                      child
line:86   pound_sign              [
line:86   id                      index
line:86   pound_sign              ]
line:86   operator                !=
line:86   integer                 0
line:86   operator                &&
line:86   id                      p
line:86   pound_sign              [
line:86   integer                 0
line:86   pound_sign              ]
line:86   operator                !=
line:86   integer                 0
line:86   pound_sign              )
line:87   keyword                 return
line:87   id                      strcmp
line:87   pound_sign              (
line:87   pound_sign              (
line:87   keyword                 char
line:87   operator                *
line:87   pound_sign              )
line:87   pound_sign              (
line:87   id                      p
line:87   pound_sign              )
line:87   pound_sign              ,
line:87   pound_sign              (
line:87   keyword                 char
line:87   operator                *
line:87   pound_sign              )
line:87   pound_sign              (
line:87   id                      node
line:87   operator                ->
line:87   id                      child
```

```
line:87   pound_sign             [
line:87   id                     index
line:87   pound_sign             ]
line:87   pound_sign             )
line:87   pound_sign             )
line:87   operator               ==
line:87   integer                0
line:87   pound_sign             ;
line:88   keyword                else
line:89   keyword                return
line:89   id                     FALSE
line:89   pound_sign             ;
line:90   pound_sign             }
line:91   keyword                return
line:91   id                     trie_find
line:91   pound_sign             (
line:91   pound_sign             (
line:91   id                     node
line:91   operator               ->
line:91   id                     child
line:91   pound_sign             [
line:91   id                     index
line:91   pound_sign             ]
line:91   pound_sign             )
line:91   pound_sign             ,
line:91   id                     p
line:91   operator               +
line:91   integer                1
line:91   pound_sign             )
line:91   pound_sign             ;
line:92   pound_sign             }
line:93   keyword                int
line:93   id                     main
line:93   pound_sign             (
line:93   pound_sign             )
line:94   pound_sign             {
line:95   id                     FILE
line:95   operator               *
line:95   id                     f
line:95   operator               =
line:95   id                     fopen
line:95   pound_sign             (
line:95   string                 " pattern-gbk"
line:95   pound_sign             ,
```

```
line:95   string                  " rt"
line:95   pound_sign              )
line:95   pound_sign              ;
line:96   keyword                 if
line:96   pound_sign              (
line:96   id                      f
line:96   operator                ==
line:96   id                      NULL
line:96   pound_sign              )
line:97   pound_sign              {
line:98   id                      printf
line:98   pound_sign              (
line:98   string                  " can not open file 'pattern'\n"
line:98   pound_sign              )
line:98   pound_sign              ;
line:99   keyword                 return
line:99   integer                 1
line:99   pound_sign              ;
line:100  pound_sign              }
line:101  keyword                 struct
line:101  id                      trie_node
line:101  operator                *
line:101  id                      root
line:101  operator                =
line:101  id                      new_node
line:101  pound_sign              (
line:101  integer                 0
line:101  pound_sign              )
line:101  pound_sign              ;
line:102  keyword                 char
line:102  id                      buffer
line:102  pound_sign              [
line:102  integer                 256
line:102  pound_sign              ]
line:102  pound_sign              ;
line:103  keyword                 while
line:103  pound_sign              (
line:103  id                      fgets
line:103  pound_sign              (
line:103  id                      buffer
line:103  pound_sign              ,
line:103  keyword                 sizeof
line:103  pound_sign              (
line:103  id                      buffer
```

```
line:103  pound_sign              )
line:103  pound_sign              ,
line:103  id                      f
line:103  pound_sign              )
line:103  operator                !=
line:103  id                      NULL
line:103  pound_sign              )
line:104  pound_sign              {
line:105  keyword                 char
line:105  operator                *
line:105  id                      p
line:105  operator                =
line:105  id                      trim_str
line:105  pound_sign              (
line:105  id                      buffer
line:105  pound_sign              )
line:105  pound_sign              ;
line:106  keyword                 if
line:106  pound_sign              (
line:106  operator                *
line:106  id                      p
line:106  operator                ==
line:106  integer                 0
line:106  pound_sign              )
line:107  keyword                 continue
line:107  pound_sign              ;
line:108  id                      trie_add
line:108  pound_sign              (
line:108  id                      root
line:108  pound_sign              ,
line:108  pound_sign              (
line:108  keyword                 unsigned
line:108  keyword                 char
line:108  operator                *
line:108  pound_sign              )
line:108  id                      strdup
line:108  pound_sign              (
line:108  id                      p
line:108  pound_sign              )
line:108  pound_sign              )
line:108  pound_sign              ;
line:109  pound_sign              }
line:110  id                      fclose
line:110  pound_sign              (
```

```
line:110  id                    f
line:110  pound_sign            )
line:110  pound_sign            ;
line:111  id                    FILE
line:111  operator              *
line:111  id                    fData
line:111  operator              =
line:111  id                    fopen
line:111  pound_sign            (
line:111  string                " input-gbk"
line:111  pound_sign            ,
line:111  string                " rt"
line:111  pound_sign            )
line:111  pound_sign            ;
line:112  keyword               if
line:112  pound_sign            (
line:112  id                    fData
line:112  operator              ==
line:112  id                    NULL
line:112  pound_sign            )
line:113  pound_sign            {
line:114  id                    printf
line:114  pound_sign            (
line:114  string                " can not open file 'input'\n"
line:114  pound_sign            )
line:114  pound_sign            ;
line:115  keyword               return
line:115  integer               1
line:115  pound_sign            ;
line:116  pound_sign            }
line:117  id                    FILE
line:117  operator              *
line:117  id                    fwrite
line:117  operator              =
line:117  id                    fopen
line:117  pound_sign            (
line:117  string                " yipipei.txt"
line:117  pound_sign            ,
line:117  string                " w"
line:117  pound_sign            )
line:117  pound_sign            ;
line:118  keyword               if
line:118  pound_sign            (
line:118  id                    fwrite
```

