词法分析程序测试方案

1. 测试目标

能够实现词法分析程序的基本功能；能够输出原测试文件内容、词法分析的结果和所有的错误信息，错误信息包括错误所在行数和错误类型；能够输出识别的文件信息、时间和错误数。

1. 测试环境

操作系统：macOS Mojave

编辑器：vs code

代码类型：C++

1. 程序说明

代码放在Source Code，分别是LexicalAnalysis.h和main.cpp两个文件。LexicalAnalysis.h实现了与词法分析有关的函数，main.cpp实现了整个tokenOut.txt的大体框架。

代码的编译和运行通过命令行终端实现，先是进入到Source Code文件夹，然后运行以下两个命令



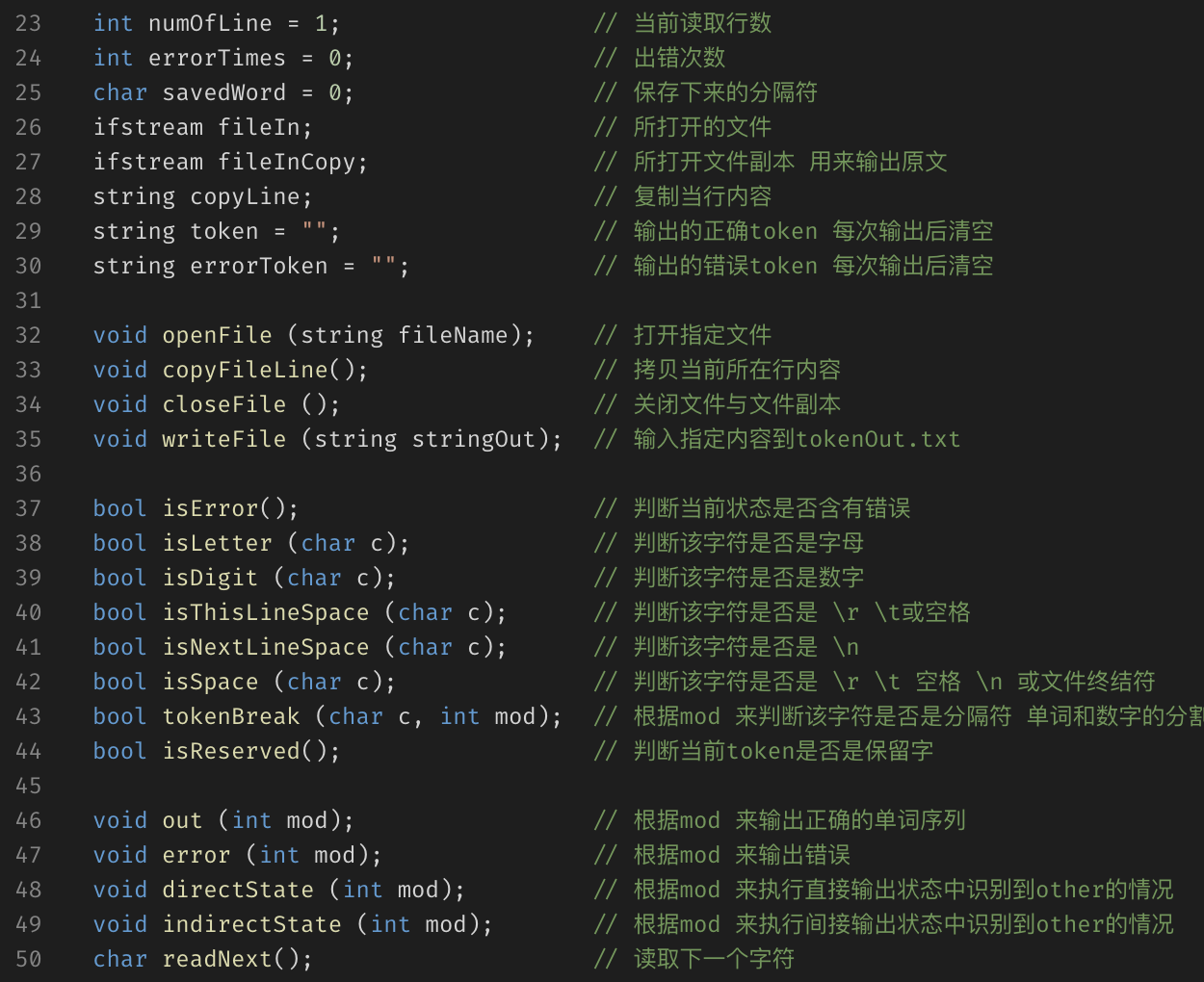
分别是使用g++编译器编译生成Unix可执行文件LexicalAnalysis，并且**运行该可执行文件，后面需要附上所要执行的文件名称作为main函数的参数，才可对所选文件进行词法分析，如果没有参数，则程序会运行错误。**

