MP1, MP2 Report

总述

- MP1+MP2实现了cool语言的词法分析+语法分析+中间代码生成，即实现了cool语言子集的编译器前端。
- 其中在MP1中实现了cool语言的词法分析与语法分析，MP2中实现了cool语言子集的中间代码生成。
- 实验主要包含以下几个方面
    - 使用flex把输入的字符流转化为token流
    - 使用bison把token流转化为AST
    - 利用AST生成LLVM IR

问题与对策

MP1

- MP1.1
    - 回答六个问题，只需要看一下list, string table的代码。
    - 遇到问题
        - 对std::list了解不够深，对策是查资料(STL源码剖析)结合看代码(stl_list.h)
- MP1.2
    - 实现词法分析器。
    - 遇到问题
        - 正则表达式表达能力有限，对于前后匹配类型的表达式，只用正则是没法表示的，因此需要使用start condition
        - 实现的词法分析器没有通过所有测试，原因如下
            - 没有考虑周全，在出现类似"*"+)的表达式时，没有处理，导致最后出现bug
            - string存在1025的长度限制，但是我把这个限制同时加在了id上，导致没有通过长度大于1025的id测试
- MP1.3

- 实现语法分析器
- 第一次提交
    - 处理正确的程序
    - 遇到问题
        - let语句行号与reference-binaries里的parser产生的不同
            - 在没有对let语句中的临时变量赋值的情形下出现
            - 原因应该在于，如果没有赋值语句，在赋值处没有进行规约，导致行号与let语句结束时一致
            - 因此只需要修改规约方式，赋值或者不赋值都会规约，就可以使行号与reference-binaries里的parser一致
- 第二次提交
    - 错误处理
    - 遇到问题
        - 存在考虑一方面但另一方面仍有问题的情况
        - 例如下面的情况

```
class a{};
A{};
class a{};
```

        - 如果写成下面这样就能同时检查出这三行都有错

```
class : CLASS error ';'
       { yyerrork; }
```

        - 如果写成下面这样，中间一行就会被吃掉，只报第一和第三行错。

```
class : CLASS error
       { yyerrork; }
```

        - 但也不是说上面那种就完美了，像下面这种情况，如果照

上面的写法，就会一直吃到第二行最后的分号，导致第二行的class其实有错但没有报

```
class a <- {}
class a {};
```

- 我觉得这是一种tradeoff，为了尽可能多报错误而导致报错出错，或者为了不出现报错出错而少报错，都是不好的。其实只要控制有一定数量的报错，让程序员能够快速找到错误的地方，提高效率即可。

MP2

- 实现中间代码生成
- 遇到问题
  - ValuePrinter类虽然说提供了丰富的接口，但是有的接口并不那么好用。
    - 比如说make_fresh_operand，利用它可以生成名为vtpm.\d类型的变量名，但是后面的数字是静态变量，且那个文件不能改，有时需要的调整无法进行。
    - 打印有些东西时候会出现多余的换行。
    - 解决方式是有时候直接弃用ValuePrinter的某些接口，使用CgenEnvironment::new_name和仿造ValuePrinter造的方法make_main_operand来获取变量名。
  - cond_class的code函数一开始不知道怎么实现，因为不知道如何在执行分支之前得到if表达式的值类型。直接把全程变为std::stringstream，在执行一个分支后进行流回退，最后再把std::stringstream的内容转为std::string输出
    - 后来发现其实可以利用宏Expression_EXTRAS里面定义的函数get_type获得Expression的类型
    - 我的做法的缺陷就在于程序大的时候疯狂吃内存。
    - 可以知道cool-tree.handcode.h的主要作用就是可以通过宏定义，在不改cool-tree.h的前提下修改各个AST节点的定义
  - let语句是唯一有对ID赋值与ID语句的地方。因此开始检查的时候有所

疏忽，最后一次提交修正的好几个bug都出在这里，有了变量之后会出现额外的store等操作，而这是外面语句所没有的，有可能外面检查不出的顺序问题都出在这里。这反映出我构造测试例还是不够认真。

        - 其中有几个问题，let语句中的ID可赋初值也可不赋(其实还是要根据是Int还是Bool赋值0或false)，还有进入作用域和出作用域需要注意``var_table`。
    - 对method_class的code函数实现做一定处理，使得其更具有拓展性和通用性。由于本次MP2只需要实现一个Main_main方法，参数为空，返回值为Int。但是如果以后实验要调用method_class的code函数则不一定满足这几点了。所以对method_class的处理需要具有通用性，加入参数列表，检查返回类型，生成函数名(生成方式现在看来是类名+下划线+方法名)。

核心问题、设计、实现设计

- MP1
    - 需要掌握flex与bison的基本使用。
    - 按照先易后难的顺序
        - MP1.2先实现除多行注释和字符串外的其他表达式，再利用start condition实现多行注释和字符串
        - MP1.3先实现除let外其他表达式，最后再实现let
- MP2
    - 首先理清MP2的执行过程
    - main函数位于cgen-phase.cc，调用cgen函数利用parser(semant)的输出一步步生成代码
    - cgen构造CgenClassTable对象，CgenClassTable开始构造对象，总共有两遍的过程，实际上我们只要完成第二遍的代码生成即可
    - 代码生成调用code_module，进一步会生成main函数与Main_main函数
    - 代码生成的过程就是调用各个表达式类的成员(其实也是各表达式类)的虚函数code，由于都是指针，执行时根据不同表达式类的动态类型自顶向下调用不同的code函数自底向上生成代码
    - 所以可以知道，MP2的主要任务
        - 实现main函数和Main_main函数

        - 实现各种表达式类的code函数
    - 其中有一部分难点就在于能够清楚知道如何准确地调用各种API来很好地实现。
    - 所以也是先易后难，先实现常量、算数运算等较为简单的，最后实现let，if，while等语句

参考资料

1. 候捷. STL 源码剖析[J]. 2002.
2. Moo B E, Lajoie J. C++ Primer[J]. 2012.
3. Flex manual
4. Bison manual(https://www.gnu.org/software/bison/manual)