# TOY\_GREP总结

# 实现原理:

实现基于龙书3.9节，使用NFA/DFA实现简单正则表达式的搜索。

简单的表达式仅包括括号，星号和或表达式。

基于C++实现，使用了vector。

# 表达式解析:

使用类似LR(0)自底向上解析方式（无状态转换表）。词法分析主要获取当前字符的属性，并处理转义字符，后续可以增加对范围的模糊匹配。对于操作符与操作符之间的可忽略问题在词法阶段解析，如a|（或和结束符），(a|)，a||，a\*\*，对于操作符出现位置的异常，在语法解析阶段处理，如\*开头，|开头，避免双栈式的复杂场景。

# 心得：

1. 数据结构：
   1. 树：最多二元的语法树
   2. 栈：简单实现的双栈，用于解析表达式
   3. 图：根据语法树计算的属性值构造出来的状态转换图
   4. 集合：简单的集合用来表示图的边，本打算用位图实现，但是由于位图不支持动态开辟大小，因此还是简单基于vector<bool>进行实现。
   5. 注：原本使用C实现，没有bool，因此使用链表处理集合。
2. 使用双栈式解析正则表达式，为正则表达式构建语法树。
   1. 两个栈拥有类似操作，进栈，出栈，显示栈顶，空栈判断，栈的显示，构造与析构，此外，由于操作符栈需要对优先级判断，根据优先级决定栈是否需要调整操作，而符号栈不需要调整，因此与基本的栈有些许不同，因此使用一个模板类和向量容器来实现一个虚栈类。
   2. 栈是否需要调整直接使用>=判断，因此对于具体的实现过程，需要重载运算符。
   3. 注：模板类中使用迭代器时候需要在迭代器声明的时候加上typename。Typename指定后面vector<T>::iterator是一个数据类型。详细见（<http://blog.csdn.net/acidgl8757/article/details/17317303>）
3. 树的遍历：三种遍历方式
   1. 中序遍历：输出整棵语法树的时候使用的其实是中序遍历，先输出左儿子，再输出自己节点，最后输出右儿子。
   2. 后序遍历：在由实体节点计算连接节点（CAT节点，或节点，星节点）时候，从整棵语法树的根开始，计算一个节点之前先计算该节点的两个子节点的属性，再计算该节点属性。
   3. 先序遍历：根据nullable、first\_op、last\_op计算每个节点的follow\_op时，是根据该节点左右孩子及自己节点的属性进行计算，再计算各个孩子。（其实该计算并不依赖于遍历方式）
4. 使用Valgrind对内存进行检查。
5. 状态机的算法实现
6. 疑似：根据状态转换图进行匹配的时候，使用的是深度优先搜索？
7. 学习autotools的使用，使用autotools进行构建。

# 未完待续：

1. 将NFA转换成为DFA。
2. 最小化DFA的状态。
3. 集合是否有更好更合理的实现方案
4. 是否有更好更合理的设计模式