深 圳 大 学 实 验 报 告

课程名称：编译原理

实验项目名称：词法分析程序设计

学院：计算机与软件学院

专业：计算机科学与技术

指导教师：罗成文

报告人：黎浩然 学号：2018112061 班级：01

实验时间：2022/5/18

实验报告提交时间：

教务部制

实验目的与要求：

实验目的

- TINY 语言的词法由 TINYSyntax.ppt 描述;
- TINY 语言的词法分析器由 TINYScanner.rar 的 C 语言代码实现;
- TINY+语言的词法由 TINY+Syntax.doc 描述。

任务:理解 TINY 语言的词法及词法分析器的实现，并基于该词法分析器， 实现拓展语言 TINY+的词法分析器。

实验要求

(1) TINY+词法分析器以 TINY+源代码为输入，输出为识别出的 token 序列;

(2) 词法分析器以最长匹配为原则，例如‘:=’应识别为赋值符号而非单独的‘:’及‘=’;

(3) Token 以(种别码，属性值)表示，包含以下类型的种别码:

a) KEY 为关键字;

b) SYM 为系统特殊字符;

c) ID 为变量;

d) NUM 为数值常量;

e) STR 为字符串常量。

(4) 识别词法错误。词法分析器可以给出词法错误的行号并打印出对应的出错消息，主要包含以下类型的词法错误:

a) 非法字符。即不属 TINY+字母表的字符，比如$就是一个非法字符;

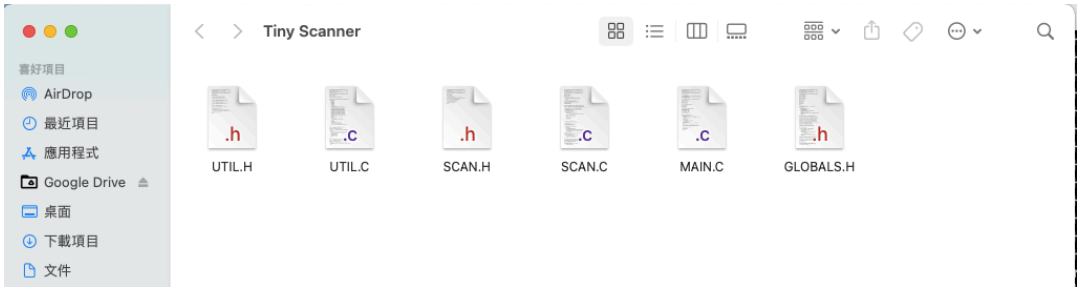
b) 字符串匹配错误，比如右部引号丢失，如‘scanner

c) 注释的右部括号丢失或匹配错误，如{thisisanexample

实验方法、步骤及过程：

CMake 构建、Clang 编译 TINY Scanner

打开 TINY SCANNER 的源代码文件夹，有以下六个文件：



发现所有的文件名都是大写的。因为我的 Mac 使用的是大小写不敏感的 APFS 文件系统，所以不需要修改。否则应该将所有文件名改为对应的小写名称才能工作。

使用 cmake 跨平台构建系统 MacOS 平台下的 Makefile。首先用 Mac 的软件包管理工

具 homebrew 安装 cmake：“brew install cmake”：

```
Tiny Scanner — -zsh — 120×40
ernest@MacBook-Pro Tiny Scanner % brew install cmake
Running `brew update --preinstall`...
==> Auto-updated Homebrew!
Updated Homebrew from db723f5dd to 34dd8e305.
Updated 2 taps (homebrew/core and homebrew/cask).
```

编写 CMakeLists.txt，工程和最终的可执行文件的名称均为 tinyplus。将这六个代码文件加入到工程中。注意这里需要指定用 C 编译器来编译，而不是 C++编译器：

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.20)
project(tinyplus C)

set(CMAKE_C_STANDARD 90)

add_executable(tinyplus main.c globals.h scan.c util.c scan.h util.h)
```

然后就是两部曲 cmake .和 make。cmake 生成平台专有的 Makefile，然后用平台下的 GNU make 来构建项目：

