实 验 报 告

评分：

计算机 系 14 级 学号 PB14011026 姓名 熊家靖 日期 2017-1-13

实验题目：

- MP 1：使用 Flex 和 Bison 生成 Cool 语言的词法分析器 lexer 和语法分析器 parser，要求语法分析器具有基本的错误恢复功能
- MP 2：实现 Cool 语言一个子集的 LLVM 代码生成器 cgen

综合分析：

- MP 1：

☐ **MP 1.2：比较简单，直接对照 Cool 的文档把词法规则按照 Flex 语法录入就好了**

 # 所遇困难：

字符串的处理需要涉及到 Flex 的一些开关功能，稍微有点复杂

 # 解决方法：

只要认真阅读助教提供的 Flex 的官方说明文档，就能在里面发现对 Flex 开关功能的解释，而且说明文档非常良心地还给开关功能配了一个字符串处理的例子，简直业界良心，所以只要仿照那个例子写就能搞定了！

☐ **MP 1.3.1：按照 Cool 给的语法规则以及优先级定义，将其输入 Bison 即可**

 # 所遇困难：

最大的困难大概就是支持库比较庞大，而实验要涉及到的函数调用非常多，如果支持库中的函数不看懂的话，在规约成功后根本不知道要使用哪个函数建立语法树结点

 # 解决方法：

最笨的办法，耐着性子看，每看懂一个函数，及时添加一些注释，然后发现看了几个之后，其实是能发现一些规律的，以至于最后只要看到函数原型中的变量的类型，就能猜出这个函数到底是用来干嘛以及该怎么用了，这主要还是支持库的变量名都取得很用心，想了想，如果是自己写这么大的程序，那最后的取名可能乱七八糟，值得反思

☐ **MP 1.3.2：在某些地方添加 error 标记，然后必要的时候用 yychar 把吞掉的字符写回**

 # 所遇困难：

因为需要做错误的精细划分，那么就可能遇见连续几个位置都存在错误，比如一个成员属性正确的声明为 a : A，如果声明为 A : a 的话，则程序最好能够指出两边存在的错误（比如左边应该使用小写字母代表的属性名，但是碰见了大写字母代表的类名，而右边应该使用大写字母代表的类名，但是碰见了小写字母代表的属性名），而不只是简单地说这个定义不合法，但是为了实现这个机制，就要求在规约的任何时候对于错误都是敏感的，但是 Bison 机制规定，遇到一个错误后，存在一个简短的恢复期，在这个恢复期内遇到的错误都将被忽视，于是在检测到左边的错误后，程序不会再检测出右边的错误

 # 解决方法：

因为我想要完成的就是一旦检测到错误，就能立即执行 yyerrok，相当于在规约的过

程中添加命令，然后机智地想到了书上介绍过的标记非终结符的方法，只要把每一个规约式都添加一些标记非终结符，然后在完成标记非终结符规约的时候进行 yyerrok，就大功告成了

 # 所遇困难：

在检测程序健壮性的时候，我自己构造了很多小程序，然后调试出了很多的 bug 并一一解决了，直到在构造 if 语句的时候，if 的条件部分，如果是一个等于判断，如果等号左侧是一个属性值，而右侧是一个错误的表达式，那么右侧的错误可以被成功报告，然后错误检查可以继续进行，但是如果等号左侧是一个错误的表达式，那么就会造成整个程序的宕机，因为 Bison 非常不好调试，所以我采用输出调试的方法，在每一步进行规约后都输出规约结果，经过常时间的调试检查，我发现问题在于，Bison 在我觉得它应该执行 yyerrok 的时候，并没有执行它，导致了后续运行跳出了我的控制范围

 # 解决方法：

我查阅了 Bison 相关文档，并尽可能留意了有关错误恢复的介绍中的每一个细节，然而并没有出现与我的认知相悖的情况，换句话说，这些原理介绍都是符合我对 Bison 的理解的，但是上述问题仍然得不到解释，之后与助教进行了一次简短的讨论，向助教反映了具体的情况，助教也有一些疑惑，未能给出答复，虽然后来又花费了整整一个周末的时间进行调试，并请同学帮助分析，但是仍然没有解决上述问题，最后只得作罢

