编译原理实验-FLEX+BISON+DEV C++完美解决方案

原创

davendw 于 2017-11-12 02:07:49 发布 6257 收藏 20

分类专栏： 编译原理 文章标签： 编译原理 flex c语言 bison yacc

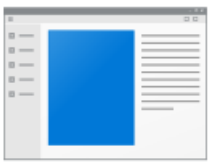
本文链接：https://blog.csdn.net/qq_33380032/article/details/78509805

一点介绍

flex，前身是lex，lex是1975年由Mike Lesk和当时尚在AT&T实习的Eric Schmidt共同完成的基于UNIX环境的词法分析器的生成工具。这个lex很有名气，但是无奈效率太低加上有bug，让人用的很不爽。后来伯克利实验室的Vern Paxson用C重新写了lex，并命名为flex（Fast Lexical Analyzer Generator）。

Bison，bison的前身是传说中的yacc，yacc是由贝尔实验室的S.C.Johnson基于Knuth大神的LR分析技术，于1975～1978年写成。1987年 UC Berkeley 的Bob Corbett在BSD下重写了yacc。在后来GNU project接管了项目，添加了很多特性，形成了今天的GNU Bison语法分析器生成工具。

这里提供的bison和flex的程序长这个样子：



实验噱头

看着很多人拿到老师的文档按部就班的做了，可是……很迷

下面给大家展示一下爽的一笔的实验过程：

第一步

先看下老师的指示和相应的文件：

- Main.c, lex.yy.c,util.c
- Globals.h, util.h,scan.h
- 设置VC++项目的环境

使用bison

- 将bison.exe和tiny.y复制到同一个目录下，并切换到该目录
- bison-d tiny.y生成语法分析程序tiny.tab.c和tiny.tab.h
- 修改YACC目录中的那个Globals.h使其包含的y.tab.h改为生成的tiny.tab.h
- 再如前新建一个工程把语法分析要用到的程序加入后生成词法分析程序的exe
  - Main.c, lex.yy.c,util.c,tiny.tab.c
  - Globals.h(选择YACC目录中的那个），util.h,scan.h,parse.h,tiny.tab.h

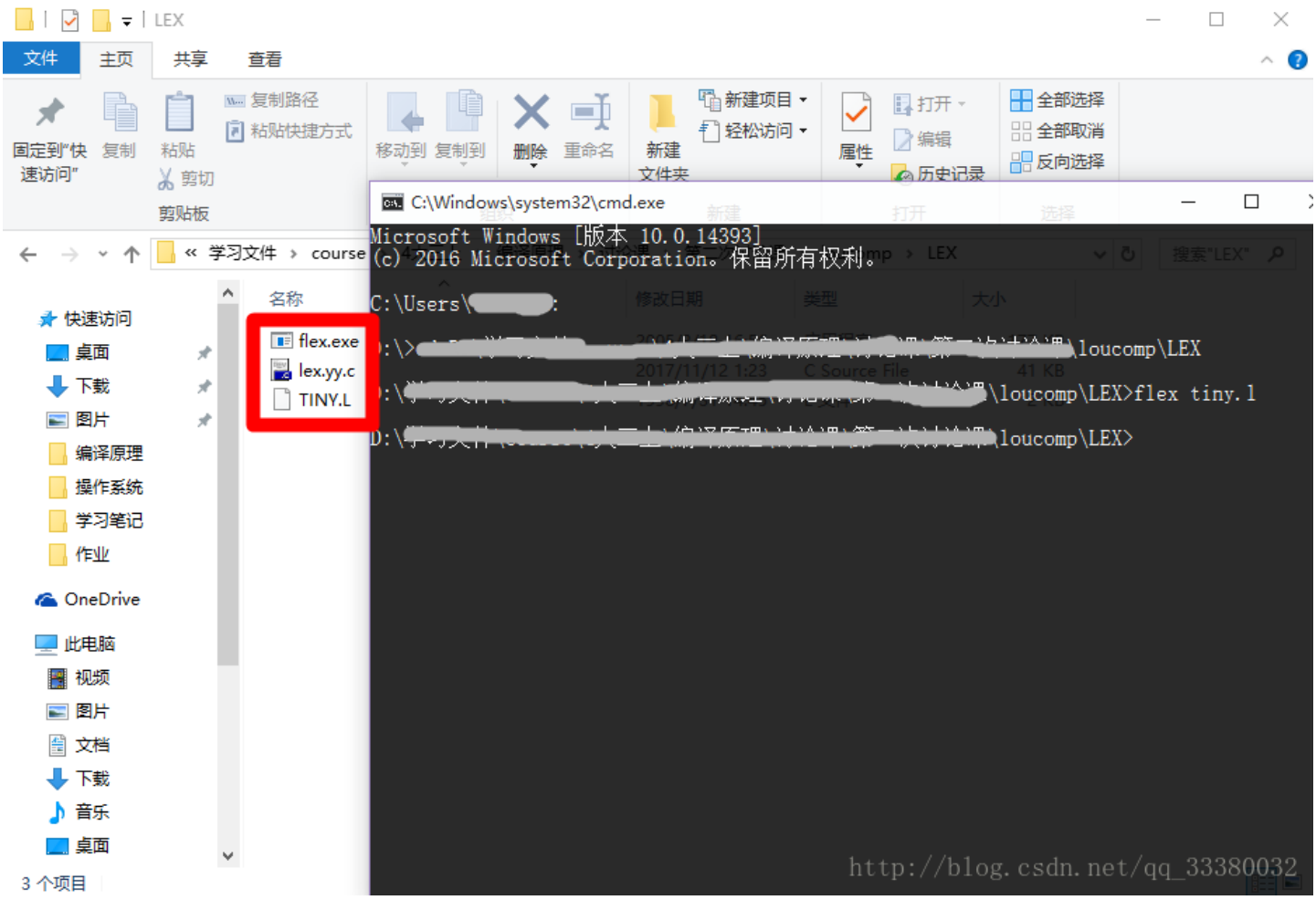
文件呢，这里有前言说的flex和bison工具还有现有的tiny编译器的一些工具