可以在命令行看到有输出，并且在文件夹有tokenOut.txt的生成，具体输出和内容会在测试内容中进行展示。

main.cpp的部分截图如下



LexicalAnalysis.h的部分截图如下



可以看待代码的编写过程满足C++编程规范，以及写有相应的注释

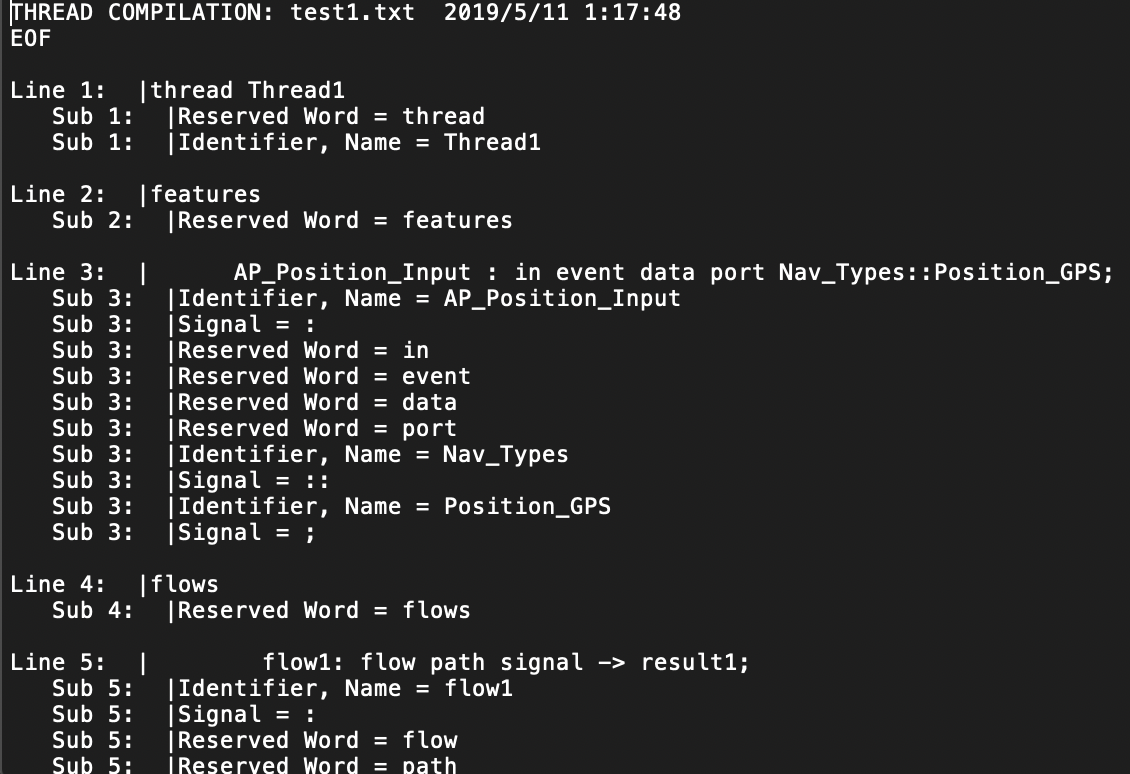
1. 测试内容
2. **实验所提供的测试文件**
3. 使用测试文件test1.txt

