

编译原理第一次实验报告-C-Minus 扫描程序

郑明钰

201711210110

1.实验目的：针对 C-Minus 语言，设计并实现扫描程序

2.C-Minus 语言的词法说明，输入、输出的格式

2.1 (1) C-Minus 语言的关键字有： else if int return void while
共 6 个，所有的关键字都是保留字，并且必须是小写

(2) C-Minus 语言的专有符号： + - * / < <= > >= == != = ; , ()
[] {} /* */

(3) 其他标记是 ID 和 NUM，通过下列正则表达式定义：

ID=letter letter*

NUM=digit digit*

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

digit=0|...|9

小写和大写字母有区别

(4) 空格由空白、换行符和制表符组成，空格通常被忽略，除了它必须分开 ID、NUM 关键字

(5) 注释用通常的 C 语言符号 /*...*/围起来，注释可以放在任何空白出现的位置，且可以超过一行，注释不能嵌套。

2.2 (1) 输入的格式是项目文件夹下的一个 txt 文件，更改其中的内容来设置不同的输入，通过文件流读取文件

(2) 输出的格式类似课本 P56 Tiny 扫描程序的输出，输出到标准输出流 stdout 中，进而显示在屏幕上。

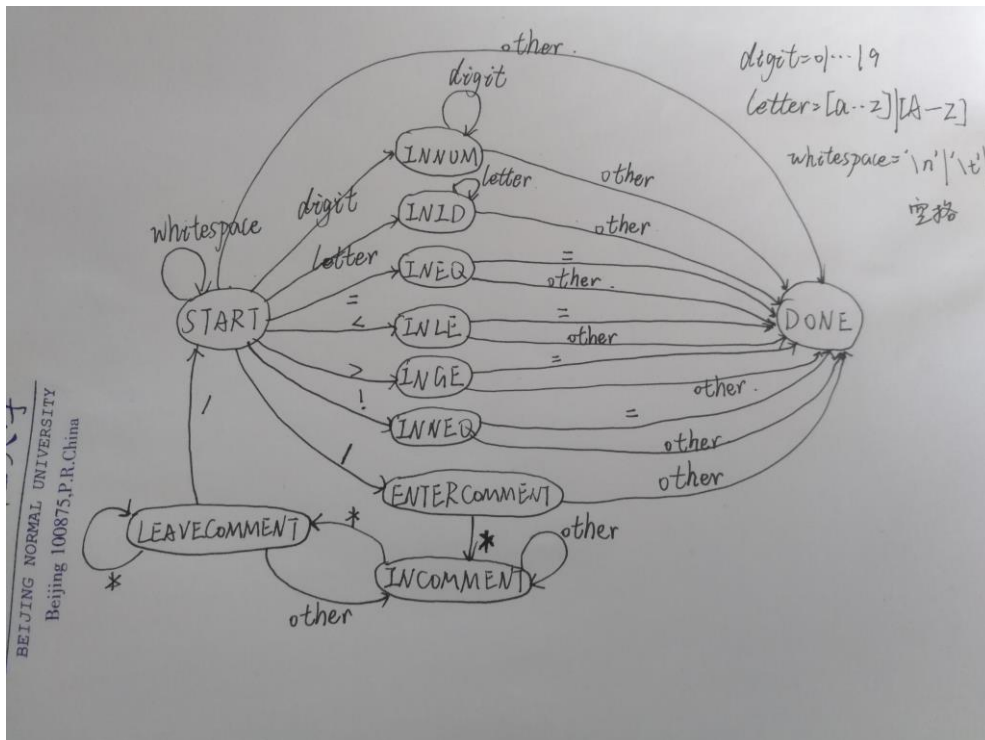
3. 实验原理：模仿 Tiny 语言的扫描程序来实现 C-Minus 语言的扫描程序

3.1 记号的种类和记号所代表的串集：记号种类如代码所示

```
typedef enum
{
    //book-keeping tokens
    ENDFILE, ERROR,
    //保留字
    ELSE, IF, INT, RETURN, VOID, WHILE,
    //专用符号
    PLUS, MINUS, TIMES, OVER, LT, LEQ, GT, GEQ, EQ, NEQ, ASSIGN, SEMI, COMMA, LPAREN, RPAREN,
    LMBRACKET, RMBRACKET, LBBRACKET, RBBRACKET,
    //一般数字和标识符
    NUM, ID
}TokenType;
```