```
Tiny Scanner — -zsh — 120×40
ernest@MacBook-Pro Tiny Scanner % cmake .
-- The C compiler identification is AppleClang 13.1.6.13160021
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /Library/Developer/CommandLineTools/usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/ernest/Desktop/Tiny Scanner
ernest@MacBook-Pro Tiny Scanner % make
[ 25%] Building C object CMakeFiles/test.dir/main.c.o
/Users/ernest/Desktop/Tiny Scanner/main.c:8:10: warning: non-portable path to file '"GLOBALS.H"'; specified path differs
 in case from file name on disk [-Wnonportable-include-path]
#include "globals.h"
         ^~~~~~~~~~~
         "GLOBALS.H"
/Users/ernest/Desktop/Tiny Scanner/main.c:20:10: warning: non-portable path to file '"UTIL.H"'; specified path differs i
n case from file name on disk [-Wnonportable-include-path]
#include "util.h"
         ^~~~~~~~
         "UTIL.H"
/Users/ernest/Desktop/Tiny Scanner/main.c:21:10: warning: non-portable path to file '"SCAN.H"'; specified path differs i
n case from file name on disk [-Wnonportable-include-path]
#include "scan.h"
         ^~~~~~~~
         "SCAN.H"
3 warnings generated.
[ 50%] Building C object CMakeFiles/test.dir/scan.c.o
/Users/ernest/Desktop/Tiny Scanner/scan.c:8:10: warning: non-portable path to file '"GLOBALS.H"'; specified path differs
 in case from file name on disk [-Wnonportable-include-path]
#include "globals.h"
         ^~~~~~~~~~~
         "GLOBALS.H"
/Users/ernest/Desktop/Tiny Scanner/scan.c:9:10: warning: non-portable path to file '"UTIL.H"'; specified path differs in
 case from file name on disk [-Wnonportable-include-path]
#include "util.h"
         ^~~~~~~~
         "UTIL.H"
/Users/ernest/Desktop/Tiny Scanner/scan.c:10:10: warning: non-portable path to file '"SCAN.H"'; specified path differs i
/Users/ernest/Desktop/Tiny Scanner/scan.c:10:10: warning: non-portable path to file '"SCAN.H"'; specified path differs i
n case from file name on disk [-Wnonportable-include-path]
#include "scan.h"
         ^~~~~~~~
         "SCAN.H"
3 warnings generated.
[ 75%] Building C object CMakeFiles/tinyplus.dir/util.c.o
/Users/ernest/Desktop/Tiny Scanner/util.c:9:10: warning: non-portable path to file '"GLOBALS.H"'; specified path differs
 in case from file name on disk [-Wnonportable-include-path]
#include "globals.h"
         ^~~~~~~~~~~
         "GLOBALS.H"
/Users/ernest/Desktop/Tiny Scanner/util.c:10:10: warning: non-portable path to file '"UTIL.H"'; specified path differs i
n case from file name on disk [-Wnonportable-include-path]
#include "util.h"
         ^~~~~~~~
         "UTIL.H"
2 warnings generated.
[100%] Linking C executable tinyplus
[100%] Built target tinyplus
```

最后得到可执行文件 tinyplus：

```
ernest@MacBook-Pro Tiny Scanner % ls
CMakeCache.txt          GLOBALS.H               SCAN.C                  UTIL.H
CMakeFiles              MAIN.C                  SCAN.H                  cmake_install.cmake
CMakeLists.txt          Makefile                UTIL.C                  tinyplus
ernest@MacBook-Pro Tiny Scanner %
```

对 tiny.txt 测试如下：

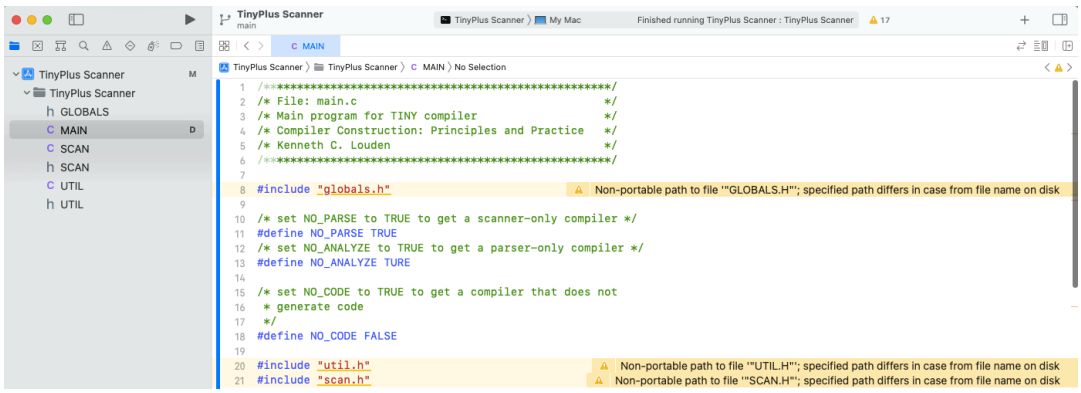
```
Tiny Scanner — -zsh — 120×49
ernest@MacBook-Pro Tiny Scanner % ./tinyplus testcode/tiny.txt