命令行所运行代码和输出如下



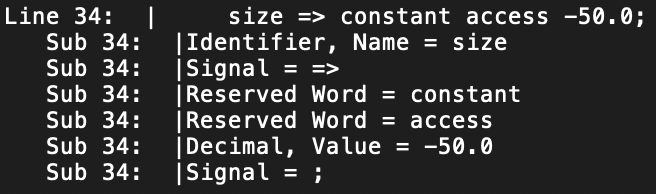
输出内容为所检测到的词法分析错误为0。

接下来打开所生成的tokenOut.txt文件，**具体文件会在“词法分析程序实验提交/附件/代码测试结果/test1”给出**，以下是部分截图。

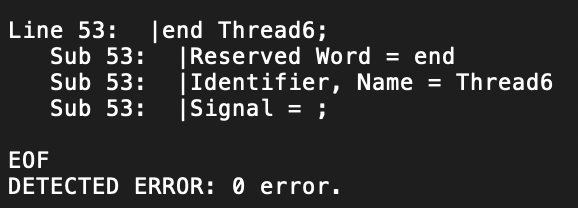


在文件开头的一二行中，可以看到这是面向线程的词法分析程序，并且有标注所执行的文件名称、词法分析的时间和EOF的文件开始读取标志，意思是在这之后就是词法分析的结果。

可以看到，在词法分析结果中，无缩进的是当前所扫描行的全部内容，有缩进的是该行内的token分析。对比着来看的话，能够重现当前的扫描内容，从而更好地判断token分析结果是否有误。



在Line 34一行中，可以看到已经实现了对空白的忽略、特定符号的识别、标识符的识别、保留字的识别和浮点数的识别，实现了实验所要求的词法分析基本功能。



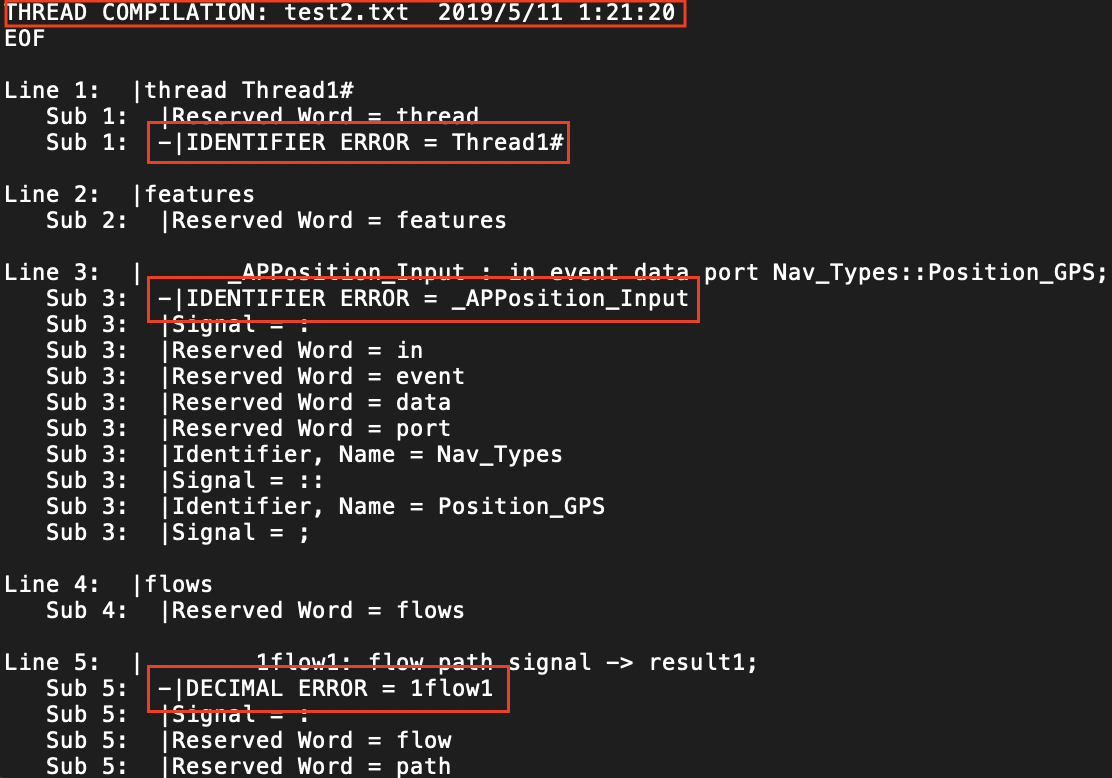
在tokenOut.txt文件的最后，能够看到第二个象征着词法分析结束的EOF标志，并且在最后会给出所检测到的错误总量。