```
line:118  operator                ==
line:118  id                      NULL
line:118  pound_sign              )
line:119  pound_sign              {
line:120  id                      printf
line:120  pound_sign              (
line:120  string                  " can not open file 'pattern'\n"
line:120  pound_sign              )
line:120  pound_sign              ;
line:121  keyword                 return
line:121  integer                 1
line:121  pound_sign              ;
line:122  pound_sign              }
line:123  keyword                 int
line:123  id                      index
line:123  operator                =
line:123  integer                 0
line:123  pound_sign              ;
line:124  keyword                 int
line:124  id                      count
line:124  operator                =
line:124  integer                 0
line:124  pound_sign              ;
line:125  keyword                 while
line:125  pound_sign              (
line:125  id                      fgets
line:125  pound_sign              (
line:125  id                      buffer
line:125  pound_sign              ,
line:125  keyword                 sizeof
line:125  pound_sign              (
line:125  id                      buffer
line:125  pound_sign              )
line:125  pound_sign              ,
line:125  id                      fData
line:125  pound_sign              )
line:125  operator                !=
line:125  id                      NULL
line:125  pound_sign              )
line:126  pound_sign              {
line:127  id                      index
line:127  operator                ++
line:127  pound_sign              ;
line:128  keyword                 char
```

```
line:128  operator              *
line:128  id                    p
line:128  operator              =
line:128  id                    trim_str
line:128  pound_sign            (
line:128  id                    buffer
line:128  pound_sign            )
line:128  pound_sign            ;
line:129  keyword               if
line:129  pound_sign            (
line:129  operator              *
line:129  id                    p
line:129  operator              ==
line:129  integer               0
line:129  pound_sign            )
line:130  keyword               continue
line:130  pound_sign            ;
line:131  keyword               if
line:131  pound_sign            (
line:131  id                    trie_find
line:131  pound_sign            (
line:131  id                    root
line:131  pound_sign            ,
line:131  pound_sign            (
line:131  keyword               unsigned
line:131  keyword               char
line:131  operator              *
line:131  pound_sign            )
line:131  id                    p
line:131  pound_sign            )
line:131  pound_sign            )
line:132  pound_sign            {
line:133  id                    count
line:133  operator              ++
line:133  pound_sign            ;
line:134  id                    printf
line:134  pound_sign            (
line:134  string                " %d: %s yes\n"
line:134  pound_sign            ,
line:134  id                    index
line:134  pound_sign            ,
line:134  id                    p
line:134  pound_sign            )
line:134  pound_sign            ;
```

```
line:135  id                    fputs
line:135  pound_sign            (
line:135  id                    p
line:135  pound_sign            ,
line:135  id                    fwrite
line:135  pound_sign            )
line:135  pound_sign            ;
line:136  id                    fprintf
line:136  pound_sign            (
line:136  id                    fwrite
line:136  pound_sign            ,
line:136  string                " \n"
line:136  pound_sign            )
line:136  pound_sign            ;
line:137  pound_sign            }
line:138  keyword               else
line:139  id                    printf
line:139  pound_sign            (
line:139  string                " %d: %s no\n"
line:139  pound_sign            ,
line:139  id                    index
line:139  pound_sign            ,
line:139  id                    p
line:139  pound_sign            )
line:139  pound_sign            ;
line:140  pound_sign            }
line:141  id                    printf
line:141  pound_sign            (
line:141  string                " read %d lines, found %d\n"
line:141  pound_sign            ,
line:141  id                    index
line:141  pound_sign            ,
line:141  id                    count
line:141  pound_sign            )
line:141  pound_sign            ;
line:142  id                    fclose
line:142  pound_sign            (
line:142  id                    fData
line:142  pound_sign            )
line:142  pound_sign            ;
line:143  id                    fclose
line:143  pound_sign            (
line:143  id                    fwrite
line:143  pound_sign            )
```

```
line:143  pound_sign              ;
line:144  keyword                 return
line:144  integer                 0
line:144  pound_sign              ;
line:145  pound_sign              }
line:146  pound_sign              }
total_lines:             146
total_chars:             784
total_errors:            0
num of id:               228
num of integer:          27
num of Unsigned number:  3
num of keyword:          80
num of operater:         103
num of pound_sign:       327
num of string:           15
num of char:             1
```

lex 输出

```
line1:(POUND_SIGN, #)

line1:(IDENTIFIER, include)

line1:(OPERATOR, <)

line1:(IDENTIFIER, stdio)

line1:(OPERATOR, .)