LEX	1998/7/29 16:56	文件夹	生成词法分析器和语法分析器	
YACC	1998/7/29 16:56	文件夹		
ANALYZE.H	1998/8/1 14:01	C Header File		1 KB
CGEN.H	1998/8/1 14:01	C Header File		1 KB
CODE.H	1998/8/1 14:01	C Header File		3 KB
GLOBALS.H	1998/8/1 14:01	C Header File		3 KB
PARSE.H	1998/8/1 14:01	C Header File		1 KB
SCAN.H	1998/8/1 14:01	C Header File		1 KB
SYMTAB.H	1998/8/1 14:01	C Header File		1 KB
UTIL.H	1998/8/1 14:01	C Header File		2 KB
ANALYZE.C	1998/8/1 14:02	C Source File		5 KB
CGEN.C	1998/8/1 14:02	C Source File		7 KB
CODE.C	1998/8/1 14:02	C Source File		3 KB
MAIN.C	1998/8/1 14:02	C Source File		3 KB
PARSE.C	1998/8/1 14:02	C Source File		6 KB
SCAN.C	1999/8/4 16:05	C Source File		6 KB
SYMTAB.C	1998/8/1 14:02	C Source File		4 KB
TM.C	1998/8/1 14:02	C Source File		17 KB
UTIL.C	1998/8/1 14:02	C Source File		5 KB
README.DOS	1998/7/31 15:15	DOS 文件		2 KB
SAMPLE.TM	1998/7/31 16:47	TM 文件	测试用	1 KB
SAMPLE.TNY	1996/8/25 15:33	TNY 文件		1 KB
MAKEFILE	1998/2/3 22:29	文件		2 KB
TINY.EXE	1998/4/26 21:47	应用程序	成功之后	40 KB
TM.EXE	1998/3/20 13:40	应用程序		14 KB

