词法分析程序实验

17341092 梁萍佳

实验目的

通过扩充已有的样例语言TINY语言的词法分析程序，为扩展TINY语言TINY + 构造词法分析程序，从而掌握词法分析程序的构造方法

实验内容

了解样例语言TINY及TINY编译器的实现，了解扩展TINY语言TINY + ，用C语言在已有的TINY词法分析器基础上扩展，构造TINY + 的词法分析程序。

实验要求

将TINY + 源程序翻译成对应的TOKEN序列，并能检查一定的词法错误。

实验过程

下面我们首先找到TINY+的定义，然后根据定义用自动状态机来完成识别程序

词法定义

这是从网上资料找来的词法定义。

> **2.1 TINY+语言的词法定义**
>
> 1. TINY+语言的关键字（keyword）包括：
>
> or and int bool char while do
>
> if then else end repeat until read write
>
> 所有的关键字是程序设计语言保留使用的，并且用小写字母表示，用户自己定义的标识符不能和关键字重复。
>
> 2. 特殊符号的定义如下：

> <= >= , '

{ } ; := + - * / ( ) < =

3. 其他种类的单词包括标识符 ID，数字 NUM 以及字符串 STRING，他们的正规表达式的定义如下：

   ID=letter (letter | digit)*

   标识符是以字母开头，由字母和数字混合构成的符号串。

   NUM=digit digit*

   TINY+中对数字的定义和 TINY 相同。

   STRING=' any character except ' '

   一个字符串类型的单词是用单引号括起来的字符串'...'，引号内可出现除了' 以外的任何符号。一个字符串不能跨行定义。

   letter=a|...|z|A|...|Z

   digit=0|...|9

   小写和大写字母是不同的。

4. 空白包括空格、回车以及 Tab。所有的空白在词法分析时，被当作单词 ID，NUM 以及保留字的分隔符，在词法分析之后，他们不被当作单词保留。

5. 注释是用花括号括起来的符号串{...}，注释不能嵌套定义，但注释的定义可以跨行。

翻译程序

这里首先我们将介绍代码中的数据定义，然后再介绍如何根据自动机思想构造识别的函数。

数据定义

首先定义词的类型，这可以直接从上述词法定义中得到:

```
enum TOKEN_TYPE { KEY, SYM, ID, NUM, STR ,EMPTY};
```

这里面除了词法定义的类型外，考虑到后面语法分析可能会有的需求，因此还加上了一个EMPTY，这个用来表示代码的末尾。（即$符号）。

然后我们定义Token的数据结构，这里type就是Token的词类型，value就是对应值。print方法用来输出。

```
struct Token {
    TOKEN_TYPE type;
    string value;
    Token(TOKEN_TYPE t, string val) {
        type = t;
        value = val;
```

```
    }
    Token() {
        type = EMPTY;
        value = "";
    }
    void print() {
        switch (type)
        {
        case KEY:
            cout << "(KEY," << value << ")";
            break;
        case SYM:
            cout << "(SYM," << value << ")";
            break;
        case ID:
            cout << "(ID," << value << ")";
            break;
        case NUM:
            cout << "(NUM," << value << ")";
            break;
        case STR:
            cout << "(STR," << value << ")";
            break;
        default:
            cout << "(-1," << value << ")";
            break;
        }
    }
};
```

接下来还要定义的是一些全局变量以及辅助函数，在这里

tokens将会用来存储分析完后的全部token序列；

keywords,symbols则是用来在分析过程中使用；

next_state表示状态机的下一个状态；

line_cnt表示分析到的行数，用于输出错误时给出错误的定位；

buf为当前已经扫描而还没形成token的字符序列；

cursym为当前扫描到的字符；

sym_used是用来表示cursym是否已经用过，这个是用来处理两个token之间没有空格直接连接这种情况的；

void error(string)用来输出词法错误

```
vector<Token> tokens;
//15
string keywords[]={ "if","then","else","end","repeat","until",
"read","write","or","and","int","bool","char","while","do" };
//15
string symbols[]={ "+","-","*","/","=","<","(",")",";",
":=",">","<=",">=",",","'" };
int next_state = 0;
int line_cnt = 0;//标记当前扫描到的行数
string buf = "";//当前扫过的还没成token的字符序列
```

```
int sym_used = 1;//标识cursym已经用过与否，用过为1，否则为0
char cursym = ' ';
void error(string mes) {
    cout << "error at line " << line_cnt << " :" << mes << endl;
}
```