line1:(IDENTIFIER, h)

line1:(OPERATOR, >)

line3:(KEYWORD, int)

line3:(IDENTIFIER, main)

line3:(POUND_SIGN, ()

line3:(POUND_SIGN, ))
```

```
line3:(POUND_SIGN, {)

line4:(IDENTIFIER, printf)

line4:(POUND_SIGN, ()

line4:(STRING, "Hello world\n")

line4:(POUND_SIGN, ))

line4:(POUND_SIGN, ;)

line4:(LINE_NOTES, //这是一个注释)

line5:(MUL_LINE_NOTES, /*这是一个多行注释
      #include<stdio.h>
    */)

line8:(LINE_NOTES, //整数与无符号数测试)

line9:(UNSIGNED_NUM, 2.5E+1)

line9:(POUND_SIGN, ;)

line10:(INTEGER, 1)

line10:(POUND_SIGN, ;)

line11:(UNSIGNED_NUM, 2e1)

line11:(POUND_SIGN, ;)

line12:(UNSIGNED_NUM, 2e-1)

line12:(POUND_SIGN, ;)

line13:(LINE_NOTES, //运算符测试+分隔符测试)

line14:(POUND_SIGN, ()

line14:(IDENTIFIER, a)

line14:(OPERATOR, +)
```

```
line14:(IDENTIFIER, b)

line14:(POUND_SIGN, ))

line14:(POUND_SIGN, ;)

line15:(POUND_SIGN, [)

line15:(IDENTIFIER, a)

line15:(OPERATOR, -)

line15:(IDENTIFIER, b)

line15:(POUND_SIGN, ])

line15:(POUND_SIGN, ;)

line16:(POUND_SIGN, {)

line16:(IDENTIFIER, a)

line16:(OPERATOR, <=)

line16:(IDENTIFIER, b)

line16:(POUND_SIGN, })

line17:(IDENTIFIER, a)

line17:(OPERATOR, <)

line17:(IDENTIFIER, b)

line17:(POUND_SIGN, ;)

line18:(IDENTIFIER, a)

line18:(OPERATOR, >)

line18:(IDENTIFIER, b)

line18:(POUND_SIGN, ;)
```

```
line19:(IDENTIFIER, a)

line19:(OPERATOR, >=)

line19:(IDENTIFIER, b)

line20:(IDENTIFIER, a)

line20:(OPERATOR, =)

line20:(IDENTIFIER, b)

line20:(POUND_SIGN, ;)

line21:(IDENTIFIER, a)

line21:(OPERATOR, ==)

line21:(IDENTIFIER, b)

line21:(POUND_SIGN, ;)

line22:(IDENTIFIER, a)

line22:(OPERATOR, +=)

line22:(INTEGER, 1)

line22:(POUND_SIGN, ;)

line23:(IDENTIFIER, a)

line23:(OPERATOR, *)

line23:(IDENTIFIER, b)

line23:(POUND_SIGN, ;)

line24:(IDENTIFIER, a)

line24:(OPERATOR, *=)

line24:(IDENTIFIER, b)
```

```
line24:(POUND_SIGN, ;)

line25:(IDENTIFIER, a)

line25:(OPERATOR, &)

line25:(IDENTIFIER, b)

line25:(POUND_SIGN, ;)

line26:(IDENTIFIER, a)

line26:(OPERATOR, &&)

line26:(IDENTIFIER, b)

line26:(POUND_SIGN, ;)

line27:(IDENTIFIER, a)

line27:(OPERATOR, &=)

line27:(IDENTIFIER, b)

line27:(POUND_SIGN, ;)

line28:(OPERATOR, !)

line28:(IDENTIFIER, a)

line29:(IDENTIFIER, a)

line29:(OPERATOR, !=)

line29:(IDENTIFIER, b)

line29:(POUND_SIGN, ;)

line30:(IDENTIFIER, a)

line30:(OPERATOR, %)

line30:(IDENTIFIER, b)
```

```
line30:(POUND_SIGN, ;)

line31:(IDENTIFIER, a)

line31:(OPERATOR, %=)

line31:(IDENTIFIER, b)

line31:(POUND_SIGN, ;)

line32:(IDENTIFIER, a)

line32:(OPERATOR, ^)

line32:(IDENTIFIER, b)

line32:(POUND_SIGN, ;)

line33:(IDENTIFIER, a)

line33:(OPERATOR, ^=)

line33:(IDENTIFIER, b)

line33:(POUND_SIGN, ;)

line34:(IDENTIFIER, a)

line34:(OPERATOR, |)

line34:(IDENTIFIER, b)

line34:(POUND_SIGN, ;)

line35:(IDENTIFIER, a)

line35:(OPERATOR, ||)

line35:(IDENTIFIER, b)

line35:(POUND_SIGN, ;)

line36:(IDENTIFIER, a)
```

```
line36:(POUND_SIGN, ,)

line36:(IDENTIFIER, b)

line36:(POUND_SIGN, ;)

line37:(LINE_NOTES, //字符串常量与字符常量测试)

line38:(STRING, "this is a \"string")

line38:(POUND_SIGN, ;)

line39:(CHAR, '\'')

line39:(POUND_SIGN, ;)

line40:(LINE_NOTES, //一段 C 语言程序测试)

line41:(KEYWORD, struct)

line41:(IDENTIFIER, trie_node)

line41:(OPERATOR, *)

line41:(IDENTIFIER, new_node)

line41:(POUND_SIGN, ()

line41:(KEYWORD, int)

line41:(IDENTIFIER, level)

line41:(POUND_SIGN, ))

line42:(POUND_SIGN, {)

line43:(KEYWORD, struct)

line43:(IDENTIFIER, trie_node)

line43:(OPERATOR, *)

line43:(IDENTIFIER, p)
```

```
line43:(OPERATOR, =)

line43:(POUND_SIGN, ()