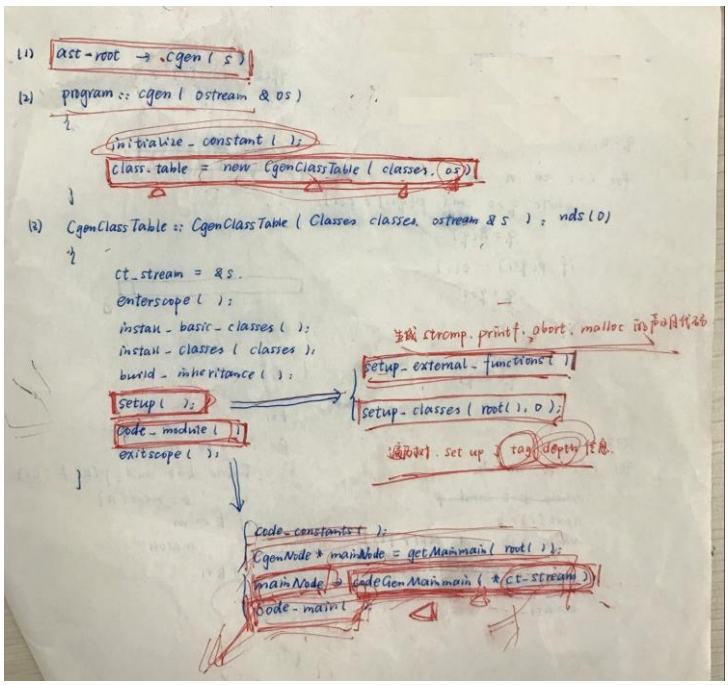
- MP 1：按照助教给的实验攻略，一步一步完成，将输出调整到与 std 基本一致

 # 所遇困难：

这个实验是真的难，虽然整体框架已经全部给出，但是支持库异常庞大，且没有任何说明文档，需要花费大量的时间硬着头皮去看源码，然后理清各种文件之间的关系以及每个文件内定义的各种东西函数的使用方式，而在不动手写的情况下，又不可能完全理清这些东西，可问题是，不理清一些，又不知道如何开始动手

 # 解决方法：

硬着头皮读了一天的支持库的文件，读完了最重要的关于 operand 和 printer 的支持代码，然后理清了整个文件之间的依赖关系，搞清了程序的运行逻辑：（附一张手稿）



然后自己写完了用来调用 main 函数的代码，实现过程大概类似于：按照自己的理解写，然后对照自己的输出与标准输出，基本可以定位出是哪一句话的问题，如果自己看出来错误了，就订正，否则就是某些类型使用不得当，那么就多尝试几种（比如到底是指针的数组还是数组的指针之类的），直到最后与标准输出相符。在完成这一步后，开始对 main 函数中的各种语句的代码生成，然而进展非常艰难，只能先阅读同学们在群里面分享过的 github 的源码，然而边读边仿照着来写，实测发现他们的某些支持库与我们的存在一些差别，直接移植连编译都无法通过……

整体反思：

本学期编译原理课程中，亲自动手实现了一个略微有点寒酸的小型编译器，虽然为此付出了巨大的代价，但是收获确实不匪：

1、 第一次在别人写好的框架下工作，相当于学习了如何与他人协作完成一个工程
2、 学会了 Flex 和 Bison 的基本使用方法，听说这两个玩意儿以后都还挺有用的
3、 更加深入理解了编译器的运作方式，这跟只上课只看书的效果肯定是存在巨大差别的
4、 学会了很多 Linux 相关知识，顺便学会了 Git 的基本操作方法，以后又多了一个工具了
5、 阅读了大量英文文献，也阅读了大量的源码，再也不怕看一大堆工程方面的说明了