第二步

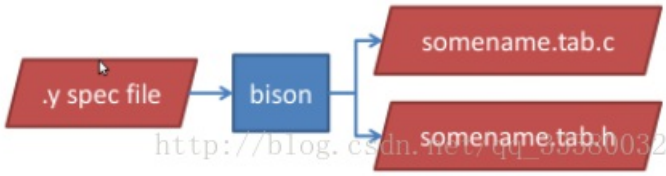
进行flex生成词法分析器操作：

将flex放到tiny.l旁边，命令行执行，生成lex.yy.c

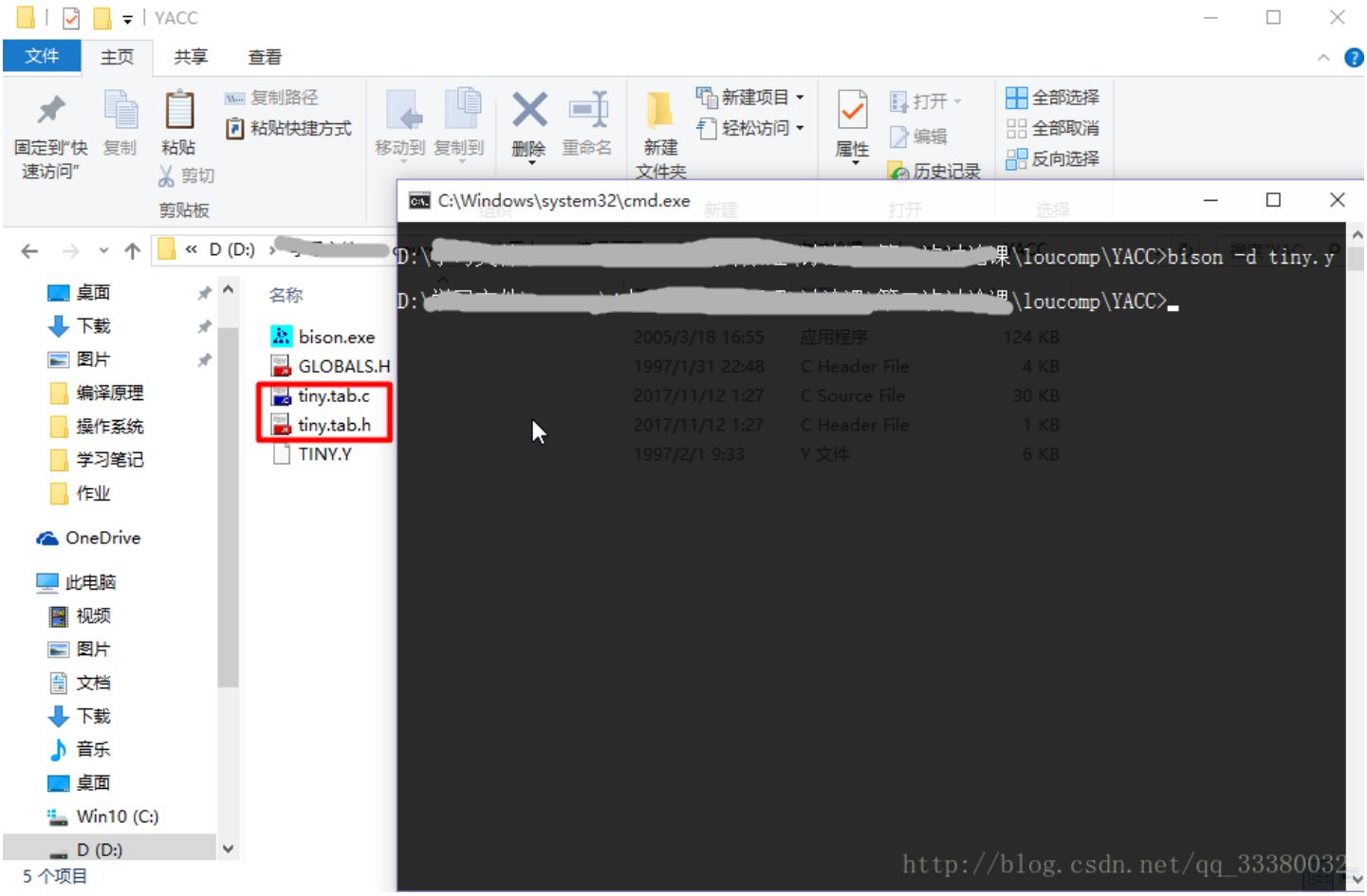


第三步