line43:(KEYWORD, struct)

line43:(IDENTIFIER, trie_node)

line43:(OPERATOR, *)

line43:(POUND_SIGN, ))

line43:(IDENTIFIER, malloc)

line43:(POUND_SIGN, ()

line43:(KEYWORD, sizeof)

line43:(POUND_SIGN, ()

line43:(KEYWORD, struct)

line43:(IDENTIFIER, trie_node)

line43:(POUND_SIGN, ))

line43:(POUND_SIGN, ))

line43:(POUND_SIGN, ;)

line44:(IDENTIFIER, memset)

line44:(POUND_SIGN, ()

line44:(IDENTIFIER, p)

line44:(POUND_SIGN, ,)

line44:(INTEGER, 0)

line44:(POUND_SIGN, ,)

line44:(KEYWORD, sizeof)
```

```
line44:(POUND_SIGN, ()

line44:(KEYWORD, struct)

line44:(IDENTIFIER, trie_node)

line44:(POUND_SIGN, ))

line44:(POUND_SIGN, ))

line44:(POUND_SIGN, ;)

line45:(IDENTIFIER, p)

line45:(OPERATOR, ->)

line45:(IDENTIFIER, level)

line45:(OPERATOR, =)

line45:(IDENTIFIER, level)

line45:(OPERATOR, +)

line45:(INTEGER, 1)

line45:(POUND_SIGN, ;)

line46:(KEYWORD, return)

line46:(IDENTIFIER, p)

line46:(POUND_SIGN, ;)

line47:(POUND_SIGN, })

line48:(LINE_NOTES, // 增加节点)

line49:(KEYWORD, void)

line49:(IDENTIFIER, trie_add)

line49:(POUND_SIGN, ()
```

```
line49:(KEYWORD, struct)

line49:(IDENTIFIER, trie_node)

line49:(OPERATOR, *)

line49:(IDENTIFIER, node)

line49:(POUND_SIGN, ,)

line49:(KEYWORD, unsigned)

line49:(KEYWORD, char)

line49:(OPERATOR, *)

line49:(IDENTIFIER, pattern)

line49:(POUND_SIGN, ))

line50:(POUND_SIGN, {)

line51:(KEYWORD, unsigned)

line51:(KEYWORD, int)

line51:(IDENTIFIER, cnt)

line51:(OPERATOR, =)

line51:(IDENTIFIER, pattern)

line51:(POUND_SIGN, [)

line51:(INTEGER, 0)

line51:(POUND_SIGN, ])

line51:(POUND_SIGN, ;)

line52:(KEYWORD, if)

line52:(POUND_SIGN, ()
```

```
line52:(IDENTIFIER, node)

line52:(OPERATOR, ->)

line52:(IDENTIFIER, type)

line52:(POUND_SIGN, [)

line52:(IDENTIFIER, cnt)

line52:(POUND_SIGN, ])

line52:(OPERATOR, ==)

line52:(IDENTIFIER, TRIR_CHILD_TYPE_NULL)

line52:(POUND_SIGN, ))

line53:(POUND_SIGN, {)

line54:(KEYWORD, if)

line54:(POUND_SIGN, ()

line54:(IDENTIFIER, cnt)

line54:(POUND_SIGN, ))

line55:(IDENTIFIER, set_child_str)

line55:(POUND_SIGN, ()

line55:(IDENTIFIER, node)

line55:(POUND_SIGN, ,)

line55:(IDENTIFIER, cnt)

line55:(POUND_SIGN, ,)

line55:(IDENTIFIER, pattern)

line55:(POUND_SIGN, ))
```

```
line55:(POUND_SIGN, ;)

line56:(KEYWORD, else)

line57:(IDENTIFIER, set_child_str)

line57:(POUND_SIGN, ()

line57:(IDENTIFIER, node)

line57:(POUND_SIGN, ,)

line57:(IDENTIFIER, cnt)

line57:(POUND_SIGN, ,)

line57:(IDENTIFIER, NULL)

line57:(POUND_SIGN, ))

line57:(POUND_SIGN, ;)

line58:(KEYWORD, return)

line58:(POUND_SIGN, ;)

line59:(POUND_SIGN, })

line60:(KEYWORD, else)

line60:(KEYWORD, if)

line60:(POUND_SIGN, ()

line60:(IDENTIFIER, node)

line60:(OPERATOR, ->)

line60:(IDENTIFIER, type)

line60:(POUND_SIGN, [)

line60:(IDENTIFIER, cnt)
```

```
line60:(POUND_SIGN, ])

line60:(OPERATOR, ==)

line60:(IDENTIFIER, TRIR_CHILD_TYPE_NODE)

line60:(POUND_SIGN, ))

line61:(POUND_SIGN, {)

line62:(IDENTIFIER, trie_add)

line62:(POUND_SIGN, ()

line62:(POUND_SIGN, ()

line62:(KEYWORD, struct)

line62:(IDENTIFIER, trie_node)

line62:(OPERATOR, *)

line62:(POUND_SIGN, ))

line62:(POUND_SIGN, ()

line62:(IDENTIFIER, node)

line62:(OPERATOR, ->)

line62:(IDENTIFIER, child)

line62:(POUND_SIGN, [)

line62:(IDENTIFIER, cnt)

line62:(POUND_SIGN, ])

line62:(POUND_SIGN, ))

line62:(POUND_SIGN, ,)

line62:(IDENTIFIER, pattern)
```

```
line62:(OPERATOR, +)