状态定义

首先定义初始状态S0，这个状态是用来根据token第一个字符分析接下来考虑识别的词类型的。我们从词法定义中可以发现各个词类型是可以很容易根据第一个字符来分辨的：若为数字则为NUM；若第一个字符为字母，则可能为KEY或ID；若为符号 ' 则为STR ；若为{，则为注释；剩下的则为SYM。

```
int S0_initial(FILE* source) {//初始态 返回值表示已经识别出来的token类别,-1表示还没识别好
    if (sym_used == 1)cursym = fgetc(source);//当前符号用过了才再读
    sym_used = 0;
    if (cursym == -1) {//EOF已经读完了 要在自动机里检测cursym
        return -1;
    }
    if (cursym == ' ' || cursym == '\t') {//空字符就继续回到0状态
        next_state = 0;
    }
    else if (cursym == '\n') {
        next_state = 0;
        line_cnt += 1;
    }
    else if (isdigit(cursym)) {//是数字 那就可能是NUM，去S1识别num
        next_state = 1;
    }
    else if (isalpha(cursym)) {//是字母，可能是ID或KEY，去S2识别
        next_state = 2;
    }
    else if (cursym == '\'') {//是 ' 那可能是STR，去S3识别
        next_state = 3;
    }
    else if (cursym != '{') {//又不是'{' 那只可能是SYM啦，去S4
        next_state = 4;
    }
    else {//cursym=='{' 这开始是注释,去把注释都扫完再回来!
        next_state = 5;
    }
    return -1;
}
```

然后定义状态S1，这个状态是用来识别数字的。

```
int S1_num(FILE* source) {//识别数字中
    buf += cursym;
    cursym = fgetc(source);
    if (cursym == -1) {//EOF 已经读完了
        return NUM;
    }
    if (cursym == ' ' || cursym == '\t') {//空字符就识别完啦，
        next_state = 0;
        sym_used = 1;
        return NUM;
```

```
    }
    else if (cursym == '\n') {
        next_state = 0;
        line_cnt += 1;
        sym_used = 1;
        return NUM;
    }
    else if (isdigit(cursym)) {//还是数字就继续识别，继续S1
        next_state = 1;
    }
    else {//其他那也当作识别完啦，可能已经遇到下一个token了，
        next_state = 0;
        sym_used = 0;//读了下一个token的字符，因此标记一下
        return NUM;
    }
    return -1;
}
```

状态S2用来识别ID或KEY，这里用来分辨ID和KEY的方法就是在读完字母或数字之后，直接用字符串匹配来判断是否为KEY，否则就是ID

```
int S2_id(FILE* source) {//字母开头的，标识符没跑啦 扫完再看是不是关键字
    buf += cursym;
    cursym = fgetc(source);
    if (cursym == -1) {//EOF 已经读完了
        for (int i = 0; i < 15; i++) {//看看是不是keyword
            if (buf == keywords[i]) {
                return KEY;
            }
        }
        //不是KEY那就是ID了
        return ID;
    }
    if (isdigit(cursym) || isalpha(cursym)) {//还是字母或数字，那就继续S2识别
        next_state = 2;
    }
    else {//剩下的要么是符号要么是空格，反正不是token的
        sym_used = 0;
        next_state = 0;
        for (int i = 0; i < 15; i++) {//看看是不是keyword
            if (buf == keywords[i]) {
                return KEY;
            }
        }
        //不是KEY那就是ID了
        return ID;
    }
    return -1;

}
```