TINY COMPILATION: testcode/tiny.txt

1: {sample program in TINY language-computes factorial}
2: read x;  {input an integer}
        2: reserved word: read
        2: ID, name= x
        2: ;
3: if 0<x then  {don??t compute if x<=0}
        3: reserved word: if
        3: NUM, val= 0
        3: <
        3: ID, name= x
        3: reserved word: then
4: fact:=1;
        4: ID, name= fact
        4: :=
        4: NUM, val= 1
        4: ;
5: repeat
        5: reserved word: repeat
6: fact := fact*x;
        6: ID, name= fact
        6: :=
        6: ID, name= fact
        6: *
        6: ID, name= x
        6: ;
7: x := x-1;
        7: ID, name= x
        7: :=
        7: ID, name= x
        7: -
        7: NUM, val= 1
        7: ;
8: until x=0;
        8: reserved word: until
        8: ID, name= x
        8: =
        8: NUM, val= 0
        8: ;
9: write fact;  {output factorial of x}
        9: reserved word: write
        9: ID, name= fact
        9: ;
10: end
        10: reserved word: end
        11: EOF
```

Tiny Scanner 成功对 tiny.txt 进行词法分析。

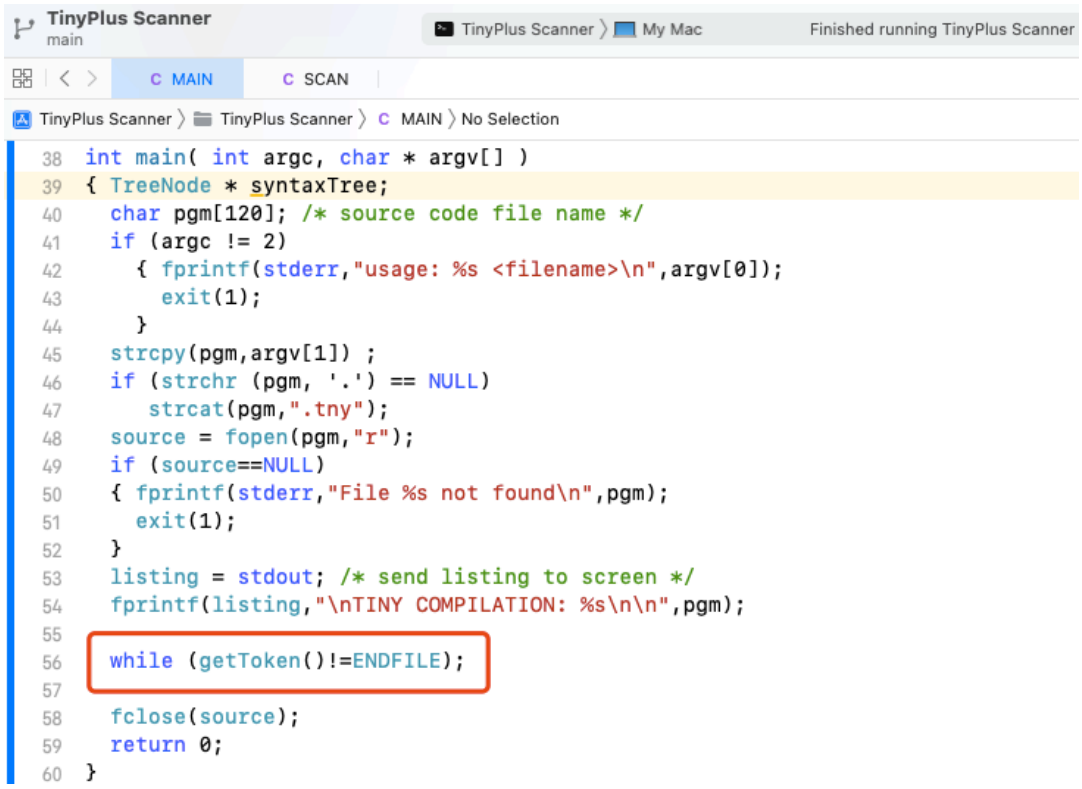
Tiny Scanner 2 Tiny+ Scanner

增加关键字（true，false，or…）

将源代码文件载入 Xcode 工程中进行二次开发。以方便查看函数之间的调用关系、跳转到函数定义和声明处等操。



在 main 函数里可以看到，词法分析的功能主要是通过一直调用 getToken 直到输入文件末尾（EOF/ENDFILE）来处理输入文件的。



GetToken 函数处正是有限状态自动机的实现之处。查看 getToken 函数的代码，发现 reversedLookup 检查 tokenString，以决定将字符串是标识符还是保留词。

```
191       if ((save) && (tokenStringIndex <= MAXTOKENLEN))
192         tokenString[tokenStringIndex++] = (char) c;
193       if (state == DONE)
194       { tokenString[tokenStringIndex] = '\0';
195         if (currentToken == ID)
196           currentToken = reservedLookup(tokenString);
197       }
```

最终由 reversedWords 数组判断：

```
55 /* lookup table of reserved words */
56 static struct
57     { char* str;
58       TokenType tok;
59     } reservedWords[MAXRESERVED]
60    = {{"if",IF},{"then",THEN},{"else",ELSE},{"end",END},
61       {"repeat",REPEAT},{"until",UNTIL},{"read",READ},
62       {"write",WRITE}};
```

修改该数组，增加保留词数量以及 TokenType 的枚举类型。注意到这里对保留词采用的是“一符一类”的分类方式。首先为 TokenType 添加枚举类型：

```
28 typedef enum
29     /* book-keeping tokens */
30    {ENDFILE,ERROR,
31     /* reserved words */
32     IF,THEN,ELSE,END,REPEAT,UNTIL,READ,WRITE,
33     _TRUE,_FALSE,OR,AND,NOT,INT,_BOOL,STRING,FLOAT,
34     DOUBLE,DO,WHILE,INCLUDE,BREAK,CONTINUE,
```

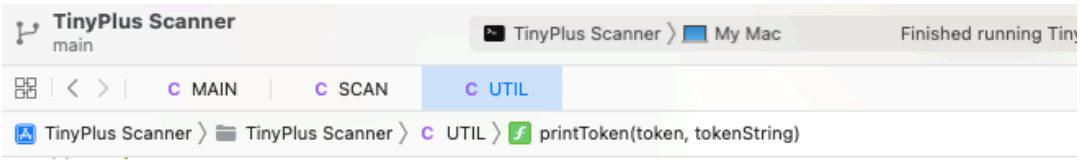
有一些拼写刚好也是 C 语言的保留词，因此加下划线以避免冲突。然后就是修改 reverWords 结构体数组的实例：