line62:(INTEGER, 1)

line62:(POUND_SIGN, ))

line62:(POUND_SIGN, ;)

line63:(KEYWORD, return)

line63:(POUND_SIGN, ;)

line64:(POUND_SIGN, })

line65:(KEYWORD, else)

line65:(KEYWORD, if)

line65:(POUND_SIGN, ()

line65:(IDENTIFIER, node)

line65:(OPERATOR, ->)

line65:(IDENTIFIER, type)

line65:(POUND_SIGN, [)

line65:(IDENTIFIER, cnt)

line65:(POUND_SIGN, ])

line65:(OPERATOR, ==)

line65:(IDENTIFIER, TRIR_CHILD_TYPE_LEAF)

line65:(POUND_SIGN, ))

line65:(LINE_NOTES, //分裂问题 )

line66:(POUND_SIGN, {)

line67:(KEYWORD, if)
```

```
line67:(POUND_SIGN, ()

line67:(IDENTIFIER, node)

line67:(OPERATOR, ->)

line67:(IDENTIFIER, child)

line67:(POUND_SIGN, [)

line67:(IDENTIFIER, cnt)

line67:(POUND_SIGN, ])

line67:(OPERATOR, ==)

line67:(IDENTIFIER, NULL)

line67:(POUND_SIGN, ))

line68:(KEYWORD, return)

line68:(POUND_SIGN, ;)

line69:(KEYWORD, unsigned)

line69:(KEYWORD, char)

line69:(OPERATOR, *)

line69:(IDENTIFIER, leaf)

line69:(OPERATOR, =)

line69:(POUND_SIGN, ()

line69:(KEYWORD, unsigned)

line69:(KEYWORD, char)

line69:(OPERATOR, *)

line69:(POUND_SIGN, ))
```

```
line69:(POUND_SIGN, ()

line69:(IDENTIFIER, node)

line69:(OPERATOR, ->)

line69:(IDENTIFIER, child)

line69:(POUND_SIGN, [)

line69:(IDENTIFIER, cnt)

line69:(POUND_SIGN, ])

line69:(POUND_SIGN, ))

line69:(POUND_SIGN, ;)

line70:(KEYWORD, struct)

line70:(IDENTIFIER, trie_node)

line70:(OPERATOR, *)

line70:(IDENTIFIER, child_node)

line70:(OPERATOR, =)

line70:(IDENTIFIER, new_node)

line70:(POUND_SIGN, ()

line70:(IDENTIFIER, node)

line70:(OPERATOR, ->)

line70:(IDENTIFIER, level)

line70:(POUND_SIGN, ))

line70:(POUND_SIGN, ;)

line71:(IDENTIFIER, add_child_node)
```

```
line71:(POUND_SIGN, ()

line71:(IDENTIFIER, node)

line71:(POUND_SIGN, ,)

line71:(IDENTIFIER, cnt)

line71:(POUND_SIGN, ,)

line71:(IDENTIFIER, child_node)

line71:(POUND_SIGN, ))

line71:(POUND_SIGN, ;)

line72:(IDENTIFIER, trie_add)

line72:(POUND_SIGN, ()

line72:(IDENTIFIER, child_node)

line72:(POUND_SIGN, ,)

line72:(IDENTIFIER, leaf)

line72:(OPERATOR, +)

line72:(INTEGER, 1)

line72:(POUND_SIGN, ))

line72:(POUND_SIGN, ;)

line73:(IDENTIFIER, trie_add)

line73:(POUND_SIGN, ()

line73:(IDENTIFIER, child_node)

line73:(POUND_SIGN, ,)

line73:(IDENTIFIER, pattern)
```

```
line73:(OPERATOR, +)

line73:(INTEGER, 1)

line73:(POUND_SIGN, ))

line73:(POUND_SIGN, ;)

line74:(KEYWORD, return)

line74:(POUND_SIGN, ;)

line75:(POUND_SIGN, })

line76:(POUND_SIGN, })

line77:(IDENTIFIER, BOOL)

line77:(IDENTIFIER, trie_find)

line77:(POUND_SIGN, ()

line77:(KEYWORD, struct)

line77:(IDENTIFIER, trie_node)

line77:(OPERATOR, *)

line77:(IDENTIFIER, node)

line77:(POUND_SIGN, ,)

line77:(KEYWORD, unsigned)

line77:(KEYWORD, char)

line77:(OPERATOR, *)

line77:(IDENTIFIER, p)

line77:(POUND_SIGN, ))

line78:(POUND_SIGN, {)
```

```
line79:(KEYWORD, int)

line79:(IDENTIFIER, index)

line79:(OPERATOR, =)

line79:(IDENTIFIER, p)

line79:(POUND_SIGN, [)

line79:(INTEGER, 0)

line79:(POUND_SIGN, ])

line79:(POUND_SIGN, ;)

line80:(KEYWORD, if)

line80:(POUND_SIGN, ()

line80:(IDENTIFIER, node)

line80:(OPERATOR, ->)

line80:(IDENTIFIER, type)

line80:(POUND_SIGN, [)

line80:(IDENTIFIER, index)

line80:(POUND_SIGN, ])

line80:(OPERATOR, ==)

line80:(IDENTIFIER, TRIR_CHILD_TYPE_NULL)

line80:(POUND_SIGN, ))

line81:(KEYWORD, return)

line81:(IDENTIFIER, FALSE)