状态S3用来识别STR，这里就只需要直接扫描至有另一个符号'为止即可。注意到字符串定义不能换行，如果有换行符则为错误。

```
int S3_str(FILE* source) {// '开头的只能是字符串咯 一直识别到另一个'为止
    cursym = fgetc(source);//先拿一下第一个字符,原来开头的'不需要
```

```
    while (cursym != '\'') {
        if (cursym == -1) {//字符串还没结束文件就没了
            error("字符串还没结束就没了");
            exit(0);
        }
        else {
            buf += cursym;
        }
        cursym = fgetc(source);
    }
    //扫到cursym=='\''了 字符串已经完结了
    sym_used = 1;//记得是用过这个'\''了
    next_state = 0;
    return STR;//识别成功咯
}
```

状态S4用来识别符号，由于有最长匹配原则所以还需稍加处理。

```
int S4_sym(FILE* source) {
    buf += cursym;
    sym_used = 1;
    //注意最长匹配原则
    if (cursym == ':' || cursym == '<' || cursym == '>') {//有可能是两个字符的sym
        cursym = fgetc(source);
        if (cursym == '=') {//的确是两字符的
            buf += cursym;
            sym_used = 1;
        }
        else {//不是，那就是拿错了，标记下
            sym_used = 0;
        }
    }
    for (int i = 0; i < 15; i++) {
        if (buf == symbols[i]) {//匹配上，的确是合法SYM了
            next_state = 0;
            return SYM;
        }
    }
    //都没匹配上，失败啦
    error("这SYMBOL不对啊！");
    exit(0);
    return -1;
}
```

状态S5用来将注释扫完，跳过注释，这里只需要扫描到}出现即可

```
int S5_note(FILE* source) {//把注释全部过掉
    cursym = fgetc(source);//先拿一下第一个字符,原来开头的'不需要
    while (cursym != '}') {
        if (cursym == -1) {//注释还没结束文件就没了
            error("注释还没结束就没了");
            exit(0);
        }
        if (cursym == '\n') {//注释里面的行数还是要算的
            line_cnt++;
        }
```

```
        cursym = fgetc(source);
    }
    //扫到cursym=='\''了 字符串已经完结了
    sym_used = 1;//记得是用过这个'}'了
    next_state = 0;
    return -1;//注释已经过完咯
}
```

识别Token

接下来只需要用有限状态自动机的算法来识别即可得到Token了。这里先定义一个函数数组，其值就是上面的各个状态对应的函数，这样方便编写；然后注意到这些函数返回值代表的是识别出来的Token类型，-1则表示还没识别出来，将继续，否则已识别则根据返回值以及buf内容生成Token并返回Token。cursym为-1即EOF，文件已经读完，则返回一个空的，表示末尾。

```
int (*state[])(FILE* source) = { S0_initial,S1_num,S2_id,S3_str,S4_sym,S5_note
};
Token getNextToken(FILE* source) {
    next_state = 0;
    int type = -1;
    buf = "";
    while (type = -1) {
        cout<<"next_state:"<<next_state<<endl;
        type = state[next_state](source);
        if (type != -1) {
            return Token((TOKEN_TYPE)type, buf);
        }
        if(cursym==-1){
            return Token();
        }
    }
}
```

验证结果

为了验证我们得到的程序是正确的，下面编写了调用词法分析的程序，以及一段TINY+代码，然后观察输出是否符合预期。

TINY+代码 (src.txt)，里面前半部分为TINY+里面各种可能出现的词，后半部分为一个样例程序：

```
{these are some possible words in TINY}
or and int bool char
while do if then else
end repeat until read write
, ; := + -
* / ( ) < = > <= >= a2c 123 'EFG'

{this is an example source code}
int A,B;
bool C1,C2,C3;
char D;
D:='scanner';
while A<=B do
    A:=A*2
end
```

验证程序将获取src.txt里面的Token序列，并输出：

```
int main() {
    char src_name[100];
    scanf("%s",src_name);
    FILE* source = fopen(src_name, "r");
    while (cursym != -1) {
        Token a=getNextToken(source);
        a.print();
        cout<<" ";
        tokens.push_back(a);
    }
    if(tokens[tokens.size()-1].type!=-1){
        tokens.push_back(Token());
    }
    fclose(source);
    cout<<endl<<endl;
    for (int i = 0; i < tokens.size(); i++) {
        tokens[i].print();
        cout <<"  ";
        if(i%5==0)cout<<endl;
    }
}
```