进行bison生成语法分析器操作：

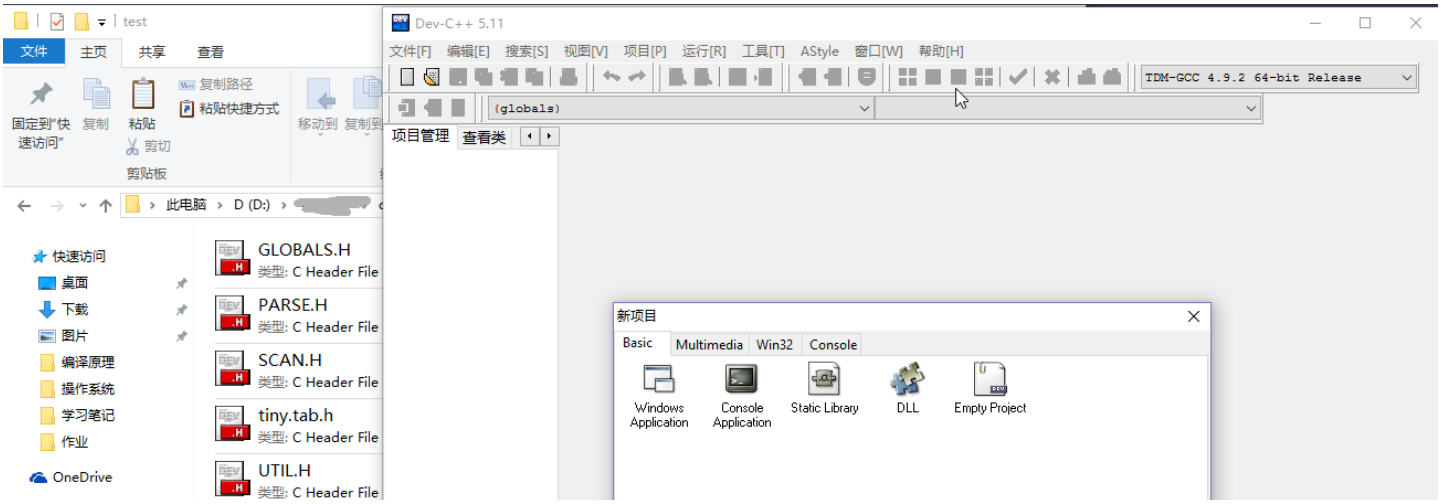


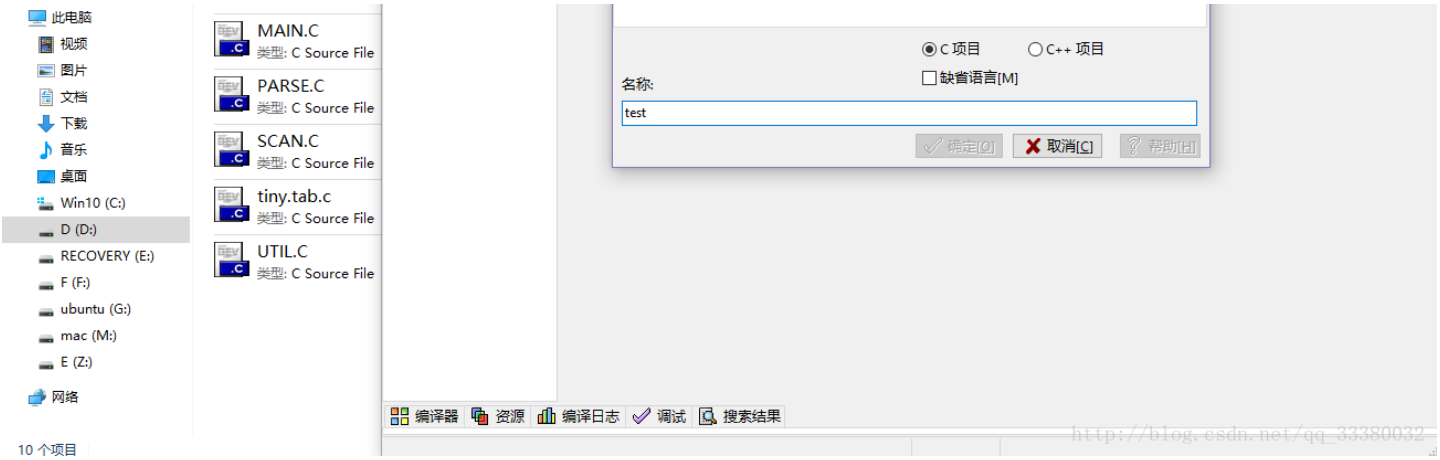
将bison放到tiny.y旁边，命令行执行，生成tiny.tab.h&tiny.tab.c