1. 使用测试文件test2.txt

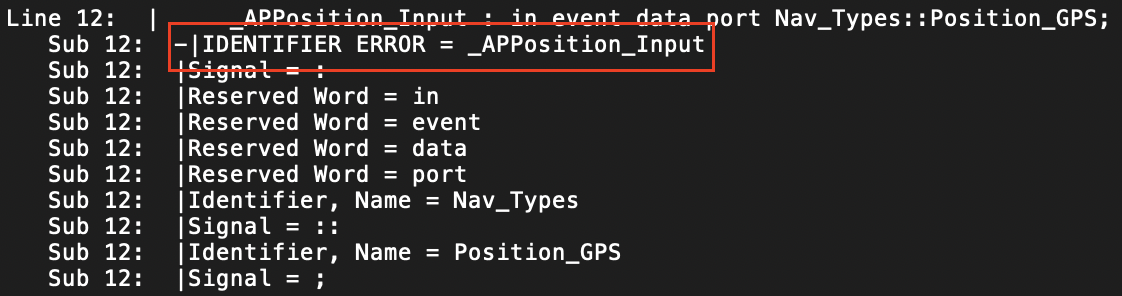
命令行所运行代码和输出如下

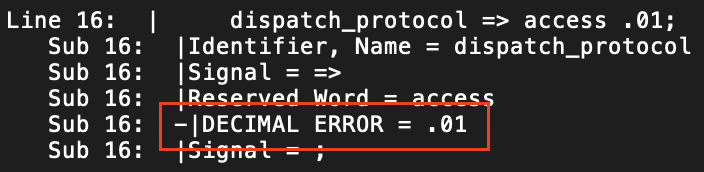


可以看到，输出内容显示，在test2.txt文件中检测到5个错误。



可以看到，在文件一开始，依然是执行文件名称及程序运行的时间。在下面的内容中，我们可以看到Line 1的第二个sub、Line 3的第一个sub和Line 5的第一个sub，都是以“-|”符号进行开头，这是标志着发现错误的意思。此处，已经找到了5处错误中的3处，接下来来找剩下的2个错误。





以下按顺序分别说明一下这五个错误的情况：

第1个错误：Thread#在这里是作为标识符出现，标识符后面不能跟“#”符号，因此是标识符错误，给出了与之相应的“IDENTIFIER ERROR”的表示，并且在前面已经标出了“Sub 1:”的字样，因此也表示了错误出现的行数，是在第一行。

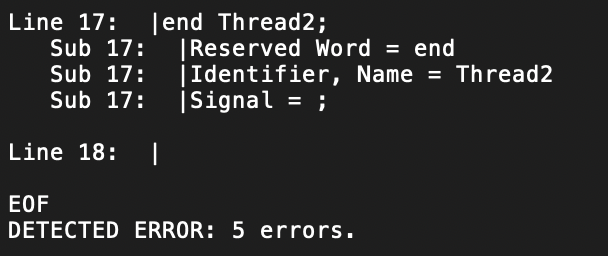
第2个错误：\_APPosition\_Input在这里是作为标识符出现，标识符前面要以字母开头，不能以“\_”符号开头，因此是标识符错误，给出了与之相应的“IDENTIFIER ERROR”的表示，并且在前面已经标出了“Sub 3:”的字样，因此也表示了错误出现的行数，是在第3行。