```
/* lookup table of reserved words */
static struct
    { char* str;
      TokenType tok;
    } reservedWords[MAXRESERVED]
   = { {"if",IF},{"then",THEN},{"else",ELSE},{"end",END},
       {"repeat",REPEAT},{"until",UNTIL},{"read",READ},
       {"write",WRITE},{"true",_TRUE},{"false",_FALSE},
       {"or",OR},{"and",AND},{"not",NOT},{"int",INT},
       {"bool",_BOOL},{"string", STRING},{"float",FLOAT},
       {"double",DOUBLE},{"do",DO},{"while",WHILE},
       {"include", INCLUDE},{"break",BREAK},{"continue", CONTINUE}
   };
```

注意相关联的宏 MAXRESERVED 也要从 8 修改为 23

```
/* MAXRESERVED = the number of reserved words */
#define MAXRESERVED 23
```

此时程序已经可以识别新增加的保留词了，不过为了能顺利输出结果，需要修改 printToken 函数增加对应的 case：



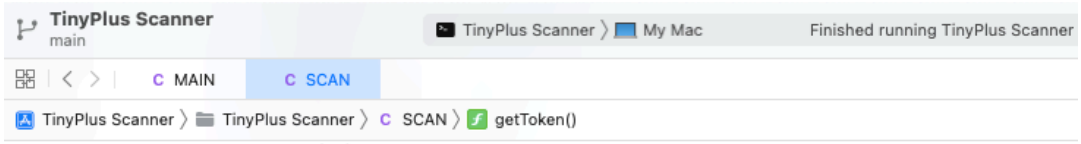

```
void printToken( TokenType token, const char* tokenString )
{ switch (token)
  { case IF:
    case THEN:
    case ELSE:
    case END:
    case REPEAT:
    case UNTIL:
    case READ:
    case WRITE:
      case _TRUE:
      case _FALSE:
      case OR:
      case AND:
      case NOT:
      case INT:
      case _BOOL:
      case STRING:
      case FLOAT:
      case DOUBLE:
      case DO:
      case WHILE:
      case INCLUDE:
      case BREAK:
      case CONTINUE:
      fprintf(listing,
         "reserved word: %s\n",tokenString);
      break;
```

此时已经可以顺利地词法分析新增加的保留词了。

增加特殊符号（>, <=, >=…）

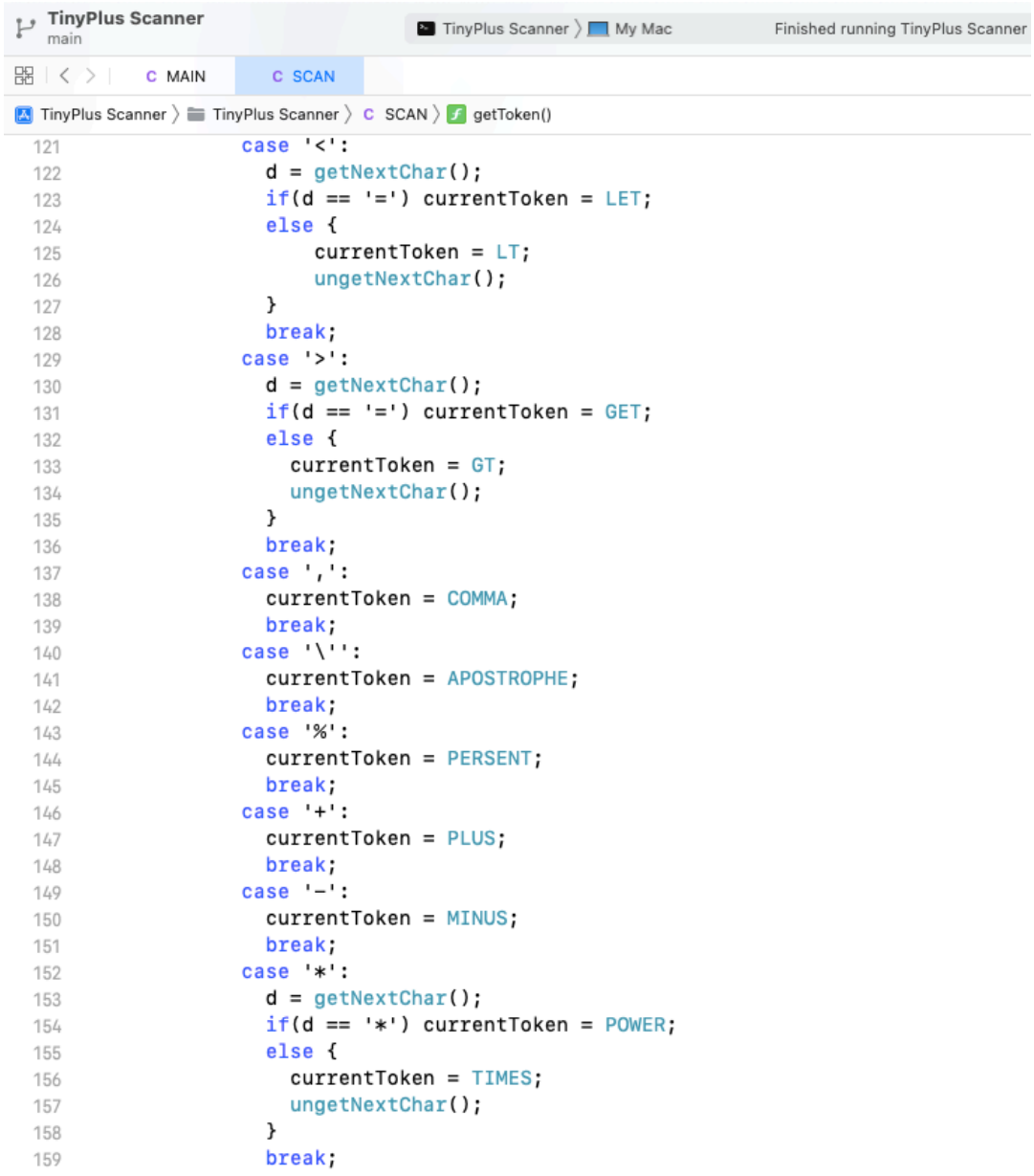
```
typedef enum
    /* book-keeping tokens */
   {ENDFILE,ERROR,
    /* reserved words */
    IF,THEN,ELSE,END,REPEAT,UNTIL,READ,WRITE,
    _TRUE,_FALSE,OR,AND,NOT,INT,_BOOL,STRING,FLOAT,
    DOUBLE,DO,WHILE,INCLUDE,BREAK,CONTINUE,
    /* multicharacter tokens */
    ID,NUM,
    /* special symbols */
    ASSIGN,EQ,LT,PLUS,MINUS,TIMES,OVER,LPAREN,RPAREN,SEMI,
    GT,LET,GET,COMMA,APOSTROPHE,PERSENT,POWER
   } TokenType;
```

添加 TokenType 的枚举类型。在 getToken 函数里面添加对这些符号的识别：

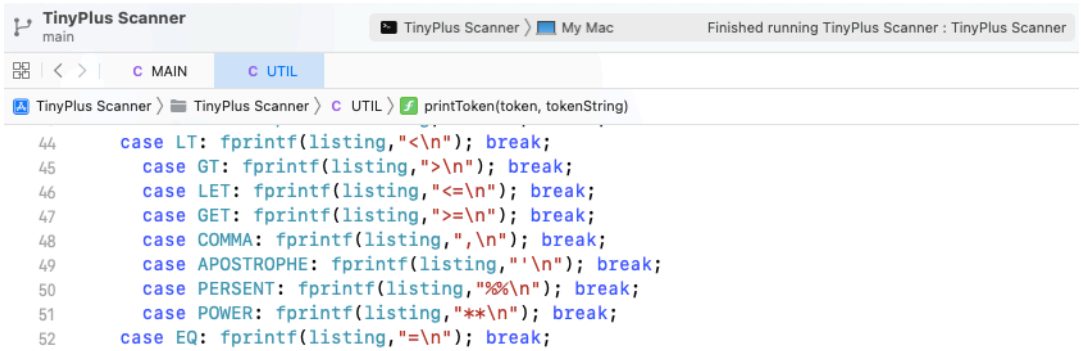