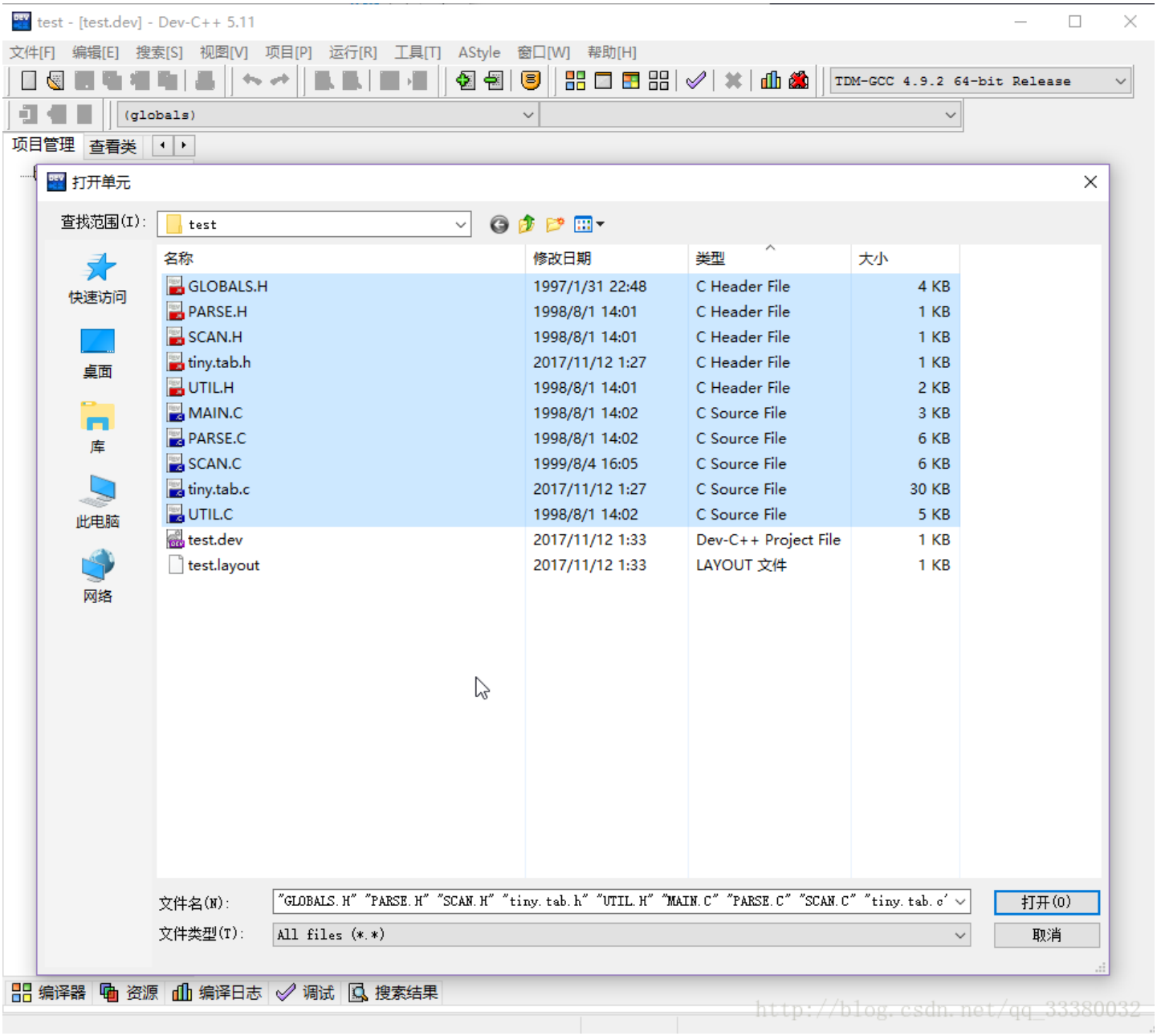
第四步

构建工程，将所需要的所有文件放到一个文件夹中（可以忽略，个人强迫症）【注意，如果选了.h就一定要带上.c，老师的文件有缺漏】





构建工程【easy不说，最好是c项目+Console Application】——记住flex生成的文件也要放进来，图中忘记截图的

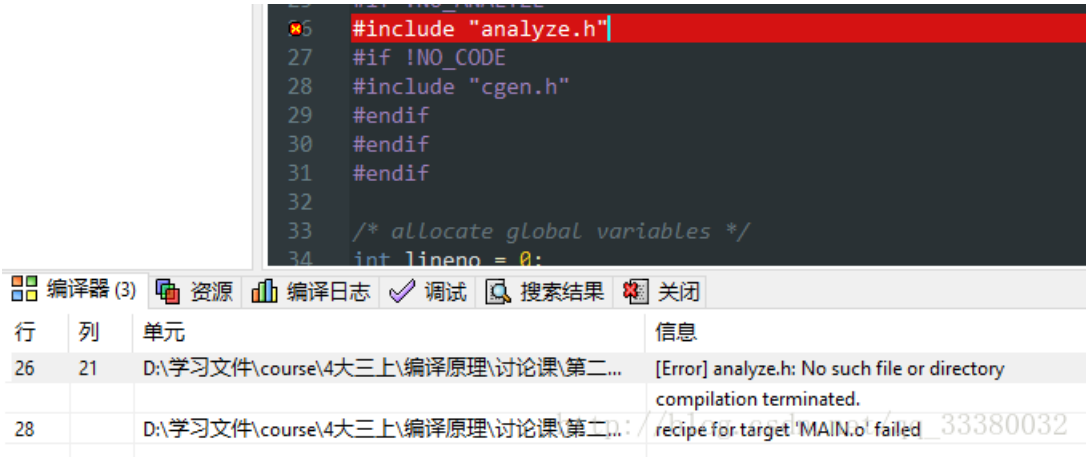


之后！

打开global.h

> 将位于第30行附近的#include “y.tab.h”换成#include “tiny.tab.h”

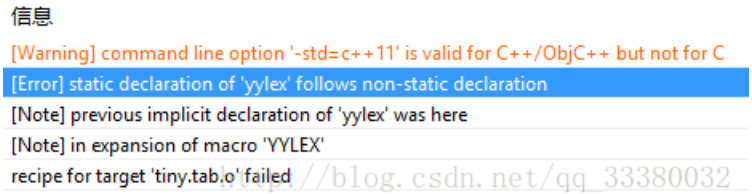
进行第一次编译！
编译之后会是这样的错误：



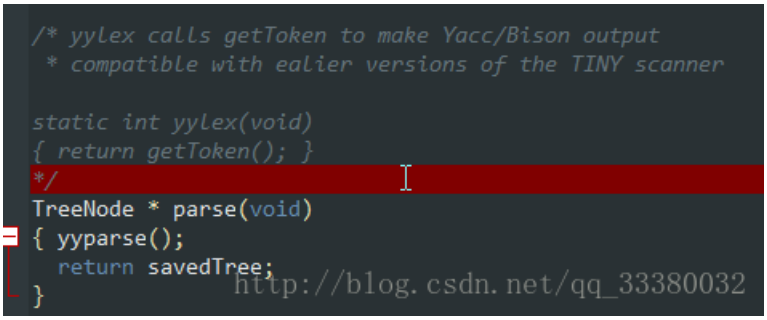
这是因为我们没有拷贝词法语法分析包（因为我们使用的flex和bison生成的，而不是tiny编译器自己的）
这里只需要将analyze的开关关上即可。

```
/* set NO_PARSE to TRUE to get a scanner-only compiler */
#define NO_PARSE FALSE
/* set NO_ANALYZE to TRUE to get a parser-only compiler */
#define NO_ANALYZE TRUE
```

进行第二次编译（第一次出现很多warning和错误，再编译一次让warning消停会）
这个时候会有这样的错误



还是不要方，这里的问题是yylex的重复定义，你们还记得书上是怎么要求yacc的吗，最后面一定要有yylex，但是我们的这些个代码中已经在词法分析器的时候就已经设计好可yylex，解决方法很多，最简单粗暴的就是将tiny.tab.c中的yylex（最下面）给删了：