line81:(POUND_SIGN, ;)
```

```
line82:(KEYWORD, else)

line82:(KEYWORD, if)

line82:(POUND_SIGN, ()

line82:(IDENTIFIER, node)

line82:(OPERATOR, ->)

line82:(IDENTIFIER, type)

line82:(POUND_SIGN, [)

line82:(IDENTIFIER, index)

line82:(POUND_SIGN, ])

line82:(OPERATOR, ==)

line82:(IDENTIFIER, TRIR_CHILD_TYPE_LEAF)

line82:(POUND_SIGN, ))

line83:(POUND_SIGN, {)

line84:(KEYWORD, if)

line84:(POUND_SIGN, ()

line84:(IDENTIFIER, node)

line84:(OPERATOR, ->)

line84:(IDENTIFIER, child)

line84:(POUND_SIGN, [)

line84:(IDENTIFIER, index)

line84:(POUND_SIGN, ])

line84:(OPERATOR, ==)
```

```
line84:(INTEGER, 0)

line84:(OPERATOR, &&)

line84:(IDENTIFIER, p)

line84:(POUND_SIGN, [)

line84:(INTEGER, 0)

line84:(POUND_SIGN, ])

line84:(OPERATOR, ==)

line84:(INTEGER, 0)

line84:(POUND_SIGN, ))

line85:(KEYWORD, return)

line85:(IDENTIFIER, TRUE)

line85:(POUND_SIGN, ;)

line86:(KEYWORD, else)

line86:(KEYWORD, if)

line86:(POUND_SIGN, ()

line86:(IDENTIFIER, node)

line86:(OPERATOR, ->)

line86:(IDENTIFIER, child)

line86:(POUND_SIGN, [)

line86:(IDENTIFIER, index)

line86:(POUND_SIGN, ])

line86:(OPERATOR, !=)
```

```
line86:(INTEGER, 0)

line86:(OPERATOR, &&)

line86:(IDENTIFIER, p)

line86:(POUND_SIGN, [)

line86:(INTEGER, 0)

line86:(POUND_SIGN, ])

line86:(OPERATOR, !=)

line86:(INTEGER, 0)

line86:(POUND_SIGN, ))

line86:(LINE_NOTES, //都不为0，则判断是否相同)

line87:(KEYWORD, return)

line87:(IDENTIFIER, strcmp)

line87:(POUND_SIGN, ()

line87:(POUND_SIGN, ()

line87:(KEYWORD, char)

line87:(OPERATOR, *)

line87:(POUND_SIGN, ))

line87:(POUND_SIGN, ()

line87:(IDENTIFIER, p)

line87:(POUND_SIGN, ))

line87:(POUND_SIGN, ,)

line87:(POUND_SIGN, ()
```

```
line87:(KEYWORD, char)

line87:(OPERATOR, *)

line87:(POUND_SIGN, ))

line87:(POUND_SIGN, ()

line87:(IDENTIFIER, node)

line87:(OPERATOR, ->)

line87:(IDENTIFIER, child)

line87:(POUND_SIGN, [)

line87:(IDENTIFIER, index)

line87:(POUND_SIGN, ])

line87:(POUND_SIGN, ))

line87:(POUND_SIGN, ))

line87:(OPERATOR, ==)

line87:(INTEGER, 0)

line87:(POUND_SIGN, ;)

line88:(KEYWORD, else)

line89:(KEYWORD, return)

line89:(IDENTIFIER, FALSE)

line89:(POUND_SIGN, ;)

line90:(POUND_SIGN, })

line91:(KEYWORD, return)

line91:(IDENTIFIER, trie_find)
```

```
line91:(POUND_SIGN, ()

line91:(POUND_SIGN, ()

line91:(IDENTIFIER, node)

line91:(OPERATOR, ->)

line91:(IDENTIFIER, child)

line91:(POUND_SIGN, [)

line91:(IDENTIFIER, index)

line91:(POUND_SIGN, ])

line91:(POUND_SIGN, ))

line91:(POUND_SIGN, ,)

line91:(IDENTIFIER, p)

line91:(OPERATOR, +)

line91:(INTEGER, 1)

line91:(POUND_SIGN, ))

line91:(POUND_SIGN, ;)

line92:(POUND_SIGN, })

line93:(KEYWORD, int)

line93:(IDENTIFIER, main)

line93:(POUND_SIGN, ()

line93:(POUND_SIGN, ))

line94:(POUND_SIGN, {)

line95:(IDENTIFIER, FILE)
```

```
line95:(OPERATOR, *)

line95:(IDENTIFIER, f)

line95:(OPERATOR, =)

line95:(IDENTIFIER, fopen)

line95:(POUND_SIGN, ()

line95:(STRING, "pattern-gbk")

line95:(POUND_SIGN, ,)

line95:(STRING, "rt")

line95:(POUND_SIGN, ))

line95:(POUND_SIGN, ;)

line96:(KEYWORD, if)

line96:(POUND_SIGN, ()

line96:(IDENTIFIER, f)

line96:(OPERATOR, ==)

line96:(IDENTIFIER, NULL)

line96:(POUND_SIGN, ))

line97:(POUND_SIGN, {)

line98:(IDENTIFIER, printf)

line98:(POUND_SIGN, ()

line98:(STRING, "can not open file 'pattern'\n")

line98:(POUND_SIGN, ))

line98:(POUND_SIGN, ;)
```

```
line99:(KEYWORD, return)

line99:(INTEGER, 1)