```
121                 case '<':
122                   d = getNextChar();
123                   if(d == '=') currentToken = LET;
124                   else {
125                       currentToken = LT;
126                       ungetNextChar();
127                   }
128                   break;
129                 case '>':
130                   d = getNextChar();
131                   if(d == '=') currentToken = GET;
132                   else {
133                     currentToken = GT;
134                     ungetNextChar();
135                   }
136                   break;
137                 case ',':
138                   currentToken = COMMA;
139                   break;
140                 case '\'':
141                   currentToken = APOSTROPHE;
142                   break;
143                 case '%':
144                   currentToken = PERSENT;
145                   break;
146                 case '+':
147                   currentToken = PLUS;
148                   break;
149                 case '-':
150                   currentToken = MINUS;
151                   break;
152                 case '*':
153                   d = getNextChar();
154                   if(d == '*') currentToken = POWER;
155                   else {
156                     currentToken = TIMES;
157                     ungetNextChar();
158                   }
159                   break;
```

值得注意的是，对于一些双目特殊符号“>=”,“<=”和“**”，需要使用函数 getNextChar 来查看下个字符。如果下一个字符不能与当前字符构成双目运算符，则使用 ungetNextChar 来回退此字符。



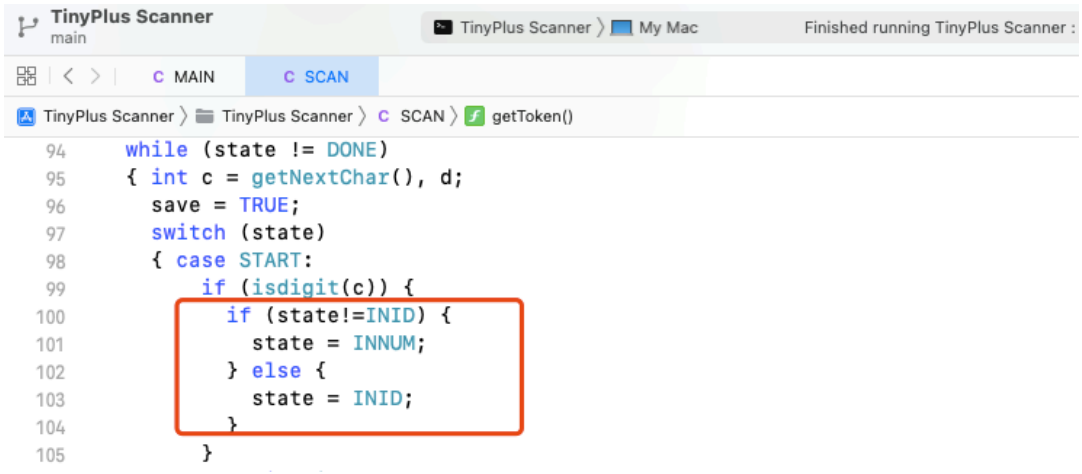
最后修改 printToken 函数增加对应的 case：



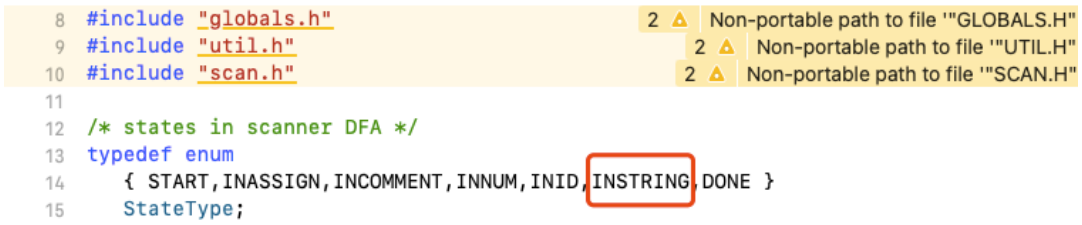
此时已经可以顺利地词法分析新增加的特殊符号了。

标识符、整数类型和字符串类型的自动机实现

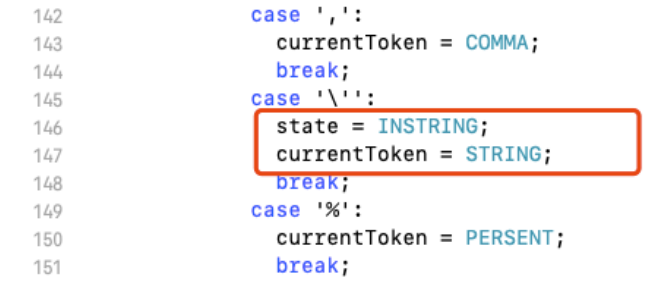
TINY 目前只支持识别不包含阿拉伯数字的标识符。通过修改 getToken 函数的代码，使得 TINY+能分析识别第一个字符不为数字的带阿拉伯数字的标识符：



当识别到数字时，根据上一个状态（state）决定将状态设置为标识符（ID）还是整数类型（ISNUM）。增加处于字符串识别的状态（state, INSTRING）：