这个时候进行第三次编译，错误将会变成：

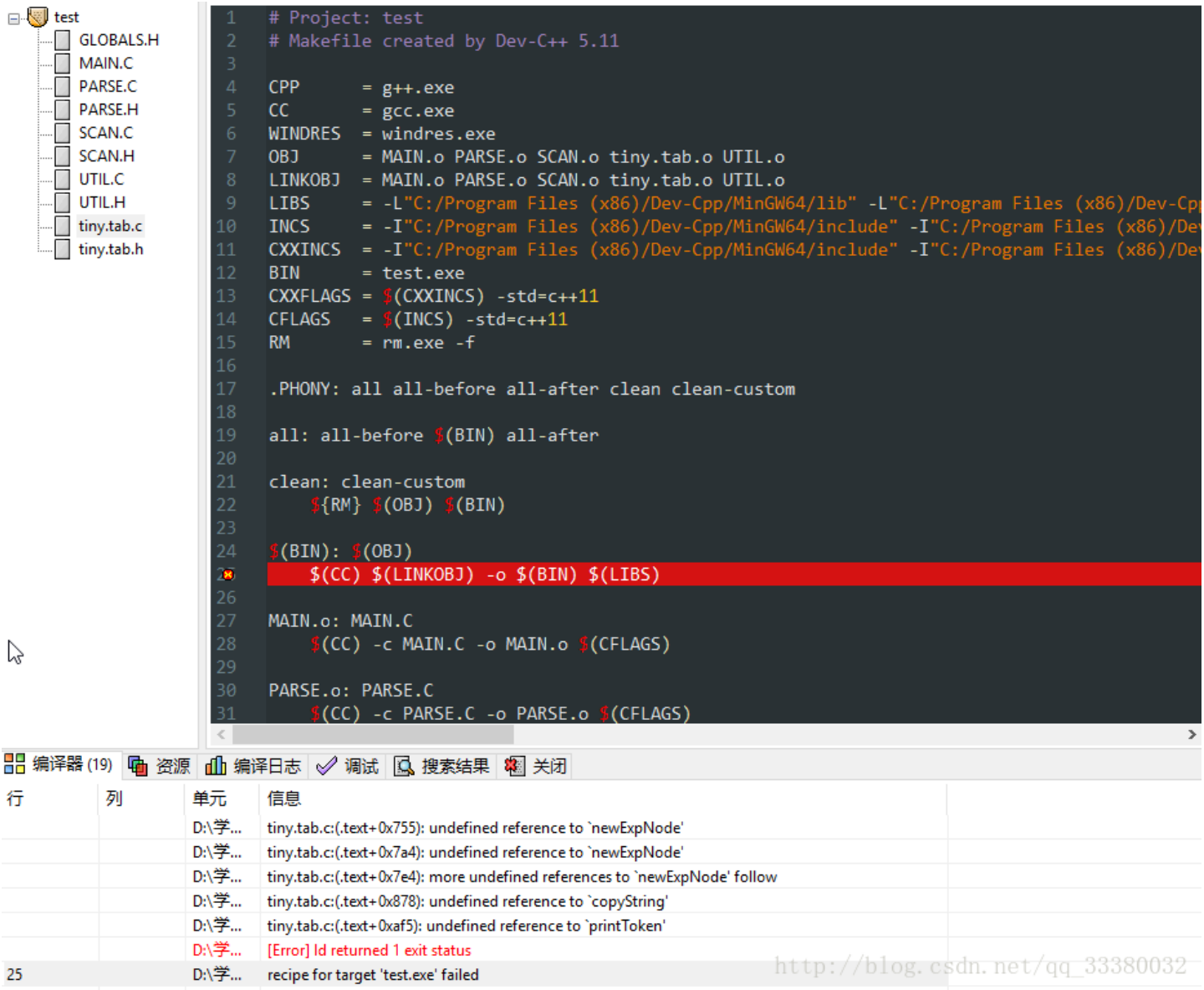
```
103     fprintf(listing,"Out of memory error at line %d\n",lineno);
104   else strcpy(t,s);
105   return t;
106 }
107
108 /* Variable indentno is used by printTree to
```

编译日志 | 调试 | 搜索结果 | 关闭

单元	信息
D:\学...	[Warning] command line option '-std=c++11' is valid for C++/ObjC++ but not for C
D:\学...	**In function 'char* copyString(char*)':**
D:\学...	[Error] invalid conversion from 'void*' to 'char*' [-fpermissive]
D:\学...	**At global scope:**
D:\学...	[Error] 'indentno' does not name a type
D:\学...	**In function 'void printSpaces()':**
D:\学...	[Error] 'indentno' was not declared in this scope
D:\学...	**In function 'void printTree(TreeNode*)':**

这里是一些类型强制转化的问题，和IDE有关，DEV C++貌似都有这个问题codeblocks等没有，所以大家可以开心进行最后这个讲得通的修改吧~

改完之后进行第四次编译出现这个爽歪歪的错误：



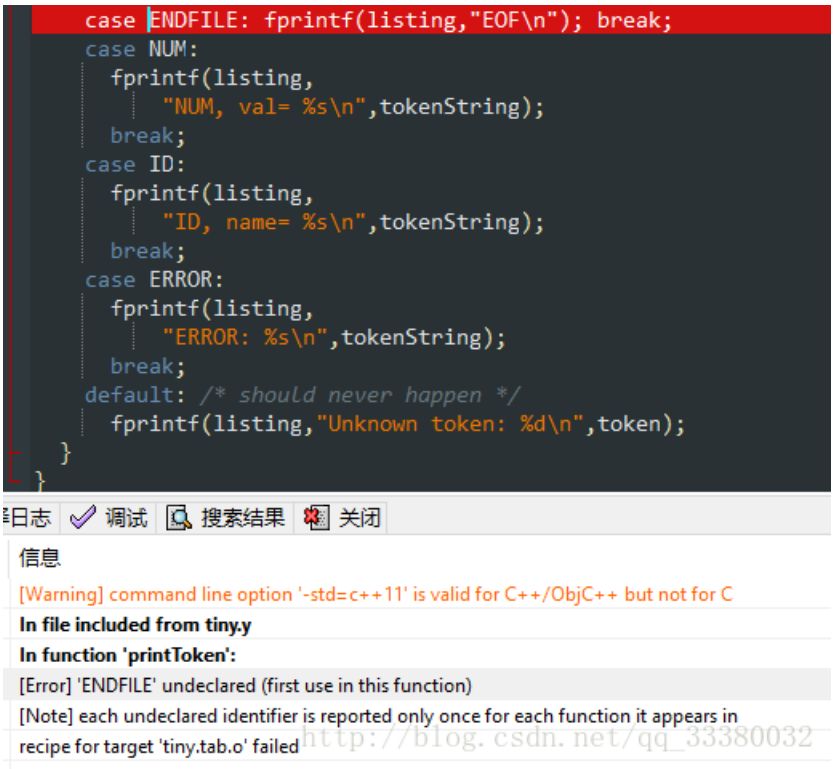
这里就开始解释不通了，这错误我们读一下，大概是有一些函数没有成功的声明，由于是多文件的编程，我们看到tiny.tab.c文件中去：

```
#include "globals.h"
#include "util.h"
#include "scan.h"
#include "parse.h"
```