line99:(POUND_SIGN, ;)

line100:(POUND_SIGN, })

line101:(KEYWORD, struct)

line101:(IDENTIFIER, trie_node)

line101:(OPERATOR, *)

line101:(IDENTIFIER, root)

line101:(OPERATOR, =)

line101:(IDENTIFIER, new_node)

line101:(POUND_SIGN, ()

line101:(INTEGER, 0)

line101:(POUND_SIGN, ))

line101:(POUND_SIGN, ;)

line102:(KEYWORD, char)

line102:(IDENTIFIER, buffer)

line102:(POUND_SIGN, [)

line102:(INTEGER, 256)

line102:(POUND_SIGN, ])

line102:(POUND_SIGN, ;)

line103:(KEYWORD, while)

line103:(POUND_SIGN, ()
```

```
line103:(IDENTIFIER, fgets)

line103:(POUND_SIGN, ()

line103:(IDENTIFIER, buffer)

line103:(POUND_SIGN, ,)

line103:(KEYWORD, sizeof)

line103:(POUND_SIGN, ()

line103:(IDENTIFIER, buffer)

line103:(POUND_SIGN, ))

line103:(POUND_SIGN, ,)

line103:(IDENTIFIER, f)

line103:(POUND_SIGN, ))

line103:(OPERATOR, !=)

line103:(IDENTIFIER, NULL)

line103:(POUND_SIGN, ))

line104:(POUND_SIGN, {)

line105:(KEYWORD, char)

line105:(OPERATOR, *)

line105:(IDENTIFIER, p)

line105:(OPERATOR, =)

line105:(IDENTIFIER, trim_str)

line105:(POUND_SIGN, ()

line105:(IDENTIFIER, buffer)
```

```
line105:(POUND_SIGN, ))

line105:(POUND_SIGN, ;)

line106:(KEYWORD, if)

line106:(POUND_SIGN, ()

line106:(OPERATOR, *)

line106:(IDENTIFIER, p)

line106:(OPERATOR, ==)

line106:(INTEGER, 0)

line106:(POUND_SIGN, ))

line107:(KEYWORD, continue)

line107:(POUND_SIGN, ;)

line108:(IDENTIFIER, trie_add)

line108:(POUND_SIGN, ()

line108:(IDENTIFIER, root)

line108:(POUND_SIGN, ,)

line108:(POUND_SIGN, ()

line108:(KEYWORD, unsigned)

line108:(KEYWORD, char)

line108:(OPERATOR, *)

line108:(POUND_SIGN, ))

line108:(IDENTIFIER, strdup)

line108:(POUND_SIGN, ()
```

```
line108:(IDENTIFIER, p)

line108:(POUND_SIGN, ))

line108:(POUND_SIGN, ))

line108:(POUND_SIGN, ;)

line109:(POUND_SIGN, })

line110:(IDENTIFIER, fclose)

line110:(POUND_SIGN, ()

line110:(IDENTIFIER, f)

line110:(POUND_SIGN, ))

line110:(POUND_SIGN, ;)

line111:(IDENTIFIER, FILE)

line111:(OPERATOR, *)

line111:(IDENTIFIER, fData)

line111:(OPERATOR, =)

line111:(IDENTIFIER, fopen)

line111:(POUND_SIGN, ()

line111:(STRING, "input-gbk")

line111:(POUND_SIGN, ,)

line111:(STRING, "rt")

line111:(POUND_SIGN, ))

line111:(POUND_SIGN, ;)

line112:(KEYWORD, if)
```

```
line112:(POUND_SIGN, ()

line112:(IDENTIFIER, fData)

line112:(OPERATOR, ==)

line112:(IDENTIFIER, NULL)

line112:(POUND_SIGN, ))

line113:(POUND_SIGN, {)

line114:(IDENTIFIER, printf)

line114:(POUND_SIGN, ()

line114:(STRING, "can not open file 'input'\n")

line114:(POUND_SIGN, ))

line114:(POUND_SIGN, ;)

line115:(KEYWORD, return)

line115:(INTEGER, 1)

line115:(POUND_SIGN, ;)

line116:(POUND_SIGN, })

line117:(IDENTIFIER, FILE)

line117:(OPERATOR, *)

line117:(IDENTIFIER, fwrite)

line117:(OPERATOR, =)

line117:(IDENTIFIER, fopen)

line117:(POUND_SIGN, ()

line117:(STRING, "yipipei.txt")
```

```
line117:(POUND_SIGN, ,)

line117:(STRING, "w")

line117:(POUND_SIGN, ))

line117:(POUND_SIGN, ;)

line118:(KEYWORD, if)

line118:(POUND_SIGN, ()

line118:(IDENTIFIER, fwrite)

line118:(OPERATOR, ==)

line118:(IDENTIFIER, NULL)

line118:(POUND_SIGN, ))

line119:(POUND_SIGN, {)

line120:(IDENTIFIER, printf)

line120:(POUND_SIGN, ()

line120:(STRING, "can not open file 'pattern'\n")

line120:(POUND_SIGN, ))

line120:(POUND_SIGN, ;)

line121:(KEYWORD, return)

line121:(INTEGER, 1)

line121:(POUND_SIGN, ;)

line122:(POUND_SIGN, })

line123:(KEYWORD, int)

line123:(IDENTIFIER, index)
```

```
line123:(OPERATOR, =)

line123:(INTEGER, 0)

line123:(POUND_SIGN, ;)