第3个错误：1flow1在这里是作为浮点出现，数字后面不能够出现字母，因此是浮点数错误，给出了与之相应的“DECIMAL ERROR”的表示，并且在前面已经标出了“Sub 5:”的字样，因此也表示了错误出现的行数，是在第5行。

第4个错误：与第2个错误相同，在此不作赘述。

第5个错误：.01在这里是作为浮点数出现，根据实验要求，浮点数只能以数字开头，不能以“.”符号开头，因此是浮点数错误，给出了与之相应的“DECIMAL ERROR”的表示，并且在前面已经标出了“Sub 16:”的字样，因此也表示了错误出现的行数，是在第16行。

综上所述，满足实验中的错误行数和错误类型的要求。



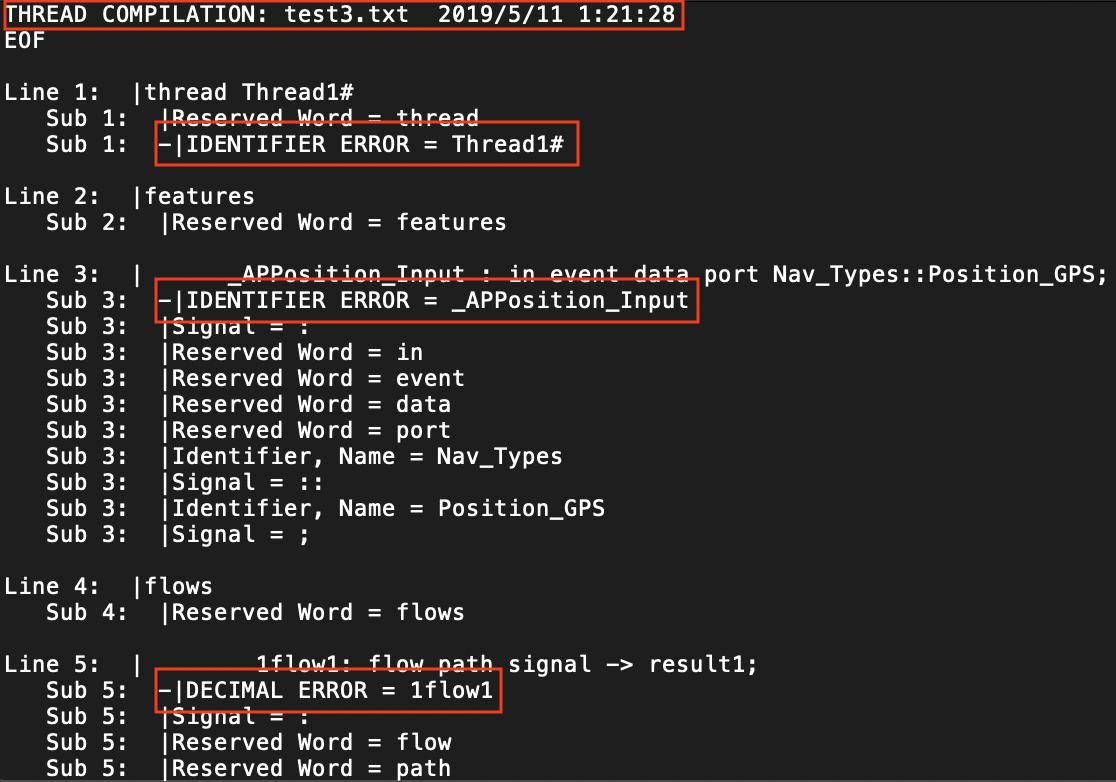
文件的最后，给出了词法分析结束标志和识别到的错误总数。

1. 使用测试文件test3.txt

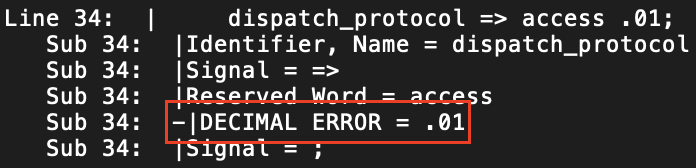