引入的是这几个文件，发现出错的函数都是在这里

```
#include "util.h"
```

我们神奇的将这个.h改成.c
回到main函数，编译第五遍：
还是错的，但是错变成这样了：



说没有define那我们……就再define一下吧

```
#include "globals.h"
#include "util.h"
#define ENDFILE 0
/* Procedure printToken prints a token
 * and its lexeme to the listing file
 */
void printToken( TokenType token, const char* tokenString )
```

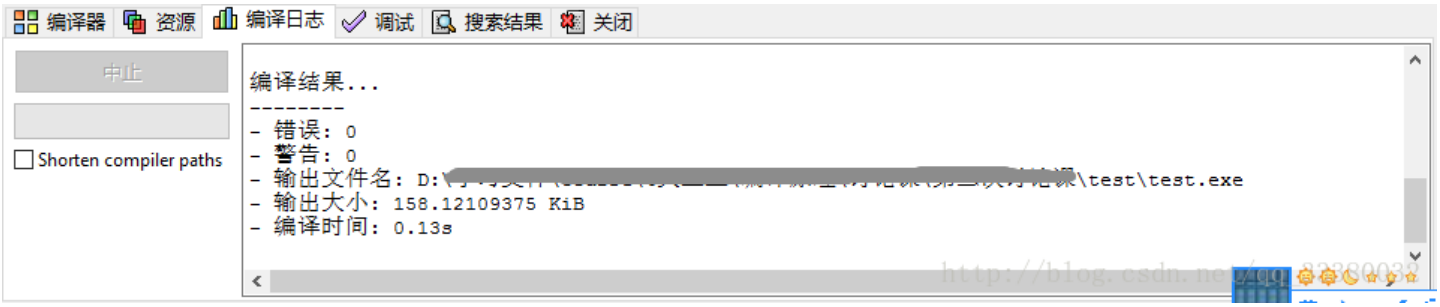
再一次的编译：

```
信息
[Warning] command line option '-std=c++11' is valid for C++/ObjC++ but not for C
lex.yy.c:(.text+0x4d6): undefined reference to `yywrap'
lex.yy.c:(.text+0xd51): undefined reference to `yywrap'
[Error] ld returned 1 exit status
recipe for target 'test.exe' failed
```