程序最后输出的Token序列结果如下：

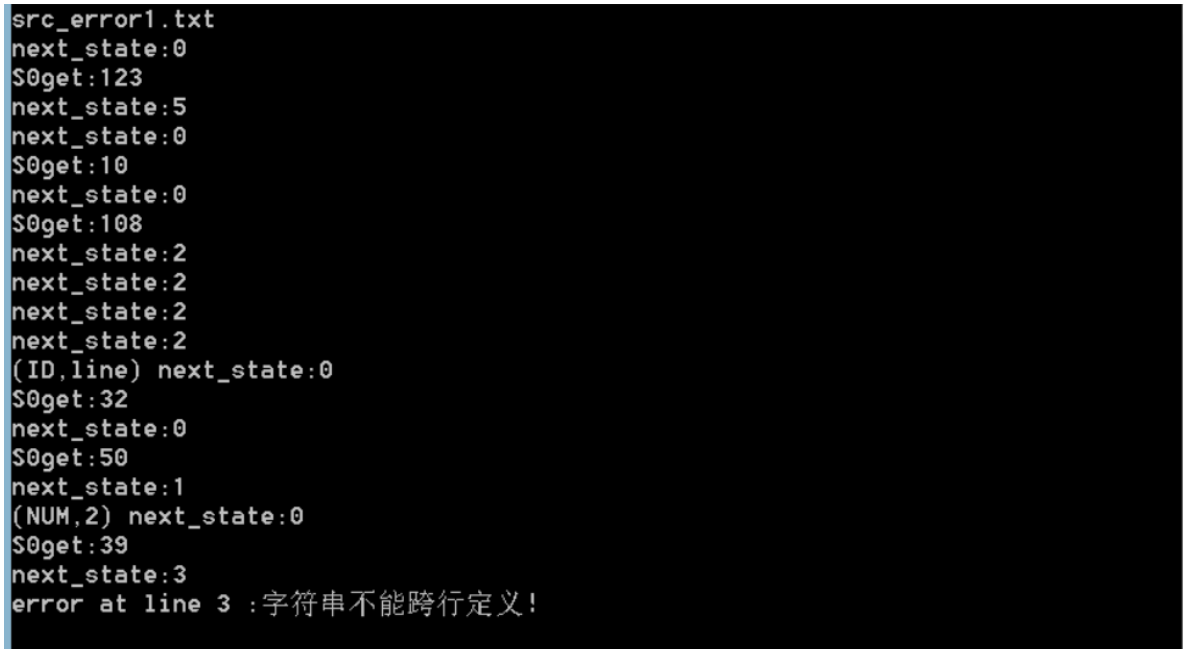
```
(KEY,or)  (KEY,and)  (KEY,int)  (KEY,bool)  (KEY,char)
(KEY,while)  (KEY,do)  (KEY,if)  (KEY,then)  (KEY,else)
(KEY,end)  (KEY,repeat)  (KEY,until)  (KEY,read)  (KEY,write)
(SYM,,)  (SYM,;)  (SYM,:=)  (SYM,+)  (SYM,-)
(SYM,*)  (SYM,/)  (SYM,()  (SYM,))  (SYM,<)
(SYM,=)  (SYM,>)  (SYM,<=)  (SYM,>=)  (ID,a2c)
(NUM,123)  (STR,EFG)  (KEY,int)  (ID,A)  (SYM,,)
(ID,B)  (SYM,;)  (KEY,bool)  (ID,C1)  (SYM,,)
(ID,C2)  (SYM,,)  (ID,C3)  (SYM,;)  (KEY,char)
(ID,D)  (SYM,;)  (ID,D)  (SYM,:=)  (STR,scanner)
(SYM,;)  (KEY,while)  (ID,A)  (SYM,<=)  (ID,B)
(KEY,do)  (ID,A)  (SYM,:=)  (ID,A)  (SYM,*)
(NUM,2)  (KEY,end)  (-1,)
```