命令行所运行代码和输出如下



输出内容显示，在test3.txt文件中也同样检测到5个错误。

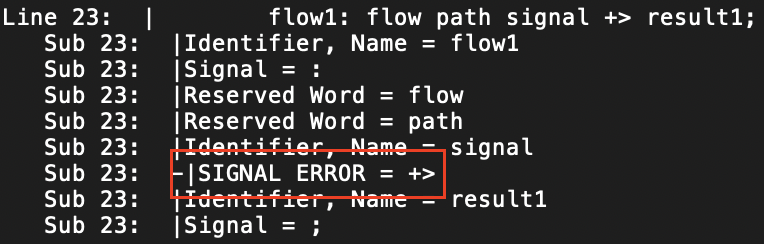


可以看到，在文件开始，依然是执行文件名称及程序运行的时间，可以清楚地看到所识别的文件是text3.txt。



以上标出的四个错误与test2.txt中的错误一致，此处不作赘述。

以下是在test3.txt中检测到的之前没出现过的错误。



+> 在这里是作为专用符号出现，但八个专用符号中并没有“+>”符号，与其相似的有“+=>”和“->”，但都不是“+>”自身，因此是符号错误，给出了与之相应的“SIGNAL ERROR”的表示，并且在前面已经标出了“Sub 5:”的字样，因此也表示了错误出现的行数，是在第5行。

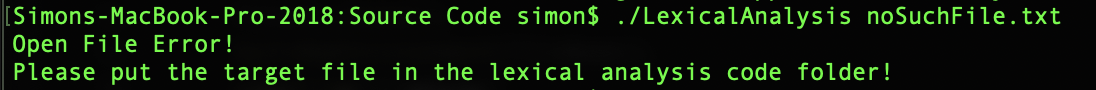
自此，实验所提供的三个测试文件已经执行完毕，并都实现了词法分析的功能，通过了测试。