3.2 记号的正则表达式：保留字和专用符号的正则表达式较为简单，如保留字 else 的正则表达式就是 else，< 的正则表达式即是元字符<，ID 和 NUM 的正则表达式在上面已有定义。

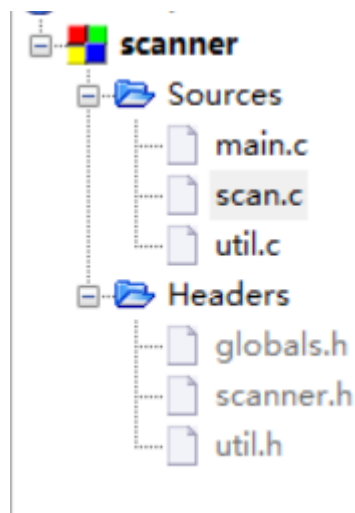
3.3 记号的 DFA：



4.程序的功能和程序说明

4.1 程序功能是从一个 txt 文件中读取 C-Minus 程序，输出其对应的记号到 stdout 显示屏上。

4.2 程序分为以下几个文件：其中扫描器主要实现在 scan.c 文件中，其他为辅助文件



主要函数有：

(1) void printToken(TokenType token,const char* tokenString) //

输出记号类型和串值

(2) static int getNextChar() //读取下一个字符

(3) static void ungetNextChar() //考虑先行问题，反填字符

(4) static TokenType reservedLookup(char*s) //查询记号是否是保留字

(5) TokenType getToken() //扫描程序主要函数，利用双重嵌套 case 语句实现 DFC，从而得到一个记号

5.输入实例：

实例一：以课本附录 A 中的 C-Minus 示例程序作为输入

输出为

```
test.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
int gcd (int u, int v)
{
    if (v == 0)
        return u ;
    else
        return gcd(v,u-u/v*v);
    /* u-u/v*v == u mod v */
}

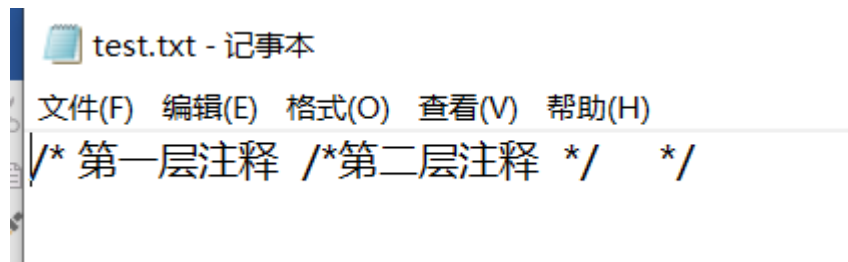
void main(void)
{
    int x; int y;
    x = input();
    y = input();
    output(gcd(x,y));
}
```

C:\Users\xx\Desktop\scanner\bin\Debug\scanner.exe

```
C-Minus Scanner:
1: int gcd (int u, int v)
  1 reserved word:int
  1 ID, name= gcd
  1 (
  1 reserved word:int
  1 ID, name= u
  1 ,
  1 reserved word:int
  1 ID, name= v
  1 )
2: {
3:   if (v == 0)
  3 reserved word:if
  3 (
  3 ID, name= v
  3 ==
  3 NUM, val= 0
  3 )
4:     return u ;
  4 reserved word:return
  4 ID, name= u
  4 ;
5:   else
  5 reserved word:else
6:     return gcd(v,u-u/v*v);
  6 reserved word:return
  6 ID, name= gcd
  6 (
  6 ID, name= v
  6 ,
  6 ID, name= u
  6 -
  6 ID, name= u
  6 /
  6 ID, name= v
  6 *
  6 ID, name= v
  6 )
  6 ;
7:   /* u-u/v*v == u mod v */
8: }
9:
10: void main(void)
  10 reserved word:void
  10 ID, name= main
  10 (
  10 reserved word:void
```

