**第一次检查要求**

**1. 要求综述**

  第一周要求实现racket计算器的基本架构。要求掌握[S-expression](http://en.wikipedia.org/wiki/S-expression)的解释过程，并实现基本的十进制分数和小数的+, -, \*, /运算。

**2. 需实现类**

  本周要求大家实现Float(包括浮点小数，在double范围内，保证所有有关此类型操作均在double范围内), Rational类(包括整数int和分数Fraction)（Rational类要求高精度计算，高精度，数字长度无限制，建议String或动态数组结构来实现）。

**3. 需实现函数**

  十进制整数，分数以及小数的加减乘除运算。

**4. 特殊要求**

a. 输入输出要求

需支持如下格式输入：

整数，分数(X/X)，浮点数(X.X，XEX).

注意：具体请参考指导手册 Lexical structure 章节内容。但不要求实现如下内容：非十进制数，#i, #e, X@X形式复数，#、\*等通配符。之后测试要求不变。

  测试的数值的范围为Float: double以及Rational:无范围限制。注意分数输出时应该为最简分数。注意得到的Float类结果同样保证在double范围内，但不保证参与运算的Rational的范围。

e.g(第一行为输入，第二行为输出):

(+ (- 2 5/2) 1)

1/2

b. 疑难点详解

  整个程序的运算的主要逻辑在calc\_exp()中实现。calc\_exp()的作用在于将一个表达式中的内容计算出来并返回一个Num的指针，指向最终的计算结果。于是，通过递归调用calc\_exp()，就可以对整个表达式进行解析并计算。  
  在calc\_exp()中，首先通过next\_token()获得一个token(一个token是指一个语法单元)，token的内容存在一个字符串内。“(”意味着语句的开始。  
  接下来再调用一次next\_token()。  
  若得到的是一个操作符，则根据该token的内容实例化一个操作符的类。然后，依次调用calc\_exp()计算各个操作数的值，并通过一个叫做Cons的链表存储各个操作数的值。Cons的car为这个单元内的值，cdr是指向下一个操作数的值的指针。cons为头结点的指针，tail为尾指针。  
  在计算好了各个操作数的值之后，就可以通过将Cons中的内容作为之前实例化好的操作符类的参数来计算最终的结果（注意计算之后，将Cons中的内容清空）。  
  若next\_token()得到的内容就是一个数值，那么直接返回用该数值构造的Num的对应子类。  
  最后，若next\_token()获得一个“)”则标志表达式的结束。  
  接下来通过(+ (- 2 3) 4)作为例子进行讲解。  
  首先，next\_token()得到一个“(”，表达式开始。然后，next\_token()得到一个操作符“+”，实例化一个Add类。接下来重复调用calc\_exp()，第一个calc\_exp()计算了(- 2 3)的值，结果为一个Int类，值为-1。第二个calc\_exp()计算了4的值，结果为一个Int类，值为4。计算到“)”的时候返回NULL，结束。计算的同时，将操作数结果存为NULL->-1->4(NULL为头结点)的一个链表cons。将cons放入Add的calc方法中计算，得到最终的结果为一个Int类，值为3。于是calc\_exp()返回该Int类。在main()中，这个Int类的值被打印出来，作为输出。  
  整个程序的框架就是这个样子，具体的函数和类是如何实现的请自行阅读代码。最后，完成缺失的Frac类和缺失的各个函数。

同时可参考demo程序框架及代码。