可见其输出符合预期。下面再验证错误识别，为此编写了src_error1.txt到src_error4.txt

src_error1.txt验证能否发现字符串不能跨行定义的错误:

```
{string can't be defined over lines}
line 2
'this is a string
aasd'
```

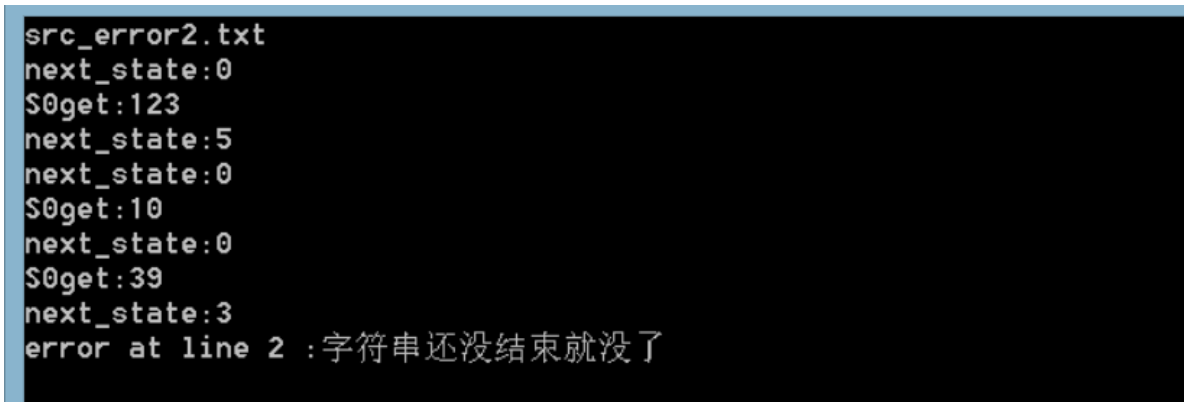
输出结果:



src_error2.txt 验证能否发现字符串没有结尾的错误:

```
{string without end}
'aa str
```

输出结果:



src_error3.txt 验证能否发现不正确的字符:

```
{undefined symbols}
helloworld!
```

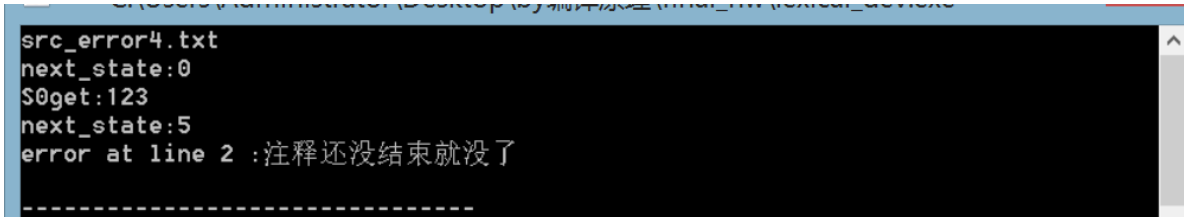
输出结果:

```
src_error3.txt
next_state:0
S0get:123
next_state:5
next_state:0
S0get:10
next_state:0
S0get:104
next_state:2
next_state:2
next_state:2
next_state:2
next_state:2
next_state:2
next_state:2
next_state:2
next_state:2
(ID,helloworld) next_state:0
S0get:33
next_state:4
error at line 2 :这SYMBOL不对啊！
```

src_error4.txt 验证能否发现注释没有结束:

```
{comment should have an end
```

输出结果:



至此，词法分析程序的正确性已经得到验证。

实验总结

这次实验我首先找到了网上关于TINY+定义的资料，然后根据其定义以及有限状态自动机的思想来实现了识别生成Token的算法。这里面虽然没有严格按照正则文法转有限状态自动机的算法来得到这些状态，但总体思想上还是有所借鉴了。这个词法分析程序还能检查一定的词法错误，给出错误类型和行数，从而方便定位。最后我还给出了几段程序，通过实验验证了程序的正确性。