1. **自身设计的测试项目**

在这里，笔者会提供一些实验提供的测试文件中所没有涵盖的测试项目。

测试一：不存在所要执行的文件

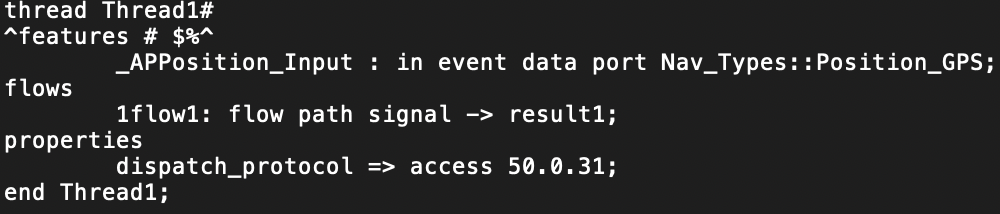
命令行所运行代码和输出如下



命令行直接输出报错，并给出解决方案，观察此时的Source Code文件夹，此时没有tokenOut.txt生成，测试通过。

测试二：实验要求的补充测试

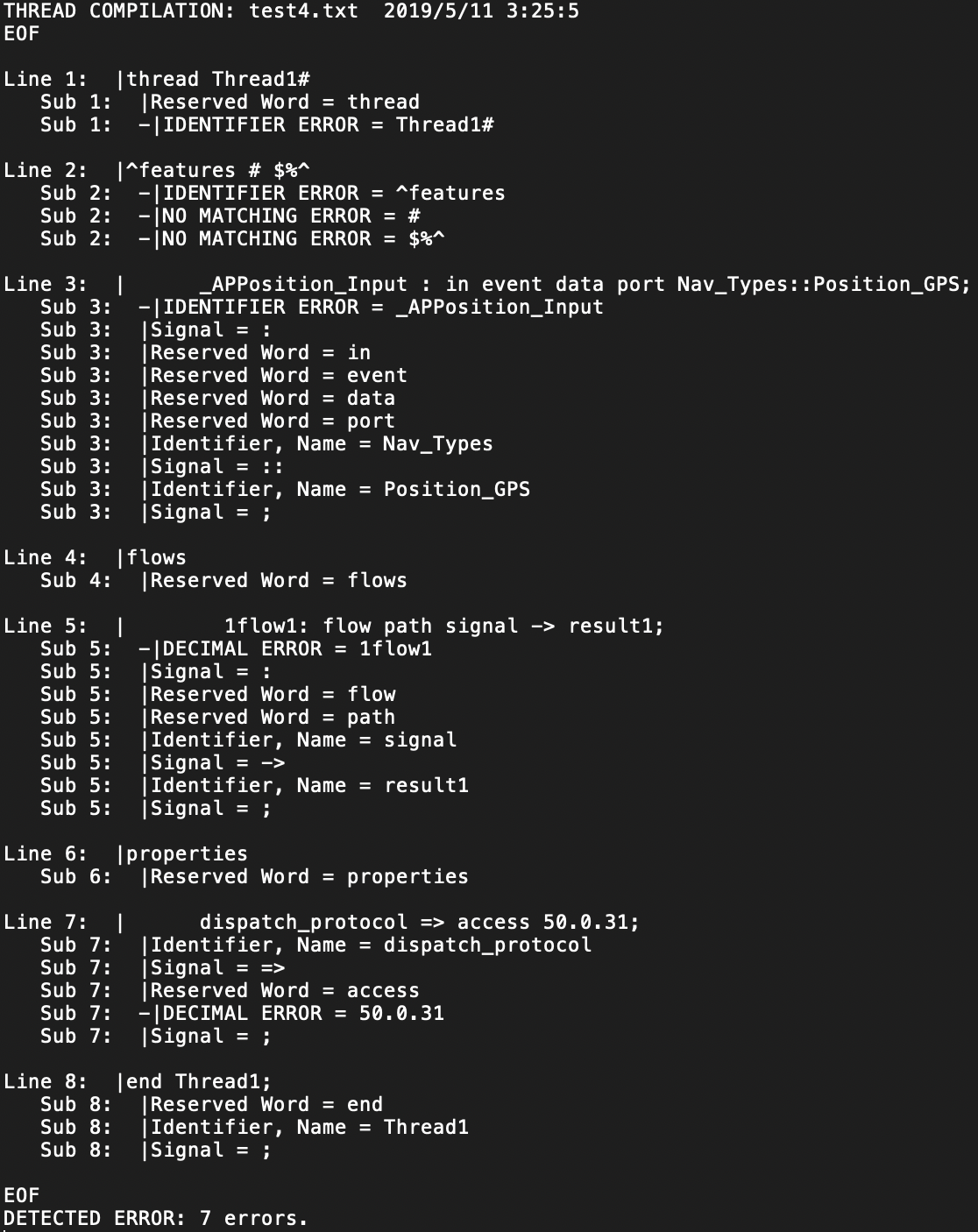
制作test4.txt文件如下



命令行所运行代码和输出如下



共发现7个错误。



其中有新的“#”“$%^”识别过程中的NO MATCHING ERROR，意味着根本在状态机一开始就得不到识别。另外还有^features、50.0.31等典型的错误测试。综上，测试通过。

**四个测试文件和其所生成的tokenOut.txt文件会在“词法分析程序实验提交/附件/测试文件和测试结果/testX”给出，X有1、2、3和4，意味着不同的测试用例。**