然而，这门课程的实验还是不够尽善尽美，经过与同学探讨，发现课程实验的整体规划与协调方面还存在以下问题：

1、 由于我院对于面向对象的介绍太少，包括我在内的部分学生刚开始接触实验时，对于面向对象的编程知识几乎为 0，C++中大量特性完全不了解（包括继承、智能指针等等），而课程的实验感觉完全没有照顾到我们这部分同学在这方面知识的欠缺，显得很不友好
2、 实验的部分说明文档可能还是不够详细，很多时候其实只要再多说一句话就能把事情交代得更加清楚，但是往往因为少了关键的一句话，大家可能要多付出相当多的时间，而这部分时间对于编译课程的增益并不明显
3、 实验的目标规划还是不够精细，容易让人在刚开始实验的时候，望着一大堆的资料和一个给定的目标茫然失措，不知道从何开始，往往要在找到方向上耗费大量时间

基于以上，我对于本课程实验有如下建议：

1、 文档阅读作为单独的 ddl 提出来，要求大家在指定时间内阅读完某一个文档（比如 Cool-Manual），并且提交阅读笔记，这样大家都会认真完成这个文档的阅读，并且整理出该文档的要点，能为之后做实验的时候提供巨大的方便。之所以需要将该 ddl 单独提出来，原因很简单，当一次实验中涉及到大量文档阅读的时候，大家一般都会有一点茫然，然而在每个文档中捡重点来读，然后按照拼凑出来的不完整的理解体系做完了实验，然后就以为自己已经熟悉了这个文档的相关内容，可是最后真正影响后面实验进度的，可能就是因为错过了这些文档中的某些关键性提示语句！
2、 在进行 Cool 语言的词法分析之前，让大家按照要求使用 Cool 语言写一些程序并且提交，确保大家真正熟悉了 Cool 语言的一些语法规则，能为后面的工作省下不少事！虽然本学期的说明文档中，助教也要求了大家自行写一些 Cool 的程序，但是完全没有把它列为一个目标，这在一篇充满了大量信息的说明文档中，很容易被大家给忽略到，不能认识到这项要求的重要性，以至于之后的路走得很不顺！总而言之，中心思想就是：任何推荐完成的事情，都作为一个子目标，单独提出来，让同学们在一定的时间内完成，有目标的驱动，真的会事半功倍！

3、 真的可以考虑在每个实验前开设一次讲解课程，让助教系统地介绍一下本次实验需要完成的事情，以及需要注意的事项，并且简单地带领大家过一遍可能要使用到的支持库代码，给大家一个比较明确的方向，不至于每次实验开始前的几个小时，都处于茫然失措的状态，不仅浪费了大量时间，也消磨了很多学习热情，因为作为过来人回头看过来，这学期的实验最困难的地方都在于去揣测自己到底该干什么，而不是去思考自己到底该怎么干，往往明白第一个问题后，第二个问题解决起来就已经很连贯了，而第一个问题的解决，对于编译课程的增益，真的相当有限，这完全是助教在一开始就能够带领大家弄清楚的事情，完全没必要让大家在这上面耗费更多时间

4、 在进行大量的 C++代码的阅读之前，最好是能够抽出一周的时间，安排大家学习 C++的相关知识，并且通过布置一些小作业的形式，将任务目标化，在大家整体对于 C++有了一个更加深入的了解后，不管是之后的代码阅读，还是课程本身会涉及到的面向对象的编译技术，都能够事半功倍

5、 总而言之，希望这门课程的实验在文档方面能够为大家提供更加详细的指导，在任务规划方面能够将一个个大目标分成一个个小目标，将目标分散到更短的时间内，减轻大家每一次任务的负担，但是又能更加出色并且全面地完成每一项任务，全面提升实验效率

参考文献：

[1] CoolAid: The Cool Reference Manual

[2] A Tour of the Cool Support Code

[3] flex and bison

[4] Kaleidoscope: Implementing a Language with LLVM

[5] LLVM Language Reference Manual

[6] LLVM Programmer’s Manual

[7] github-CharlieMartell