line124:(KEYWORD, int)

line124:(IDENTIFIER, count)

line124:(OPERATOR, =)

line124:(INTEGER, 0)

line124:(POUND_SIGN, ;)

line125:(KEYWORD, while)

line125:(POUND_SIGN, ()

line125:(IDENTIFIER, fgets)

line125:(POUND_SIGN, ()

line125:(IDENTIFIER, buffer)

line125:(POUND_SIGN, ,)

line125:(KEYWORD, sizeof)

line125:(POUND_SIGN, ()

line125:(IDENTIFIER, buffer)

line125:(POUND_SIGN, ))

line125:(POUND_SIGN, ,)

line125:(IDENTIFIER, fData)

line125:(POUND_SIGN, ))

line125:(OPERATOR, !=)
```

```
line125:(IDENTIFIER, NULL)

line125:(POUND_SIGN, ))

line126:(POUND_SIGN, {)

line127:(IDENTIFIER, index)

line127:(OPERATOR, ++)

line127:(POUND_SIGN, ;)

line128:(KEYWORD, char)

line128:(OPERATOR, *)

line128:(IDENTIFIER, p)

line128:(OPERATOR, =)

line128:(IDENTIFIER, trim_str)

line128:(POUND_SIGN, ()

line128:(IDENTIFIER, buffer)

line128:(POUND_SIGN, ))

line128:(POUND_SIGN, ;)

line129:(KEYWORD, if)

line129:(POUND_SIGN, ()

line129:(OPERATOR, *)

line129:(IDENTIFIER, p)

line129:(OPERATOR, ==)

line129:(INTEGER, 0)

line129:(POUND_SIGN, ))
```

```
line130:(KEYWORD, continue)

line130:(POUND_SIGN, ;)

line131:(KEYWORD, if)

line131:(POUND_SIGN, ()

line131:(IDENTIFIER, trie_find)

line131:(POUND_SIGN, ()

line131:(IDENTIFIER, root)

line131:(POUND_SIGN, ,)

line131:(POUND_SIGN, ()

line131:(KEYWORD, unsigned)

line131:(KEYWORD, char)

line131:(OPERATOR, *)

line131:(POUND_SIGN, ))

line131:(IDENTIFIER, p)

line131:(POUND_SIGN, ))

line131:(POUND_SIGN, ))

line132:(POUND_SIGN, {)

line133:(IDENTIFIER, count)

line133:(OPERATOR, ++)

line133:(POUND_SIGN, ;)

line134:(IDENTIFIER, printf)

line134:(POUND_SIGN, ()
```

```
line134:(STRING, "%d: %s yes\n")

line134:(POUND_SIGN, ,)

line134:(IDENTIFIER, index)

line134:(POUND_SIGN, ,)

line134:(IDENTIFIER, p)

line134:(POUND_SIGN, ))

line134:(POUND_SIGN, ;)

line135:(IDENTIFIER, fputs)

line135:(POUND_SIGN, ()

line135:(IDENTIFIER, p)

line135:(POUND_SIGN, ,)

line135:(IDENTIFIER, fwrite)

line135:(POUND_SIGN, ))

line135:(POUND_SIGN, ;)

line136:(IDENTIFIER, fprintf)

line136:(POUND_SIGN, ()

line136:(IDENTIFIER, fwrite)

line136:(POUND_SIGN, ,)

line136:(STRING, "\n")

line136:(POUND_SIGN, ))

line136:(POUND_SIGN, ;)

line137:(POUND_SIGN, })
```

```
line138:(KEYWORD, else)

line139:(IDENTIFIER, printf)

line139:(POUND_SIGN, ()

line139:(STRING, "%d: %s no\n")

line139:(POUND_SIGN, ,)

line139:(IDENTIFIER, index)

line139:(POUND_SIGN, ,)

line139:(IDENTIFIER, p)

line139:(POUND_SIGN, ))

line139:(POUND_SIGN, ;)

line140:(POUND_SIGN, })

line141:(IDENTIFIER, printf)

line141:(POUND_SIGN, ()

line141:(STRING, "read %d lines, found %d\n")

line141:(POUND_SIGN, ,)

line141:(IDENTIFIER, index)

line141:(POUND_SIGN, ,)

line141:(IDENTIFIER, count)

line141:(POUND_SIGN, ))

line141:(POUND_SIGN, ;)

line142:(IDENTIFIER, fclose)

line142:(POUND_SIGN, ()
```

```
line142:(IDENTIFIER, fData)

line142:(POUND_SIGN, ))

line142:(POUND_SIGN, ;)

line143:(IDENTIFIER, fclose)

line143:(POUND_SIGN, ()

line143:(IDENTIFIER, fwrite)

line143:(POUND_SIGN, ))

line143:(POUND_SIGN, ;)

line144:(KEYWORD, return)

line144:(INTEGER, 0)

line144:(POUND_SIGN, ;)

line145:(POUND_SIGN, })

line146:(POUND_SIGN, })

------------------------------------------------------------

Total Word: 784
Total Line: 146
Total KEYWORD: 80
Total IDENTIFIER: 228
Total OPERATOR: 103
Total POUND_SIGN: 327
Total INTEGER: 27
Total UNSIGNED_NUM: 3
Total STRING: 15
Total CHAR: 1
```

输出说明

经验证，两个程序对同一 C 语言代码进行词法分析的结果相同，各类单词统计数目以及行数信息也完全吻合，证明两个程序的正确性。