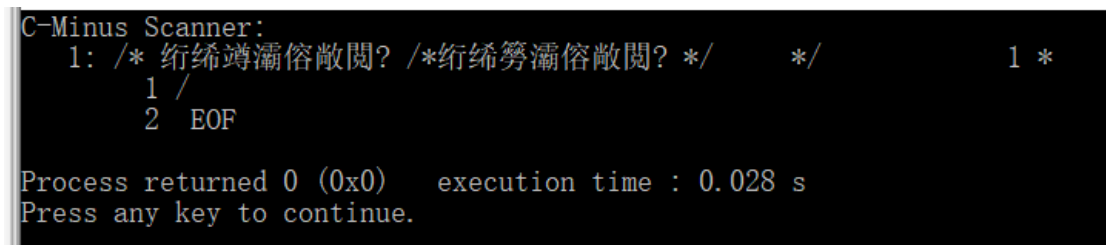
```
11: {
12:   int x; int y;
  12 reserved word:int
  12 ID, name= x
  12 ;
  12 reserved word:int
  12 ID, name= y
  12 ;
13:   x = input();
  13 ID, name= x
  13 =
  13 ID, name= input
  13 (
  13 )
  13 ;
14:   y = input();
  14 ID, name= y
  14 =
  14 ID, name= input
  14 (
  14 )
  14 ;
15:   output(gcd(x,y));
  15 ID, name= output
  15 (
  15 ID, name= gcd
  15 (
  15 ID, name= x
  15 ,
  15 ID, name= y
  15 )
  15 )
  15 ;
16: }
17 EOF
```

实例二： 考虑嵌套注释，可见扫描程序无法正确识别嵌套注释，正如 C-Minus 语法所规定的，不允许嵌套注释



```
test.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
/* 第一层注释 /*第二层注释 */ */
```

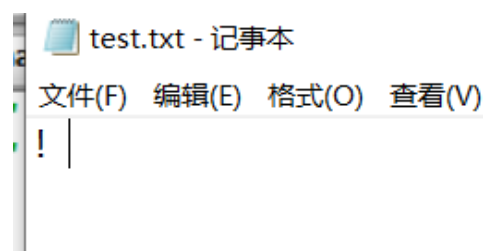
输出为：



```
C-Minus Scanner:
1: /* 第一层注释 /*第二层注释 */ */      1 *
    1 /
    2 EOF

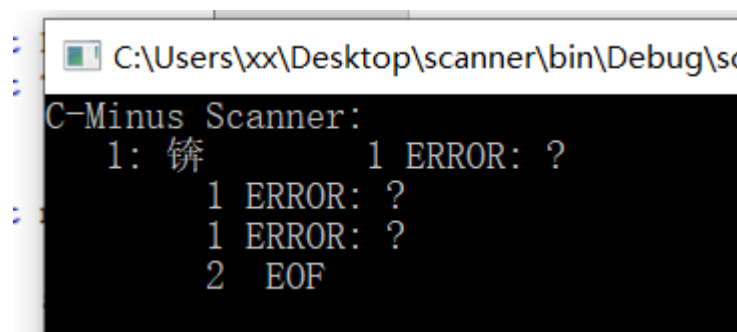
Process returned 0 (0x0)   execution time : 0.028 s
Press any key to continue.
```

实例三： 输入一个 C-Minus 不存在的专有符号 !，扫描程序识别错误，生成 ERROR 记号



```
test.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V)
!
```

输出为：



```
C:\Users\xx\Desktop\scanner\bin\Debug\sc
C-Minus Scanner:
1: !      1 ERROR: ?
    1 ERROR: ?
    1 ERROR: ?
    2 EOF
```

6.总结

我觉得重点在于设计好 DFA，然后再用双重 case 语句实现。

这个扫描程序还有许多可以改进的地方，比如记号的串值位于变量 tokenString 中，这个是提前定义好的，tokenString 的长度固定为 41，因此标识符 ID 也就不能超过 40 个字符，为每一个标识符都分配 40 个字符长度的数组也非常浪费空间，可以用动态分配空间如 realloc 来优化。

另外 getNextChar 函数在获取新代码行时只读取 256 个字符，虽然这个假设允许了更简单的代码，但却不能正确地处理行的字符超过 255 个字符的 C-Minus 程序。

在查询保留字时采用了线性搜索的方式，这对于 C-minus 这种保留字很少的语言问题不大，但是如果某种语言保留字很多，这时就要考虑搜索的效率，应该采用哈希表等方法实现。