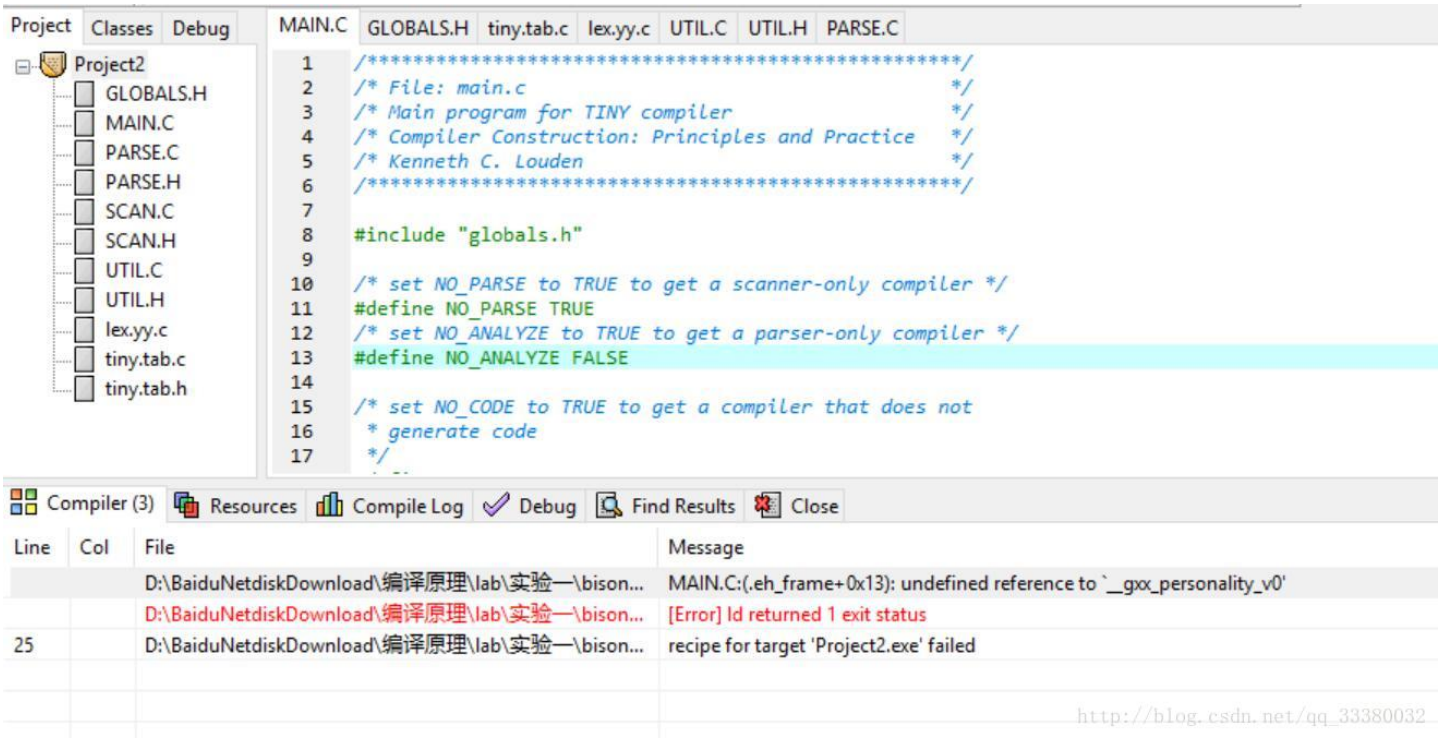

我们发现这个yywrap的可恶，不过之前的实验是自己做的同学理论上也是会遇到这个问题的，直接去yy.lex.c文件中找到注释中yywrap的位置，声明一下就好了，让这个函数有返回值就好！

```
#define yywrap() 1
/* Flag which is used to allow yywrap()'s to do buffer switches
 * instead of setting up a fresh yyin.  A bit of a hack ...
 */
```

大概是222行
这时候再一次的编译：



我还能说什么……
就这样……
结束了……
不过，如果幸运，你的电脑还会遇到这个问题：



这个时候……转战codeblocks吧

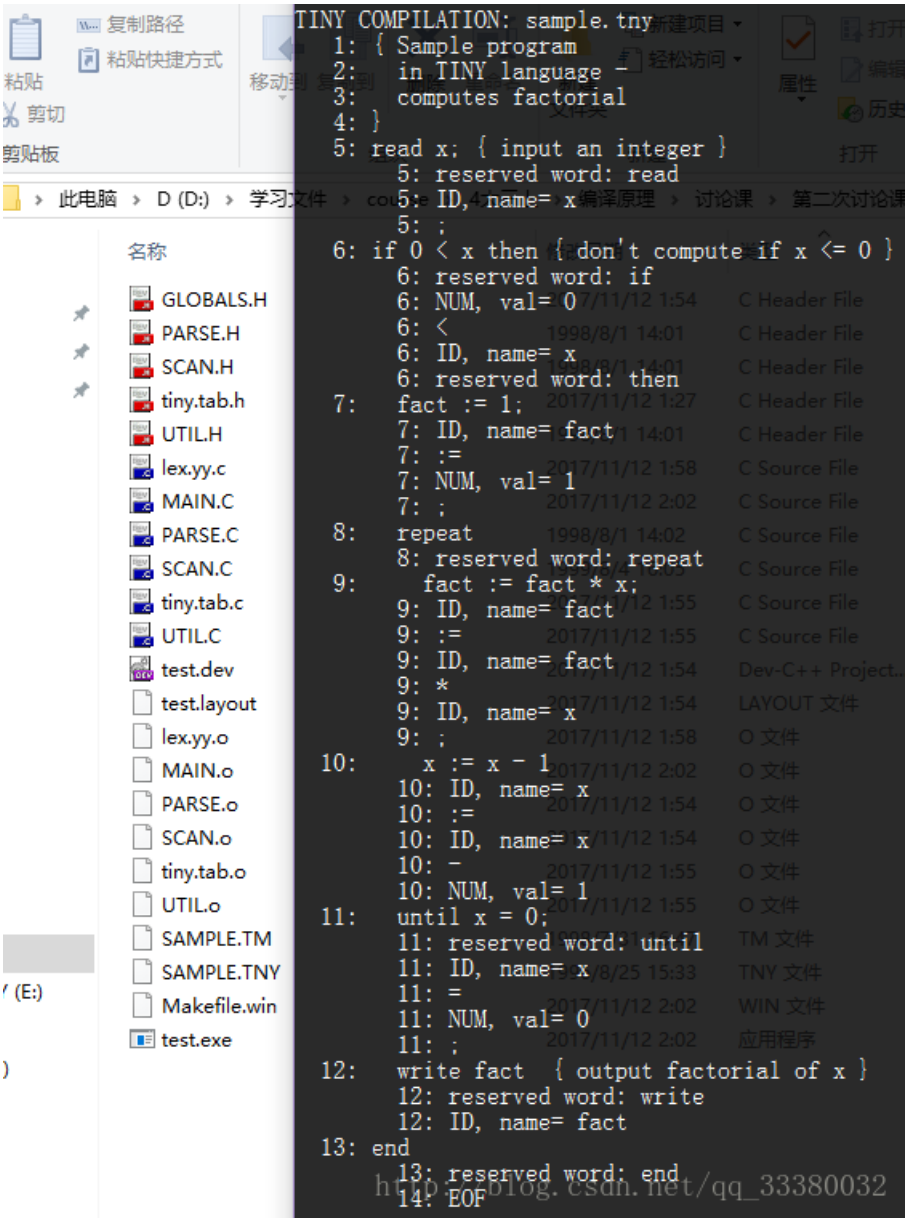
第五步

下面我们来运行一下！~

忘了！
要到main中打开这些开关

```
    /* allocate and set tracing flags */
    int EchoSource = TRUE;
    int TraceScan = TRUE;
    int TraceParse = FALSE;
    int TraceAnalyze = FALSE;
    int TraceCode = FALSE;
```

编译运行，然后，命令行执行生成的exe，和sample



大功告成

至于解释那个.h换.c

论坛里有人说是因为……重名定义了，然后如果直接引用实现，表示声明这个引用才是唯一引用。

最后我们看下原理：

就不详细说了，等我下次更新编译原理板块再说吧