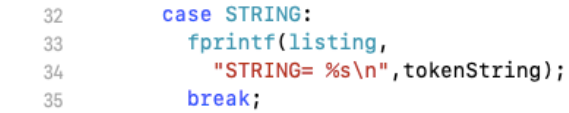


然后在第一次识别到单引号“’”时，进入识别字符串的状态（INSTRING）：



接下来对于任何非单引号的字符，全部识别为字符串的一部分。而当再次遇见单引号的时候，代表已到达字符串的末端。此时将状态机的 state 设置为 DONE 表示此字符串已经识别完成。

然后就是修改输出函数 printToken：

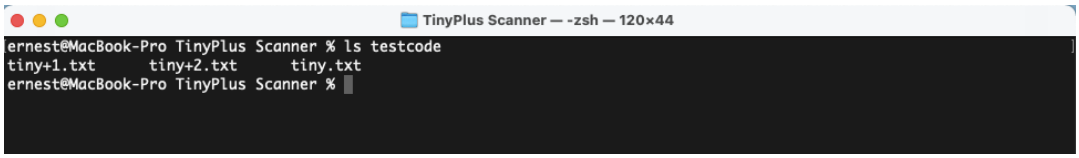


为了以保留字中的 string 区分。对于保留词 string 的 TokenType 修改为_STRING。而字符串类型变量的 TokenType 则单纯使用 STRING 表示。

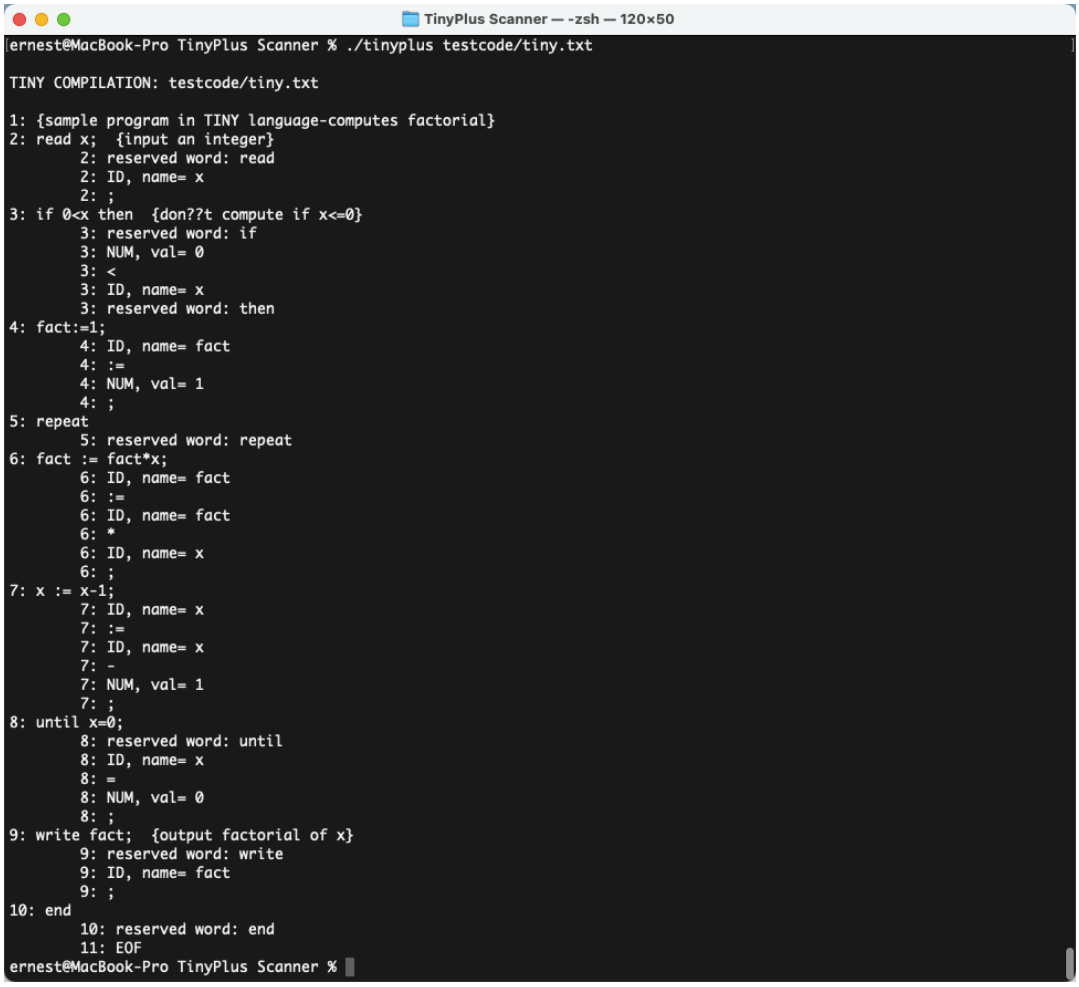
此时 TINY+ SCANNER 已经可以词法分析字符串类型变量了。

实验结果：

提供的测试文件夹中有三个测试文件，如下：

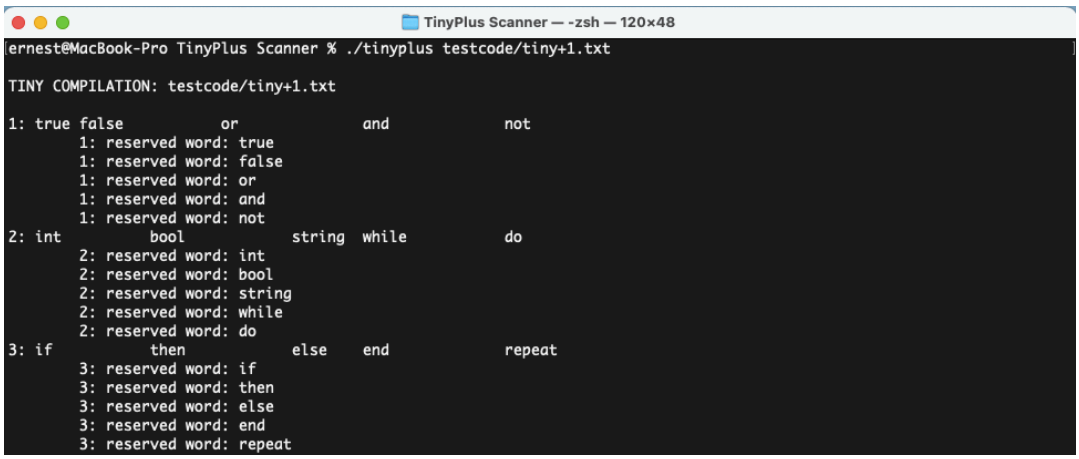


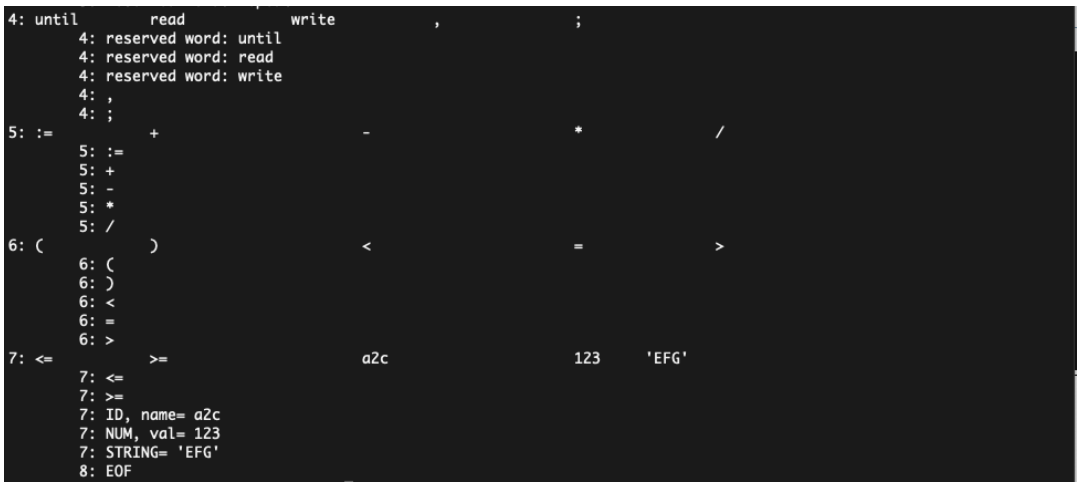
tiny.txt 的测试结果如下：



可以看到，词法分析器成功地解析 tiny.txt

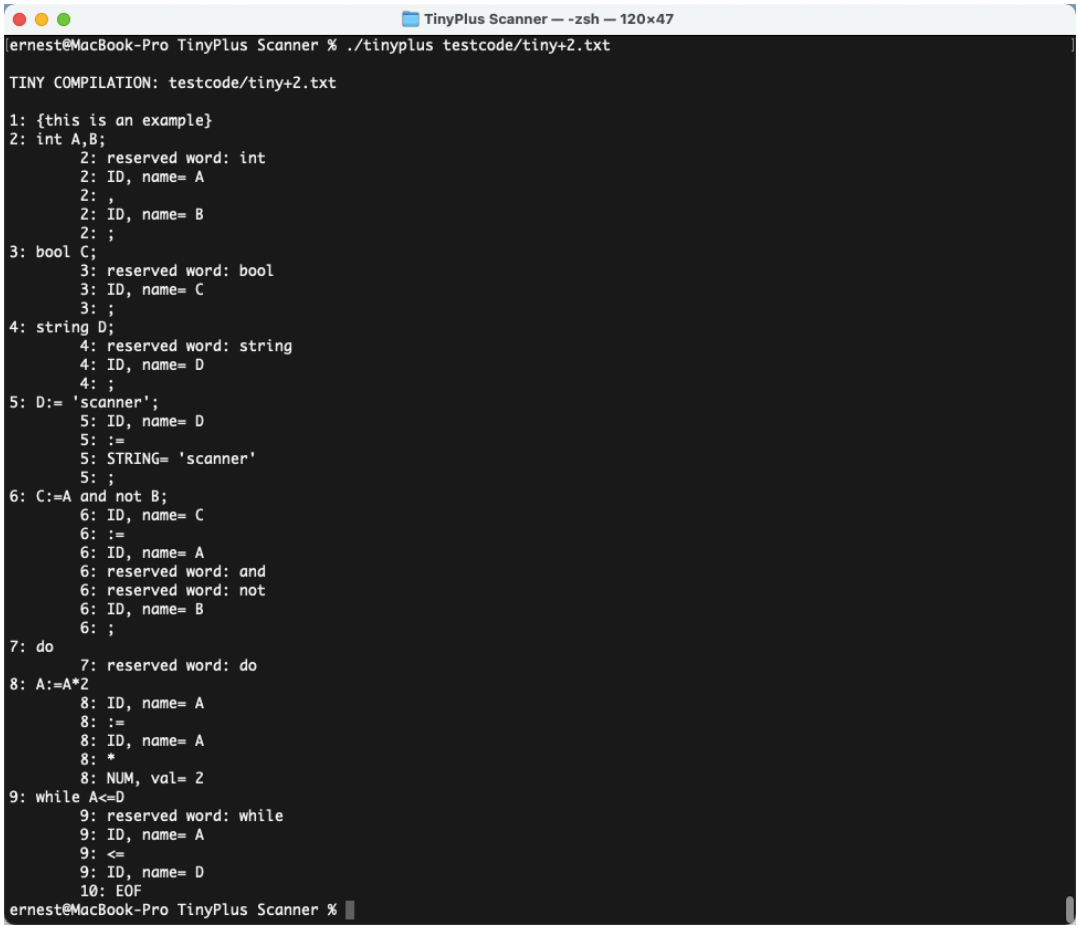
tiny+1.txt 的测试结果如下：





可以看到，词法分析器成功地解析 tiny+1.txt

tiny+2.txt 的测试结果如下：



可以看到，词法分析器成功地解析 tiny+2.txt

心得体会：

通过阅读 tiny Scanner 的源代码，加深了我对简单词法分析器的理解。与此同时，将有限状态自动机应用到实际的开发环境中还是与理论有一定距离的。拓展 tiny Scanner 成为 tiny+ Scanner 使得它能够识别更多的字符串、分析更多的数据类型，本质上还是在于阅读代码能力以及对词法分析原理的理解。

指导教师批阅意见：   成绩评定：   指导教师签字： 年　　月　　日
备注：